# Summary of Knowledge and Experiments for 1-Steiner, 1-Hanan

***Abstract*—This document summarizes "what we have done" and "what we know" regarding deep learning for the rectilinear Steiner minimum tree problem.**

## I. Terminology

TABLE I: Terminology

| | |
|---|---|
| $c(X)$ | Length of RMST over pointset $X$ |
| PR | Performance ratio. $\frac{\text{c(RL agent solution)}}{\text{c(solution of [1])}}$ |
| $\text{PR}_{random}$ | PR when choosing actions randomly |
| $\text{PR}_{RL}$ | PR when using RL model for choosing actions |
| $\text{PR}_{MST}$ | PR when not choosing any action |
| Hanan grid | The grid formed by drawing horizontal and vertical lines through each of the points in $P$ |
| I1S | Iterated 1-Steiner heuristic [1] |
| $P$ | Input pointset which defines the RSMT instance. This set will never change. |
| $S$ | Set of Steiner points. This set is initially empty. This will be our solution set when termination condition is reached. |
| LR | Learning rate of optimizer |
| $n$ | Number of points in a pointset |
| $m$ | Number of pointsets in a given (training, testing) dataset. |
| $L$ | Sampling range of $x, y$ coordinates for the points. Default = 100 |
| $d$ | Embedding size |
| $\texttt{distance}(i, j)$ | `L_1` distance between vertices $i, j$ |
| $tf\text{-}dqn$ | Model that uses buggy version of DDQN algorithm |
| $tf\text{-}dm$ | Model that uses correct version of DDQN algorithm |

The entire codebase exists in the github repo [17].

## II. Introduction

Given a set $P$ of $n$ points in the Manhattan plane, the *Rectilinear Steiner Minimum Tree* (RSMT) problem seeks to find a minimum-length tree of rectilinear edges which connects all points of $P$. An RSMT can have cost (i.e., total edge length) less than the cost of the rectilinear minimum spanning tree (RMST) over $P$, through the introduction of *Steiner points* that serve as "junction points" to save tree cost. To solve the RSMT problem, we wish to find a set of Steiner points $S$ so as to minimize the cost of the RMST over $P \cup S$. The RSMT problem is of particular interest in VLSI CAD applications, where it underlies wirelength and timing estimation, global routing, and other key physical design tasks [3] [7].

A rich literature on the RSMT problem has spanned well over 50 years. Hanan [19] showed that there always exists an RSMT in which all points of $S$ are chosen from the vertices of the grid formed by passing horizontal and vertical lines through each of the point in $P$. This grid is called the *Hanan grid*. It corresponds to a gridgraph with $n \times n$ vertices as shown in Fig. 1.

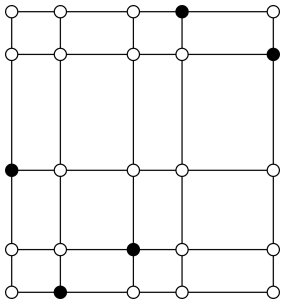

Fig. 1: Hanan grid induced by the black vertices.

Applying simple graph-topological arguments and the triangle inequality – which holds in the Manhattan plane – yields that at most $|S| \leq |P| - 2$ Steiner points suffice to achieve an optimal RSMT. Garey and Johnson [10] showed that the RSMT problem is NP-complete. Hwang [11] showed that the *rectilinear minimum spanning tree* (RMST) over $P$ is a 3/2-approximation to the RSMT. In other words, $c(RMST(P))/c(RSMT(P)) \leq 3/2$ for all pointsets $P$, with this bound being tight.[1]

The 1990 paper of Kahng and Robins [2] proposed the *Iterated 1-Steiner* (I1S) heuristic, which greedily grows the set of Steiner points $S$ to minimize the cost $c(RMST(P \cup S))$. Over the ensuing years, empirically-observed I1S suboptimality with respect to optimal solutions calculated by codes such as GeoSteiner3.1 [9] was variously reported to be in the range of 0.086% to 0.2%. The follow-on work [1] leveraged the *oriented Dirichlet cell* concept of Georgakopoulos and Papadimitriou [8] to achieve an $O(n^3)$ implementation of I1S. Tradeoffs of wirelength minimization for runtime were suggested, e.g., via a "batched" variant of I1S. We note that I1S, unlike many Steiner tree heuristics [6], enjoys a performance ratio strictly less than 3/2. [3] notes a lower bound on performance ratio of 1.3 as established by a construction of [5] and ascribes to Zelikovsky an upper bound of 1.3125 on I1S performance ratio.

## III. Problem Statement

In this work, we seek to train a deep learning model to tackle the RSMT problem.

[1]The tightness of the bound is seen with $P = \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$ (the RMST has cost = 6, while introducing a single Steiner point at $(0, 0)$ achieves RSMT cost = 4).

We have tried the following algorithms for training the deep learning model:

- Double DQN [14]
- Proximal Policy Optimization [15]

## IV. Our Approach

Taking inspiration from [12] as well as from [1], we address the RSMT problem in an RL framework. At each timestep, the RL agent chooses a vertex (ideally optimal) from the Hanan grid to be included in $S$. Let $c(X)$ be the total edge length in an RMST over pointset $X$. The reward $r$ for choosing this vertex as SP and transitioning to new state $S'$ is the change in RMST length, $r = c(P \cup S) - c(P \cup S')$.
We use the graph embedding network $structure2vec$ [13] to featurize each vertex on the Hanan grid in such a way that the context of the overall graph as well as neighbor vertices is captured, and we use $n$-step Q-learning [18] for learning the state-action values. (**NOTE** that the $n$ in "$n$-step Q-learning" is different from number of points in the RSMT instance.)

### A. Graph Embedding

We encode each vertex as an embedding so as to capture the graph information, which can be used for downstream tasks. The graph embedding network takes input as

1) current solution set: This is a vector of binary variables of length $n^2$. Each element corresponds to a vertex of the Hanan grid. If a vertex $\in P \cup S$, then its corresponding element in the vector is $= 1$; otherwise, its corresponding element is $= 0$.
2) adjacency matrix: This is a 2D matrix of size $n^2 \times n^2$. If $i, j$ are adjacent vertices in the Hanan grid, adj$[i, j] = 1$; otherwise, adj$[i, j] = 0$.
3) weight matrix: This is a 2D matrix of size $n^2 \times n^2$. If $i, j$ are adjacent vertices in the Hanan grid, then weight$[i, j] =$ `distance`$(i, j)$; otherwise, weight$[i, j] = 0$.

$structure2vec$ computes vertex embeddings $\mu$ using the following update rule iteratively.

$$\mu_v^{(t+1)} \longleftarrow \text{relu}(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \mu_u^t + \theta_3 \sum_{u \in \mathcal{N}(v)} \text{relu}(\theta_4 w(v,u))) \quad (1)$$

where $\mathcal{N}(v)$ represent the set of neighborhood vertices of vertex $v$, $x_v$ represent a vector of vertex features, $w(v, u)$ is the weight of edge between vertices $v, u$ and the $\theta$s are learnable embedding network weights. We iteratively apply this update rule for $T = 5$ times.

The above-listed features are necessary for finding embeddings. Some extra features have been also been tried as inputs to the network, but they are not helpful in increasing performance so we do not use these anymore as inputs (see Section V). Some of these include:

1) $x$ coordinates: This is a vector of length $n^2$ containing the $x$ coordinates of each vertex on the Hanan grid. Each element in this vector is $\geq 0$ and $\leq L$.
2) $y$ coordinates: This is a vector of length $n^2$ containing the $y$ coordinates of each vertex on the Hanan grid. Each element in this vector is $\geq 0$ and $\leq L$.
3) (adjacency)$^2$ matrix: This matrix contains information about vertices at graph distance = 2 (two hops away) from each given vertex.
4) complete weight matrix: This is a 2D matrix of size $n^2 \times n^2$. For every pair of vertices $i, j$, weight$[i, j] =$ `distance`$(i, j)$.

Using this information, the graph embedding network produces $d$-dimensional embeddings which is used by the Q-network to learn Q values for all state-action pairs. We use the DDQN [14] algorithm for learning these Q values. This graph embedding network also helps with the generalization of the learned model to larger values of $n$ [13] [12].

## V. Experimental Results

Given an input pointset, the model iteratively adds vertices to $S$ till a termination condition is reached. We have tried two termination criteria while training the model:

1) $n - 2$ timesteps termination: The Steiner environment return 'done=True' after $n - 2$ timesteps.
2) Bad action termination: The Steiner environment returns 'done=True' when it has taken an action which results in strictly negative reward.

The model is trained on dataset of size 4000 pointsets while the reported numbers below quantify its performance on 500 pointsets (different from training pointsets).

### A. $n - 2$ Timesteps Termination

**NOTE:** The model uses X/Y coordinates of the vertices as inputs at this point in time.

We started with naively training the neural network using 3-step Q learning. The performance ratios are mentioned in Table II:

*1) What worked:*

TABLE II: Experiments leading to decrease in PR.

| Setting | $n = 6$ | $n = 8$ | $n = 10$ |
|---|---|---|---|
| Naive training | 1.03 | 1.04 | 1.05 |
| 1−Step Q-Learning | 1.023 | 1.034 | 1.04 |
| Eval in "BA Termination" mode | 1.016 | 1.023 | 1.038 |
| Removing X/Y coordinates | 1.009 | 1.021 | TBA |

- 1-Step Q-Learning: Using 1-step Q-learning instead of 3-step Q-learning decreases the PR (Table II).
- Evaluating in a Different Mode: The strategy trained in $n - 2$ timesteps termination mode is evaluated on testing set in "bad action termination" mode, i.e., the model keeps acting till it receives strictly negative reward. This leads to a decrease in PR (Table II). This improvement indicates that model once starts making bad move, its more likely to make bad decisions.
- Removing X/Y Coordinates from Input: We observe that using X/Y coordinates as input to the NN breaks the symmetry of the model. That is, all eight symmetric

orientations of input pointset don't lead to same Steiner tree. To retain symmetry, X/Y coordinates are removed from inputs and this leads to further decrease in PR (Table II).

*2) What Did Not Work:*

TABLE III: Experiments that did not work.

| Setting | $n = 6$ | $n = 8$ |
|---|---|---|
| Buffer size | 1.06 | - |
| Non-sparse weight matrix | 1.025 | 1.03 |
| 2-hop adjacency matrix | 1.012 | 1.026 |
| Decay LR | 1.014 | 1.021 |
| Reward scaling | 1.03 | - |
| Penalize non-positive reward | 1.04 | - |
| Reward normalization; center=True | 1.011 | - |
| Reward normalization; center=False | 1.012 | - |

- Effect of Buffer Size: The best model currently uses a buffer size of $1e^4$. Increasing buffer to $1e^5$ leads to increase in PR (Table III).
- Changed weight matrix elements to: weight$[i, j]$ = distance$(i, j)$. This would give the distance between any two vertices and not just directly connected ones. Increased the PR (Table III).
- To provide information about second-nearest neighbour, tried using (adj)$^2$ matrix as input. The PR remains same (Table III).
- Decaying LR does not help. As the training progresses, the learning rate of optimizer is exponentially decayed for fine optimization from $1e^{-4}$ to $1e^{-5}$ over $5e^5$ timesteps. PR remains the same (Table III).
- Scaling the rewards by a factor of 10. The idea was to increase difference between optimal and sub-optimal rewards and thereby make it "easier" for model to learn. Leads to increase in PR (Table III).
- We observed that the model chooses too many actions with zero reward. To prevent this behavior, rewards are shaped such that if it is negative(including zero), the reward is decreased by a constant value. Leads to increase in PR III).
- Reward normalization is in general a common trick employed in RL models. We batch normalize immediate rewards before feeding them in DDQN algorithm. In one mode, mean batch reward is subtracted while normalizing (center=True), while it not in the other mode (center=False). The PR does not change for both modes III).

*3) Generalization Capacity: 'Upper Triangular' Set of Tests for Best Model:* Models are tested on 500 pointsets with larger $n$ to check generalization capacity.

TABLE IV: Generalization tests for 1-Steiner

| Setting | PR$_{n=6}$ | PR$_{n=8}$ | PR$_{n=10}$ | PR$_{n=16}$ | PR$_{n=20}$ |
|---|---|---|---|---|---|
| $n = 6$ | 1.009 | 1.019 | 1.031 | 1.084 | 1.1 |
| $n = 8$ | | 1.021 | 1.031 | 1.072 | 1.093 |
| $n = 10$ | | | 1.038 | TBA | TBA |

### B. Bad Action Termination

For every set of experiments, we observed that "$n$-2 steps termination" performs better than "bad action termination". So, we decided to stick to "$n$-2 steps termination" training mode.

The performance of models with their generalization capacities are shown in the table below.

TABLE V: Experiments for "Bad Action Termination"

| Setting | PR$_{n=6}$ | PR$_{n=8}$ | PR$_{n=10}$ | PR$_{n=16}$ | PR$_{n=20}$ |
|---|---|---|---|---|---|
| $n = 6$ | 1.030 | 1.050 | 1.070 | 1.110 | - |
| $n = 8$ | | 1.045 | 1.056 | 1.090 | 1.108 |
| $n = 10$ | | | 1.070 | 1.090 | 1.106 |

### C. Observations and Insights for Steiner Problem

1) Value estimates for the DDQN algorithm are highly overestimated as shown in Figure 3. For a fixed set of states, the DDQN state value estimate goes more than 60. The actual state value estimate is around 16. The interesting part is that $tf\text{-}dqn$ does not overestimate the states value estimates. Changing network capacity does not seem to help with this overestimation bias. This overestimate bias seems to be the reason of reduced performance of $tf\text{-}dm$.
2) For PRs mentioned above, the model has 6 FC layers and with each hidden layer size 128. Increasing the model sizes to 10 FC layers and 256 hidden size does not change performance. Its surprising that increasing model capacity to such extremes does not help in achieving a train PR of 1. Usually in supervised learning settings, model can overfit on training data given large enough model.
3) Normalizing the input weight matrix in DQN algorithm does not have any effect on performance. This is rather surprising because using normalized weight matrix in PPO is very helpful.
4) The model is fully symmetric. That is, all eight symmetric orientations of input pointset lead to same Steiner tree by RL model.
5) Using the standard 1-step Q learning instead of 2- or 3-step Q learning proved to be very beneficial. Current best model uses 1-step Q learning with "$n-2$ timesteps termination" training.
6) The current model has some generalization capability right now. Model trained on dataset with $n = 8$; PR$_{RL}$ $= 1.023$ is tested on dataset with $n = 16$ gives a PR$_{RL}$ of 1.078.
7) The RL model has a tendency to choose a large number of actions which lead to zero reward. Even when the RL model is able to find the optimal tree, the number of Steiner points added is much more than the optimal Steiner points found by Iterated 1-Steiner.
8) To facilitate better learning, we tried batch normalizing the immediate reward values before passing them to DQN algorithm. The performance remains same.

### D. Take Away Last Steiner Point Experiment

For 2000 pointsets, we run Iterated 1-Steiner algorithm and and take away the last Steiner point from each tree. This grid graph is not used as training data and model is trained to see if it can put these back.

TABLE VI: Take away Steiner point experiments

| Setting | $PR_{n=6}$ | $PR_{n=8}$ | $PR_{n=10}$ | $PR_{n=16}$ | $PR_{n=20}$ |
|---|---|---|---|---|---|
| $n = 6$ | 1.017 | 1.013 | 1.011 | 1.0067 | 1.0053 |
| $n = 8$ | | 1.011 | 1.01 | 1.006 | 1.0050 |
| $n = 10$ | | | TBA | TBA | TBA |

TABLE VII: Take away Steiner points $PR_{MST}$

| | $PR_{n=6}$ | $PR_{n=8}$ | $PR_{n=10}$ | $PR_{n=16}$ | $PR_{n=20}$ |
|---|---|---|---|---|---|
| $PR_{MST}$ | 1.034 | 1.02 | 1.014 | 1.007 | 1.0054 |

From Table VI it might seem that generalization PR gets better as $n$ increases. But its only because the reward for placing the last Steiner point becomes smaller as $n$ increases (as can be seen from Table VII).

The hope was to get PR very close to 1 since this is a much easier problem compared to full 1-Steiner.

### E. Some Comparison Stats with Iterated 1-Steiner (I1S)

We choose the best model (see Subsection V-A3) we currently have for this comparison.

1) These numbers tell us how the RL model fares against I1S algorithm. For example, the RL $>$ I1S column gives the percentage of instances where the RL model gives a lower-cost solution than I1S.

| Setting | RL $>$ I1S (%) | RL $=$ I1S (%) |
|---|---|---|
| $n = 6$; $m = 4000$ | 3.6 | 64.2 |
| $n = 8$; $m = 4000$ | 5.8 | 36.4 |
| $n = 10$; $m = 4000$ | 4.0 | 12.6 |

2) RL model can lose to I1S at most by:

| Setting | Worst RL loss (%) | Best RL win (%) |
|---|---|---|
| $n = 6$ | 13.04 | 6.09 |
| $n = 8$ | 16.04 | 4.5 |
| $n = 10$ | 15.0 | 5.99 |

## VI. Current Hurdles

1) All of the results cited above are achieved by training the model using 'buggy' DDQN algorithm. The difference is that instead of freezing target network and training only the main network, the loss gradients flow through both main and target networks. We will call this $tf\text{-}dqn$ model from now. The model which uses correct implementation of DDQN will be called $tf\text{-}dm$. One of the worrying problems right now is the increasing loss while using $tf\text{-}dm$ model as shown in Figure 2. The PR for $tf\text{-}dm$ is lower than $tf\text{-}dqn$.

| Setting | $PR_{tf\text{-}dm}$ | $PR_{tf\text{-}dqn}$ |
|---|---|---|
| $n = 6$; $m = 2000$ | 1.012 | 1.01 |
| $n = 8$; $m = 2000$ | 1.036 | 1.025 |


Fig. 2: $tf\text{-}dm$ loss plot

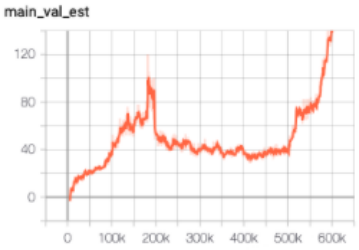


Fig. 3: $tf\text{-}dm$ values estimate

The actual average value estimates for these states is around 17. Shows that overestimation bias exists.

2) Looking at the behavior of the model, we observe that the RL model is highly likely to choose vertices which are adjacent to $P \cup S$ as Steiner point.

## VII. Back to the Basics; 1-Hanan Neighbor Problem

[16] shows that GCN-based embeddings cannot recover shortest path distances between vertices, which might be the reason we are not able to find the optimal solution. They propose a novel approach in which messages are aggregated from *anchor sets* which are randomly chosen subsets of all the vertices. The message from each vertex includes distances that reveal vertex positions as well as feature-based information from input vertex features. Each dimension of the output embedding encodes the necessary information about vertices that reside in different parts of the graph.

Taking inspiration from the above approach, we use the original points as our anchor set and update node embeddings by aggregating messages from Steiner points as well. We perform the overfitting experiment on $n = 6; m = 20$ and full grid 1-Hanan neighbor setting by calculating the nodes embeddings in the four following way. In this experiment, we consider choosing just one Steiner point over the full Hanan grid and do not deal with the complete 1-Steiner problem.

1) We use Dai's [12] embedding network as our base model. The node embeddings are found by iteratively updating:

$$\mu_v^{(t+1)} \longleftarrow \text{relu}(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \mu_u^t + \theta_3 \sum_{u \in \mathcal{N}(v)} \text{relu}(\theta_4 w(v,u))) \quad (2)$$

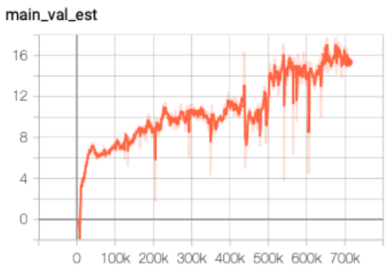


Fig. 4: $tf\text{-}dqn$ values estimate

2) Using vertices in $P$ to update node embeddings in such a way:

$$\mu_v^{(t+1)} \longleftarrow \text{relu}(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \mu_u^t + \theta_{sp\_adj} \sum_{u \in P} \mu_u^t + \theta_3 \sum_{u \in \mathcal{N}(v)} \text{relu}(\theta_4 w(v,u))) \tag{3}$$

3) Using vertices in $P$ to aggregate information about distances in such a way:

$$\mu_v^{(t+1)} \longleftarrow \text{relu}(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \mu_u^t + \theta_{sp\_adj} \sum_{u \in P} \mu_u^t + \theta_3 \sum_{u \in \mathcal{N}(v)} \text{relu}(\theta_4 w(v,u)) + \theta_{sp\_w} \sum_{u \in P} \text{relu}(\theta_4' w(v,u))) \tag{4}$$

4) Using vertices in $P$ to aggregate information about distances in such a way:

$$\mu_v^{(t+1)} \longleftarrow \text{relu}(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \mu_u^t + \theta_{sp\_adj} \sum_{u \in P} \mu_u^t + \theta_3 \sum_{u \in \mathcal{N}(v)} \text{relu}(\theta_4 w(v,u) \texttt{CONCAT}(\mu_v, \mu_u)) + \theta_{sp\_w} \sum_{u \in P} \text{relu}(\theta_4' w(v,u) \texttt{CONCAT}(\mu_v, \mu_u))) \tag{5}$$

The PRs of the above mentioned experiments are shown in the Table VIII

TABLE VIII: PRs for different node embedding models; $n = 6; m = 20$ full grid 1-Hanan neighbor

| | PR |
|---|---|
| Model 1 | 1.028 |
| Model 2 | 1.022 |
| Model 3 | 1.033 |
| Model 4 | 1.006 |

We see that computing messages using node embeddings as well as the distance between the two vertices is most beneficial.

### A. *Position aware Graph Neural Networks (PGNN)*

Implementing the model exactly in the same way as described in [16] makes the training unstable. The model never converges because of resampling of anchor sets at every forward pass. For same input pointset, the model produces different output at each forward pass which leads to unstable training for the RL algorithm.

### B. *CNN model*

The networks' inability to overfit on $m = 20$ pointsets points towards the fact that either there is not enough information for the model to learn over these examples or graph neural networks are not suited for this task, where information over long distances need to be shared so as to compute minimum path lengths. Instead of using a graph neural network, we implemented a CNN model to see if it can overfit on $m = 20$ pointsets. The input to the CNN model is a grid of size $L \times L$ wherein vertices in $P$ are assigned a value of one and all others are zero. The model produces output of size $L \times L$ wherein each value denotes the probability of choosing that vertex and adding in $S$. Since we know that all Steiner points lie on the Hanan grid formed by the points in $P$, we restrict the action space of the model by masking out the values of vertices that do not lie on the Hanan grid. There are total 25 convolution layers each with a kernel of size $5 \times 5$. The model architecture is inspired from [20].

We train this model using PPO algorithm for $n = 6; m = 20$ setting. The model achieves a PR of 1.03.

## VIII. Supervised training framework

Since we are only considering actions for 1 timestep, model can also be trained in supervised training framework instead of using RL algorithms. In supervised training, we feed the model with optimal action for each input pointset and the model is trained to learn (memorize rather) this optimal action. We perform this experiment using Model 4 (from VIII).

Surprisingly, a model trained this way can easily overfit on $m = 100$ pointsets with a PR of 1. This shows that even though we are providing enough information to the model, RL algorithm is not able to exactly overfit on these examples and our focus of work should be on algorithm side rather than on representation.

### A. *Complete 1-Steiner problem*

The model is trained for the complete 1-Steiner problem in supervised learning framework by running I1S algorithm over $m = 2000$ pointsets and storing the optimal actions for each instance. Dataset containing (pointset instance, optimal action) tuples are then used to train the model.

We use S2V [12] and S2V-Anchored (Model 4 in VIII) to train the model. The generalization PRs are shown in the table X

TABLE IX: Generalization tests for 1-Steiner in supervised framework; $m = 2000$

| Setting | $PR_{n=6}$ | $PR_{n=8}$ | $PR_{n=10}$ | $PR_{n=16}$ | $PR_{n=20}$ |
|---|---|---|---|---|---|
| S2V-Anchored $n = 6$ | 1.003 | 1.089 | 1.12 | - | - |
| S2V $n = 6$ | 1.004 | 1.016 | 1.027 | 1.084 | 1.103 |
| S2V $n = 8$ | 1.004 | 1.008 | 1.023 | 1.06 | 1.085 |
| S2V $n = 10$ | 1.006 | 1.006 | 1.015 | 1.03 | 1.046 |

TABLE X: Generalization tests for 1-Steiner in supervised framework; $m = 4000$

| Setting | $PR_{n=6}$ | $PR_{n=8}$ | $PR_{n=10}$ | $PR_{n=16}$ | $PR_{n=20}$ |
|---|---|---|---|---|---|
| S2V $n = 6$ | 1.0017 | 1.011 | 1.027 | 1.065 | - |
| S2V $n = 8$ | 1.004 | 1.007 | 1.016 | 1.05 | 1.066 |
| S2V $n = 10$ | 1.0036 | 1.007 | 1.011 | 1.043 | 1.059 |

TABLE XI: Comparison stats Supervised vs I1S

| Setting | RL $>$ I1S (%) | RL $=$ I1S (%) |
|---|---|---|
| S2V $n = 6$ | 4.8 | 76.8 |
| S2V $n = 8$ | 7.8 | 57.2 |
| S2V $n = 10$ | 8.6 | 38.8 |

TABLE XII: Comparison stats Supervised vs I1S

| Setting | Worst RL loss (%) | Best RL win (%) |
|---|---|---|
| S2V $n = 6$ | 13.01 | 7.14 |
| S2V $n = 8$ | 18.36 | 6.0 |
| S2V $n = 10$ | 18.5 | 5.07 |

## REFERENCES


[1] A. B. Kahng and G. Robins, "A New Class of Iterative Steiner Tree Heuristics with Good Performance", *IEEE Trans. on CAD* 11(7) (1992), pp. 893-902. https://vlsicad.ucsd.edu/Publications/Journals/j4.pdf

[2] A. B. Kahng and G. Robins, "A New Family of Steiner Tree Heuristics With Good Performance: The Iterated 1-Steiner Approach", *Proc. ICCAD*, 1990, pp. 428-431.

[3] A. B. Kahng and G. Robins, *On Optimal Interconnections*, Kluwer Academic Publishers, 1995.

[4] P. Berman, U. Foessmeier, M. Karpinski, M. Kaufmann and A. Z. Zelikovsky, "Approaching the 5/4-Approximation for Rectilinear Steiner Trees", *Proc. European Symp. on Algorithms*, 1994, pp. 60-71.

[5] P. Berman, U. Foessmeier, M. Karpinski, M. Kaufmann and A. Z. Zelikovsky, "Approaching the 5/4-Approximation for Rectilinear Steiner Trees", *Proc. European Symp. on Algorithms*, 1994, pp. 60-71.

[6] A. B. Kahng and G. Robins, "On Performance Bounds for a Class of Rectilinear Steiner Tree Heuristics in Arbitrary Dimension", *IEEE Trans. on CAD* 11 (1992), pp. 1462-1465.

[7] A. B. Kahng, J. Lienig, I. L. Markov and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*, Springer, 2011.

[8] G. Georgakopoulos and C. H. Papadimitriou, "The 1-Steiner Tree Problem", *J. Algorithms* 8 (1987), pp. 122-130.

[9] D. M. Warme, P. Winter and M. Zachariasen, "Exact Algorithms for Plane Steiner Tree Problems: A Computational Study", in D. Z. Du, J. M. Smith and J. H. Rubinstein (Eds.), *Advances in Steiner Trees*, Kluwer Academic Publishers, 2000, pp. 81-116.

[10] M. Garey and D. S. Johnson, "The Rectilinear Steiner Problem is NP-Complete", *SIAM J. Applied Math.* 32 (1977), pp. 826-834.

[11] F. K. Hwang, "On Steiner Minimal Trees with Rectilinear Distance", *SIAM J. Applied Math.* 30 (1976), pp. 104-114.

[12] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina and L. Song, "Learning Combinatorial Optimization Algorithms over Graphs", *arXiv preprint 1704.01665*, 2018. https://github.com/Hanjun-Dai/graph_comb_opt

[13] H. Dai, B. Dai and L. Song, "Discriminative Embeddings of Latent Variable Models for Structured Data" *arXiv preprint 1603.05629*, 2016. https://arxiv.org/abs/1603.05629

[14] H. Van Hasselt, A. Guez and D. Silver, "Deep Reinforcement Learning with Double Q-learning", *arXiv preprint 1509.06461*, 2015. https://arxiv.org/abs/1509.06461

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms", *arXiv preprint 1707.06347*, 2017. https://arxiv.org/abs/1707.06347

[16] Jiaxuan You, Rex Ying, Jure Leskovec, "Position-aware Graph Neural Networks", *arXiv preprint 1906.04817* , 2019. https://arxiv.org/abs/1906.04817

[17] https://github.com/mgwoo/tf-dqn

[18] N-Step Q Learning, https://nervanasystems.github.io/coach/components/agents/value_optimization/n_step.html

[19] M. Hanan, "On Steiner's Problem With Rectilinear Distance", *SIAM J. Applied Math.* 14 (1966), pp. 255-265.

[20] Sambhav R. Jain, Kye Okabe, "Training a Fully Convolutional Neural Network to Route Integrated Circuits", *arXiv preprint 1706.08948* , 2017. https://arxiv.org/abs/1706.08948