TreeNet: Deep Point Cloud Embedding for Routing Tree Construction

Wei Li
The Chinese University of Hong Kong

Yuxiao Qu
The Chinese University of Hong Kong

Gengjie Chen
Giga Design Automation

Yuzhe Ma
The Chinese University of Hong Kong

Bei Yu
The Chinese University of Hong Kong

Abstract

In the routing tree construction, both wirelength (WL) and pathlength (PL) are of importance. Among all methods, PD-II and SALT are the two most prominent ones. However, neither PD-II nor SALT always dominates the other one in terms of both WL and PL for all nets. In addition, estimating the best parameters for both algorithms is still an open problem. In this paper, we model the pins of a net as point cloud and formalize a set of special properties of such point cloud. Considering these properties, we propose a novel deep neural net architecture, TreeNet, to obtain the embedding of the point cloud. Based on the obtained cloud embedding, an adaptive workflow is designed for the routing tree construction. Experimental results show that the proposed TreeNet is superior to other mainstream models for the point cloud on classification tasks. Moreover, the proposed adaptive workflow for the routing tree construction outperforms SALT and PD-II in terms of both efficiency and effectiveness.

1 Introduction

In VLSI routing, wirelength (WL) and pathlength (PL) are the two fundamental metrics for the routing tree construction. Here, WL is directly related to power consumption, routing resource usage, cell delay and wire delay. Meanwhile, a long PL from the root (i.e., the source pin) implies high wire delay. However, optimizing either one of them does not necessarily benefit the other one.

The minimization of WL and PL has been investigated for a long history. Various approaches have been proposed to construct the routing tree by optimizing both WL and PL. These approaches can be roughly categorized into two types. The first type starts the construction from a tree that consists only of the source pin and iteratively adds the node into the tree with a newly-added edge. The two most influential and representative approaches of this type are the Prim-Dijkstra (PD) [1] construction and its improved version PD-II [2]. The second type [3–5] starts the construction from an initial topology with small WL such as FLUTE [6], and iteratively finds and reroutes the node whose PL is out of the bound. Among all the approaches above, PD-II [2] and SALT [5] are the two most prominent ones which demonstrate a superior trade-off between WL and PL compared with other state-of-the-art approaches. However, neither PD-II nor SALT always dominates the other one in terms of both WL and PL for all nets. Specifically, given a maximal WL constraint, although the PL of SALT is better than PD-II in most cases, there is still a considerable proportion on which PD-II is better, especially when the size of the net is large. The statistics shown in Table 2 demonstrate such a phenomenon, where PD-II outperforms SALT in 10.4% of huge nets when the WL degradation constraint is 0. Here the PL is measured by the normalized PL. Besides the uncertainty of deciding the best approach for any single case, there is another concern about the best parameter in PD-II and SALT. Both PD-II and SALT use a parameter to help decide whether one update should happen or not. For example, α in PD-II is included in the cost function to balance WL and PL. Correspondingly, ϵ in SALT is used to decide whether one node should be rerouted or not. Given any single case, i.e., a set of pins, the estimation of the best parameter is still non-trivial and also an open problem to achieve the best PL given one WL constraint.

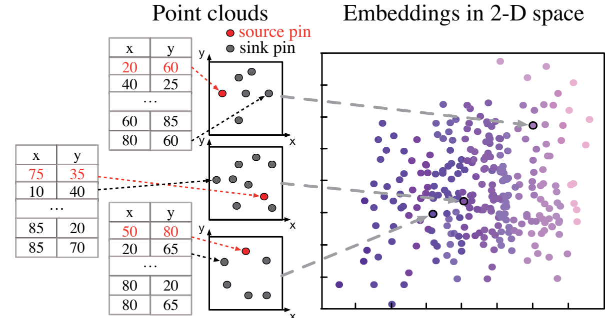


Figure 1: Cloud embeddings for tree construction, where point clouds are transformed into unified 2-D Euclidean space.

The recent overwhelming success of deep learning applications in various fields suggests that we can naturally cast the problems into the classification task (approach selection) and the regression task (parameter prediction). However, the set of 2-D points, which is the original input of the tree construction and also called the point cloud, usually varies in terms of the number of points, making it hard to be fed into a learning model. Therefore, we first need to transform a point cloud with unfixed size into a vector with a fixed size, where the vector is also called cloud embedding. The cloud embedding should be in a unified vector space with maximal representation capability such that the cloud embedding can help us to determine the best routing tree construction approach and predict the best parameter. One example of the point clouds for the routing tree construction and corresponding cloud embeddings are shown in Figure 1.

Although many works [7–10] adapt the powerful deep learning-based methods for cloud embedding, none of them handles the point

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASPDAC '21, January 18–21, 2021, Tokyo, Japan
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431566

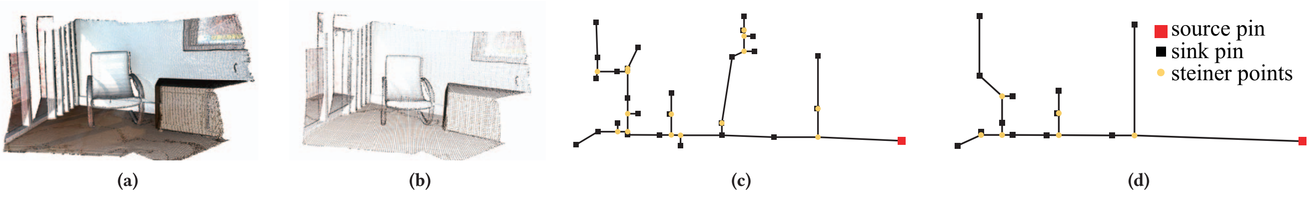


Figure 2: Examples of the down-sampling: (a) The general point cloud without the down-sampling; (b) The general point cloud with the down-sampling; (c) The constructed tree without the down-sampling; (d) The constructed tree with the down-sampling.

cloud specifically for the tree construction quite well due to some special properties in the tree construction. Through comprehensive analysis and consideration of these properties, we propose a deep learning-based model, TreeNet, to obtain the cloud embedding specifically for the tree construction. The obtained cloud embedding is then used as a representation to help select the best routing tree construction approach and predict the best parameter for the selected approach. Finally, we use the selected approach and corresponding parameter to construct the routing tree. The main contributions are summarized as follows: 1) We formalize special properties of the point cloud for the routing tree construction; 2) We design TreeNet, a novel deep net architecture to obtain the cloud embedding for the tree construction; 3) We propose an adaptive flow for the routing tree construction, which uses the cloud embedding obtained by TreeNet to select the best approach and predict the best parameter; 4) Experiments on widely used benchmarks and demonstrate the effectiveness of our embedding representation, compared with all other deep learning models; 5) Experimental results show that our methods outperform other state-of-the-art routing tree construction methods in terms of both quality and runtime.

2 Preliminaries

2.1 Routing Tree

The routing tree is constructed by a set of terminals. Assume a input net $\boldsymbol{V} = \{v_0, \boldsymbol{V}_s\}$, v_0 is the source and $\boldsymbol{V}_s$ is the set of sinks. Let $G = \{\boldsymbol{V}, \boldsymbol{E}\}$ be the connected weighted routing graph. The edge weight of G is the distance between vertices. A routing tree $T = \{\boldsymbol{V'}, \boldsymbol{E'}\}$ is a spanning/Steiner tree that is constructed from $\boldsymbol{V}$ with v_0 as the root. A Steiner tree inserts new points from $\boldsymbol{V}$, i.e., $\boldsymbol{V'} \supseteq \boldsymbol{V}$, where the newly inserted points are called steiner points. The objective of the routing tree is to minimize both WL and PL. The WL metric is called the lightness or normalized WL, which is computed by the WL ratio with that of minimum spanning tree (MST), i.e., $lightness = \frac{w(T)}{w(MST(G))}$, where $w(\cdot)$ is the total weight. The PL metric is controversial and there are two widely used metrics. The first one is called the shallowness [11], which is computed by the maximal PL ratio with the shortest-path tree (SPT) among all vertices, i.e., $shallowness = \max\{\frac{d_T(v_0, v)}{d_G(v_0, v)} | v \in \boldsymbol{V}_s\}$. The second one is called the normalized path length [2], which is computed by the total PL normalized by the total shortest-path distance, i.e., $normPL = \frac{\sum_{v \in \boldsymbol{V}} d_T(r, v)}{\sum_{v \in \boldsymbol{V}} d_G(r, v)}$. Note that $d(\cdot)$ mentioned above denotes as the Manhattan distance.

2.2 Point Cloud

Point Cloud is defined as a set of scattered points in a 2D plane or 3D space. Therefore, the input of the routing tree construction, i.e., a set of 2-D points, can be modeled as a point cloud. Typical deep learning-based methods obtain the embedding of a general point cloud by a similar philosophy of the convolution layer. The convolution-like operation is usually composed of three procedures: `Sampling`, `Grouping` and `Encoding`. `Sampling` selects a set of centroids from the original point cloud. `Grouping` selects a set of neighbors for each centroid, which is like the local region constrained by a convolution kernel in the original convolution. `Encoding` is to encode the new centroid feature using the original one and the local feature aggregated from the neighbors of the centroid.

2.3 Problem Formulation

Given a set of 2-D pins and two routing tree construction algorithms, SALT [11] and PD-II [2], our objective is to obtain the embedding of the given point cloud by TreeNet such that 1) the embedding can be used to select the best algorithms for the given point cloud; 2) the embedding can be used to estimate the best parameter ϵ of SALT for the given point cloud; 3) the embedding can be used to estimate the best parameter α of PD-II for the given point cloud.

3 The Proposed Approaches

In this section, we first formalize a set of special properties of the point cloud specifically for the routing tree construction. Then we propose TreeNet for cloud embedding by considering these properties. Finally, an adaptive workflow for the routing tree construction based on the cloud embedding is introduced.

3.1 Property Analysis

Given the input net $\boldsymbol{V} = \{v_0, \boldsymbol{V}_s\}$. Let $f : \boldsymbol{V} \rightarrow T$ be the function for routing tree construction and T is the target routing tree. Here, we say $T = T'$ if and only if T and T' have the same node coordinates and the same topology. Ideally, a powerful neural network maps nets with different (same) routing trees to embeddings which are as different (similar) as possible. Therefore, we may design the cloud embedding method by learning the behavior of f. [1]

Property 1. *Let $d : \boldsymbol{V} \rightarrow \boldsymbol{V'}$ be a function for down-sampling, where $\boldsymbol{V'}$ is a proper subset of $\boldsymbol{V}$. $f(\boldsymbol{V}) \neq f(d(\boldsymbol{V}))$ holds if there exists $v \in \boldsymbol{V} - d(\boldsymbol{V})$ so that v is not the steiner point in $f(d(\boldsymbol{V}))$.*

Property 1 points the deficiency of down-sampling. Actually, the inequality holds for most cases even without the condition. One down-sampling example is shown in Figure 2. With down-sampling, the skeleton of the general point cloud is still easy to classify as shown in Figure 2(b). However, given the point cloud for the routing

[1] Proofs of all properties are not shown here due to the page limit.

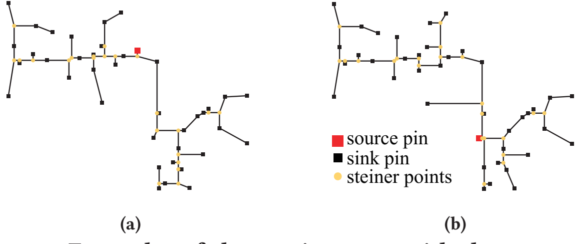


Figure 3: Examples of the routing trees with the same node distribution but different root (highlighted by red).

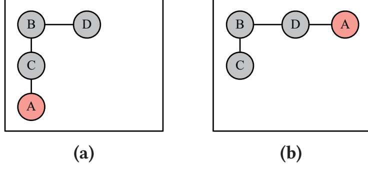


Figure 4: Examples of the node with the same coordinates and local neighbors but different parent-child relationships. Here root is highlighted in red.

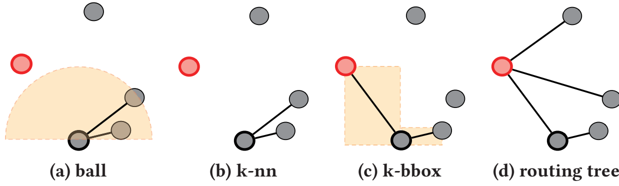


Figure 5: Comparison among ball query (a) k-nn (b) and k-bbox (c) grouping methods ($k = 2$ in this example). The orange regions represent the query ball in (a) and bounding boxes in (c). The centroid is highlighted by black and the root is by red.

tree construction, the routing tree with down-sampling (Figure 2(d)) is totally different from the one without down-sampling (Figure 2(c)). Here,

Property 2. *Let V_s^p be the permutation of the sink set V_s. $f(\{v_0, V_s^p\}) = f(\{v_0, V_s\})$ holds for any $V = \{v_0, V_s\}$.*

Property 3. *Let V^p be the permutation of the input net V. $f(V^p) \neq f(V)$ holds if the source in V^p is different from the source in V.*

Property 2 shows the permutation invariance of the point cloud and Property 3 states the sensitivity of the root. One example is shown in Figure 3, where the space distributions of point locations are totally the same while only the root is assigned differently. A typical method may regard the two point clouds as a similar pair while they actually represent completely different trees.

Property 4. *For any sink set V_s with $|V_s| > 1$, there exists two different pins, v_0 and v_0' in the 2-D plane so that $f(\{v_0, V_s\}) \neq f(\{v_0', V_s\})$. Moreover, the inequality holds when we only consider the topology.*

As an extension of Property 3, Property 4 states the possible inequality even when the sink set V_s is not changed. It not only demonstrates the sensitivity of the root, but also shows the deficiency of only considering local information, i.e, information stored in V_s. One example is shown in Figure 4, where the node B in both Figure 4(a) and Figure 4(b) have the same coordinates and local neighbors (C, D) but the parent-child relationships $B-C$ and $B-D$ are clearly different.

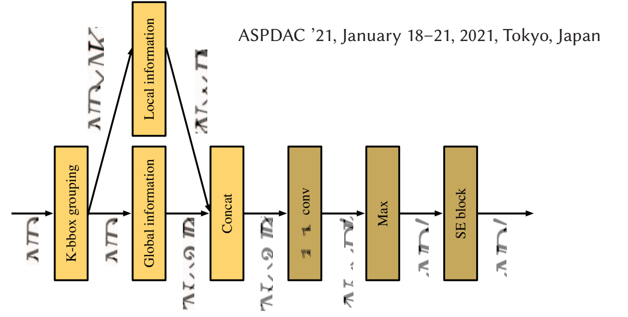


Figure 6: Illustration of TreeConv. Brighter blocks indicate `Grouping` and darker blocks indicate `Encoding`.

Property 5. *Let G_{ball}, G_{knn} and G_{bbox} be the graph constructed from V by ball query, k nearest neighbor and bounding box respectively. The minimum spanning tree, T may not be the subgraph of G_{ball} or G_{nn}, but always the subgraph of G_{bbox}.*

Property 5 states that G_{bbox} [2] is more likely to capture the structure of the routing tree, compared to G_{ball} [8] and G_{knn} [9, 10]. G_{bbox} is the graph whose nodes are connected with their *bbox-neighbors*. We call the node u_j as the *bbox-neighbor* of u_i if there is no other node in the smallest bounding box containing u_i and u_j. One example of the comparison is shown in Figure 5, G_{ball} and G_{knn} fail to find the correct neighbors of the selected centroid.

3.2 Cloud embedding by TreeNet

Considering all these properties discussed above, we propose a specialized model, TreeNet, to obtain the embedding of the point cloud for the routing tree construction. Basically, TreeNet is a hierarchical model and composed of a number of convolution-like operations. We refer to this operation as TreeConv. The comparison between TreeConv and other methods [7–10] are summarized in Table 1.

Our TreeConv (see Figure 6) leverages the local correlation information and the root information with two key procedures: `Grouping` and `Encoding`. Different from some typical works, TreeConv omits the `Sampling` phase considering Property 1. Therefore, each node is selected as the centroid. Given a point cloud $\boldsymbol{H} \in \mathbb{R}^{N \times D}$, where N is the number of points and D is the dimension of each point, our `Grouping` selects k neighbors for each centroid u_i to represent the local point cloud structure based on G_{bbox} [2]. One example is shown in Figure 5(c). We first select k nearest *bbox-neighbors* of u_i as the neighbors. If the number of *bbox-neighbors* for u_i is less than k, we then select other nearest nodes to fill up k neighbors. Therefore, `Grouping` returns a list of neighbors $\boldsymbol{E}_i \in \mathbb{R}^k$ for each centroid u_i. After `Grouping`, our `Encoding` outputs a new feature $\boldsymbol{v}_i' \in \mathbb{R}^{D'}$ for each node u_i such that the new point cloud $H' = \{\boldsymbol{v}_0', ..., \boldsymbol{v}_{N-1}'\} \in \mathbb{R}^{N \times D'}$. For each element v_{ic}' in $\boldsymbol{v}_i'$, our `Encoding` leverages the global position information [7], the "local" neighborhood information [9, 10], and the root information considering Property 4. The computation can be formulated as:

$$v_{ic}' = \max_{j \in E_i} \sigma(\boldsymbol{\theta}_c \cdot \texttt{CONCAT}(\boldsymbol{v}_i - \boldsymbol{v}_j, \boldsymbol{v}_i - \boldsymbol{v}_r, \boldsymbol{v}_i)), \tag{1}$$

where $\boldsymbol{v}_r$ is the input feature of the root, σ is the `LeakyReLU` activation function and $\boldsymbol{\theta}_c \in \mathbb{R}^{3D}$ is the trainable weight of c_{th} filter. Finally, the new feature $\boldsymbol{v}_i'$ is processed by a Squeeze-and-Excitation (SE) block [12] for exploiting channel dependencies.

The network architecture used for the cloud embedding is shown in Figure 7. The input of the first layer is the point cloud $\boldsymbol{H}_0 \in \mathbb{R}^{N \times 2}$, in which each node has a 2-D normalized coordinate feature. The normalization of each node u_i is based on the root u_r (Property 3,

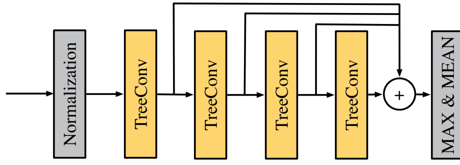


Figure 7: Illustration of TreeNet Architecture for the cloud embedding.

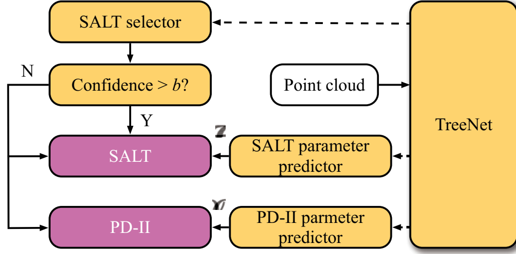


Figure 8: The workflow of our framework. Dotted arrows represent that TreeNet generates cloud embeddings and use them to select the algorithm or to predict parameters. The yellow blocks are executed in our framework while the purple blocks are executed by the selected algorithms.

Property 4) and can be formulated as:

$$\tilde{\boldsymbol{v}}_i = \frac{\boldsymbol{v}_i - \boldsymbol{v}_r}{d_{max}}, \tag{2}$$

where $\boldsymbol{v}_i$ is the original 2-D coordinate feature of node u_i and $\boldsymbol{v}_r$ is the feature of the root. d_{max} is the maximal distance between the root and any other nodes.

We stack four TreeConvs and include shortcut connections from each layer to the output to extract multi-scale features which are finally concatenated. Then, two permutation-invariant operations (Property 2), max pooling and average pooling, are used to get the cloud embedding, which can be formulated as

$$\begin{aligned} \boldsymbol{H}_c = \texttt{CONCAT}(&\max(\texttt{CONCAT}(\tilde{\boldsymbol{H}}_1, \tilde{\boldsymbol{H}}_2, \tilde{\boldsymbol{H}}_3, \tilde{\boldsymbol{H}}_4)), \\ &\text{mean}(\texttt{CONCAT}(\tilde{\boldsymbol{H}}_1, \tilde{\boldsymbol{H}}_2, \tilde{\boldsymbol{H}}_3, \tilde{\boldsymbol{H}}_4))), \end{aligned} \tag{3}$$

where $\boldsymbol{H}_c \in \mathbb{R}^{2\times(D_1+D_2+D_3+D_4)}$ is the final cloud embedding and $\tilde{\boldsymbol{H}}_i \in \mathbb{R}^{N\times D_i}$ is the scaled output of i_{th} TreeConv.

3.3 Routing Tree Construction based on Point Cloud Embedding

Given the cloud embedding $\boldsymbol{H}_c \in \mathbb{R}^D$ obtained by TreeNet, we can cast the algorithm selection and the parameter prediction problem into classification and regression problem, respectively. The workflow of our framework is shown in Figure 8.

Firstly, we use the obtained cloud embedding to determine whether the SALT algorithm is at least as good as the PD-II algorithm, where "good" means that the best PL of SALT is at least the same as that of PD-II under the given WL constraint and the PL metric. Therefore, the problem can be regarded as a 2-class classification problem, where one class indicates that the result of SALT is at least as good as PD-II while another one indicates its opposite. Given the cloud embedding $\boldsymbol{H}_c \in \mathbb{R}^D$, the 2-class classifier is implemented by three fully connected layers followed by a softmax layer and can be formulated as:

$$\boldsymbol{y} = \texttt{softmax}(\boldsymbol{W}_3\sigma(\boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{H}_c + \boldsymbol{b}_1) + \boldsymbol{b}_2)), \tag{4}$$

Algorithm 1 `ParameterGuidanceConstruction`

Input: $\epsilon \rightarrow$ Predicted parameter;
Input: $f \rightarrow$ Routing tree construction algorithm;
1: $results \leftarrow$ Run f using the parameters in $(\epsilon - \sigma, \epsilon + \sigma) \cup S$;
2: **if** no result in $results$ satisfies the WL constraint **then**
3: **return** `ParameterGuidanceConstruction`$(\epsilon + 2\sigma, f)$;
4: **else**
5: **return** the result with the best PL in $results$ satisfying the WL constraint under the PL metric;
6: **end if**

where σ is the `LeakyReLU` activation function, $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ are the weight matrix and bias vector, respectively. $\boldsymbol{y} \in \mathbb{R}^2$ is the final classification confidence. Since each wrong selection directly harms the quality of the result, we raise the bar to select SALT algorithm: We directly use SALT algorithm to construct the routing tree only when the confidence of "SALT is at least as good as PD-II" is larger than a specified confidence bar b. Otherwise, we will use both SALT and PD-II to construct the routing tree and use the better result.

After algorithm selection, corresponding parameter is predicted by the obtained cloud embedding and guides the parameter selection. The regression target of the parameter prediction is the "best" parameter, where "best" means the result using such parameter achieves the best PL under the WL degradation constraint. We follow a similar approach used in the age prediction [13]. Take the parameter prediction of SALT for example. We set 20 valid parameter $\epsilon_i, i \in \{1, \ldots, 20\}$ candidates for SALT and each valid parameter is treated as a separate class. Therefore, the structure for the parameter prediction is similar with the one for the algorithm selection formulated in Equation (4) with $\boldsymbol{y} \in \mathbb{R}^{20}$. Given the output $\boldsymbol{y}$, the predictied parameter ϵ is calculated by an element-wise summation and can be formulated as:

$$\epsilon = \sum_{i=1}^{20} \epsilon_i \cdot y_i. \tag{5}$$

Given the predicted parameter, the guidance follows a simple but effective heuristic rule specified in Algorithm 1. Generally speaking, the selected approach constructs the routing tree using the predicted parameter and other parameters in a set (line 1), where σ and S are hyper-parameters. S defines an initial set of parameters that achieve almost the best PL while also almost the worst WL. If the results of all tested parameters fail to satisfy the WL constraint, the range is widened along the direction which decreases the WL (line 2-3); Otherwise, we directly use the best parameter among these tested candidates (line 4-5).

4 Experimental Results

We implement TreeNet in Python with PyTorch. Other models (PointNet [7], PointNet++ [8], PointCNN [10], DGCNN [9]) are also implemented. The results of SALT [5] are generated by the source code and the results of PD-II [2] are provided by the authors. The experiments are conducted on the widely used benchmarks of ICCAD 2015 Contest [14]. All the two-pin and three-pin nets are removed. All the experiments are conducted on an Intel Core 2.9 GHz Linux machine with one NVIDIA TITAN Xp GPUs.

Data Preparation. We first assign labels to each net in the benchmarks according to the routing tree results of SALT and PD-II. Specifically, given the constraint of WL degradation percentage with respect to MST WL and the PL metric type (the shallowness metric and the

Table 1: Comparison to existing methods.

	`Sampling`	`Grouping`	`Encoding`
PointNet [7]	-	-	$v'_{ic} = \sigma(\boldsymbol{\theta}_c \boldsymbol{v}_i)$
PointNet++ [8]	Fathest Point Sampling (FPS)	ball query's local neighborhood	$v'_{ic} = \max_{j \in E_i} \sigma(\boldsymbol{\theta}_c \boldsymbol{v}_j)$
PointCNN [10]	Random/FPS	k nearest neighbor	$\boldsymbol{v}'_i = \mathtt{Conv}(X \times \boldsymbol{\theta}(\boldsymbol{v}_i - \boldsymbol{v}_j))$
DGCNN [9]	-	k nearest neighbor	$v'_{ic} = \max_{j \in E_i} \sigma(\boldsymbol{\theta}_c \cdot \mathtt{CONCAT}(\boldsymbol{v}_i - \boldsymbol{v}_j, \boldsymbol{v}_i))$,
Our work	-	k bounding box neighbor	$v'_{ic} = \max_{j \in E_i} \sigma(\boldsymbol{\theta}_c \cdot \mathtt{CONCAT}(\boldsymbol{v}_i - \boldsymbol{v}_j, \boldsymbol{v}_i - \boldsymbol{v}_r, \boldsymbol{v}_i))$,

Table 2: ICCAD 2015 Benchmark Label Statistics (part)

WL deg.	PL metric	Label	Small	Med.	Large	Huge	Total
0	Nor. PL	SALT	99.9%	98.8%	93.5%	89.6%	1273012 (98.3%)
		PD-II	0.1%	1.2%	6.5%	10.4%	21529 (1.7%)
0	Shallow.	SALT	99.9%	99.1%	95.6%	93.1%	1279428 (98.8%)
		PD-II	0.1%	0.9%	4.4%	6.9%	15113 (1.2%)
10%	Shallow.	SALT	99.8%	97.0%	93.6%	91.4%	1269095 (98.0%)
		PD-II	0.2%	3.0%	6.4%	8.6%	25446 (2.0%)

normalized PL metric), the net is assigned three labels: 1) *best algorithm*; 2) *best* ϵ for SALT and 3) *best* α for PD-II. The *best algorithm* is labeled as one of $\{SALT, PDII\}$. Here, the net is labeled as *SALT* (*PDII*) when the routing tree by SALT (PD-II) achieves the best PL under the chosen PL metric and the WL degradation constraint. Especially, if both SALT and PD-II construct the same routing tree, the *best algorithm* is labeled as *SALT*. The *best* ϵ for SALT is defined by:

$$\bar{y_i} = \begin{cases} \frac{1}{k}, & \text{if } \epsilon_i \text{ is one of the “best” candidates;} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

where k is the number of those "best" candidates, $i \in \{1, ..., 20\}$ and $\epsilon_i = 0.05 \times 1.5^i$ as defined in [5]. Similarly, the best α for PD-II also follows Equation (6) with $i \in \{1, ..., 19\}$ and $\alpha_i = 0.05 \cdot i$ as defined in [2]. Note that, increasing the number of data points also improves the quality of our model since the noise label is reduced. Given such label rules, the label distribution of nets in the benchmarks are clearly different based on the WL degradation constraint and PL metric type. A part of the *best algorithm* label statistics for the benchmarks is shown in Table 2, where "WL deg." denotes the percentages of permissible WL degradation with respect to MST WL.

Architecture. The output point dimension of four TreeConvs is (32, 32, 64, 128) and such the final embedding dimension is $(32 + 32 + 64 + 128) \times 2 = 512$. Then, three fully connected layers $(128, 64, c)$ are used, where c is the number of classes in the label. Dropout is applied and the keep probability is set to 0.5. The number of selected neighbors k is set to 3, which is the maximal neighbor number for a 4-point net. The confidence bar b is set as 0.99. The range σ in Algorithm 1 is set to 1 for SALT and -1 for PD-II. The set S is set to {1,2} in SALT and {18,19} in PD-II.

Training & Testing. During training, we minimize the binary cross-entropy loss for the algorithm selection and the soft cross-entropy loss for the parameter prediction. We use SGD with an initial learning rate 0.001 and momentum 0.9, and the learning rate is reduced by 30% every 20 epochs. We follow the idea of K-fold cross-validation to set up the test. Specifically, each design in the benchmark is tested using the model trained by other designs following the same configurations.

4.1 Comparison with other DL models

In this section, we compare TreeNet with other baseline models for the algorithm selection task. Due to the page limit, the result of the parameter prediction task is not shown since it is also formulated as a classification task. The WL degradation is set as 5% and the PL metric is set as the normalized PL. Formally, we mark the *SALT* label as positive and the *PDII* as negative.

The result for the algorithm selection is shown in Table 3, where "Accuracy" and "Precision" are defined as usual. "Recall*" is slightly different and defined as the fraction of the total amount of positive instances that were also predicted as positive with the confidence larger than the bar b. Therefore, the updated recall is directly related to the runtime performance. We use SALT to construct the routing tree for the predicted positive instances, and use both SALT and PD-II for the predicted negative instances. We compare four state-of-the-art models with our TreeNet and three variations: 1) Remove the root-sensitive normalization and use the original normalization (property 1); 2) Remove the root-related global information in `Encoding` phase (property 3, 4); 3) Use k-nn grouping method instead of k-bbox (property 5). According to Table 3, we can see that our TreeNet outperforms other state-of-the-art models on all three metrics. Besides, the comparison with other variations demonstrates the effectiveness of our considerations on the properties of the point cloud for the tree construction.

4.2 Comparison with routing tree constructors

In this section, we compare our adaptive workflow with SALT and PD-II in terms of both effectiveness and efficiency.

Effectiveness: We follow the same result comparison way in [2]: We first select different WL degradation constraints (0%, 5%, 10%, 15%, 20%) and then find the best shallowness and the normalized PL. Each entry in the table is the averaged shallowness (see Table 4) and the averaged normalized PL (see Table 5) across all test nets. Especially, SALT* executes SALT in a binary search manner, which results in better efficiency but may harm the quality. We also measure the improvement compared with SALT (**Imp. (%)**) and SALT* (**Imp.* (%)**). The improvement is calculated by the percentage improvement after subtracting the lower bound 1. For example, a reduction from 1.10 to 1.09 results in an improvement of 10%, i.e., $(1 - (1.09 - 1.0)/(1.10 - 1.0)) \cdot 100\%$. As Table 4 and Table 5 show, our adaptive workflow outperforms SALT and PD-II for all classes of nets under all WL constraints and PL metrics. In general, the overall improvement over SALT ranges from 1.97% to 12.16%, depending on the selected WL constraint and PL metric. The improvement over SALT* is more significant, ranging from 2.89% to 19.11%.

Efficiency: Since the source code of PD-II is not provided, we only compare the runtime performance of our adaptive workflow with SALT [5]. The runtime of our framework is composed of three parts: 1) The inference time of TreeNet; 2) The execution time of SALT on the input net; 3) The execution time of PD-II on the input net when the algorithm selector does not select SALT as the only tree constructor. Especially, the execution time of PD-II is estimated form the runtime analysis in [2], where PD-II costs 361s and SALT costs 2762s. Therefore, we estimate the runtime of PD-II by SALT with a

Table 3: Algorithm selection results

Method	Accuracy	Precision	Recall*
PointNet [7]	54.13	53.95	1.91
PointNet++ [8]	81.31	82.50	2.65
PointCNN [10]	62.18	64.24	1.16
DGCNN [9]	92.24	94.62	11.84
TreeNet w.o. Nor	87.22	88.62	15.69
TreeNet w.o. global	92.40	94.63	25.53
TreeNet w. knn	92.58	94.79	26.76
TreeNet	**94.09**	**95.38**	**50.74**

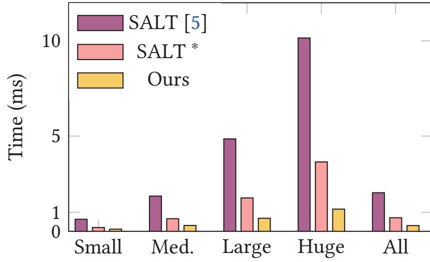
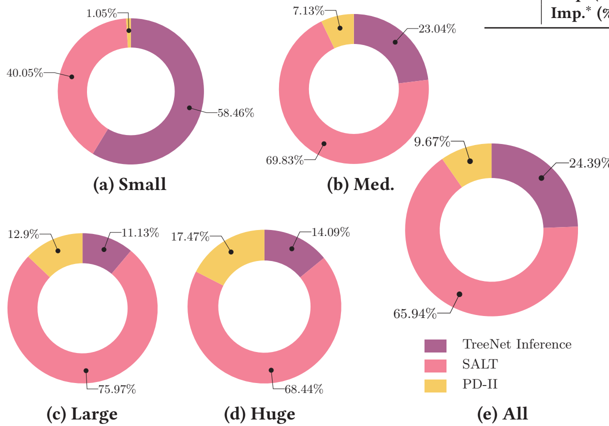


Figure 9: Runtime comparison with SALT and SALT*.



(a) Small (b) Med. (c) Large (d) Huge (e) All

Figure 10: Runtime breakdown of our framework.

Table 4: On shallowness

|V|	Method	WL deg. 0%	5%	10%	15%	20%
Small	PD-II	1.0606	1.0369	1.0240	1.0161	1.0114
	SALT	1.0462	1.0216	1.0078	1.0022	1.0006
	SALT*	1.0462	1.0216	1.0079	1.0023	1.0006
	Ours	1.0461	1.0210	1.0074	1.0021	1.0005
	Imp. (%)	**0.28**	**2.62**	**4.40**	**5.42**	**8.25**
	Imp.* (%)	**0.32**	**3.04**	**5.14**	**6.75**	**9.94**
Med.	PD-II	1.3849	1.2518	1.1688	1.1176	1.0851
	SALT	1.3456	1.1775	1.0838	1.0391	1.0181
	SALT*	1.3463	1.1815	1.0868	1.0410	1.0192
	Ours	1.3435	1.1689	1.0790	1.0370	1.0172
	Imp. (%)	**0.62**	**4.85**	**5.72**	**5.57**	**5.41**
	Imp.* (%)	**0.80**	**6.95**	**8.98**	**9.92**	**10.41**
Large	PD-II	1.9093	1.5584	1.3595	1.2473	1.1805
	SALT	1.7976	1.3549	1.1568	1.0727	1.0358
	SALT*	1.8083	1.3689	1.1648	1.0771	1.0382
	Ours	1.7755	1.3339	1.1481	1.0690	1.0341
	Imp. (%)	**2.77**	**5.91**	**5.53**	**5.11**	**4.78**
	Imp.* (%)	**4.06**	**9.50**	**10.12**	**10.52**	**10.77**
Huge	PD-II	2.1660	1.7169	1.4771	1.3438	1.2603
	SALT	2.0111	1.4398	1.2083	1.0987	1.0466
	SALT*	2.0291	1.4567	1.2183	1.1039	1.0489
	Ours	1.9793	1.4152	1.1975	1.0941	1.0444
	Imp. (%)	**3.15**	**5.61**	**5.17**	**4.69**	**4.64**
	Imp.* (%)	**4.85**	**9.09**	**9.50**	**9.47**	**9.20**
All	PD-II	1.2921	1.1822	1.1193	1.0827	1.0604
	SALT	1.2531	1.1175	1.0524	1.0236	1.0110
	SALT*	1.2555	1.1210	1.0546	1.0248	1.0117
	Ours	1.2481	1.1114	1.0495	1.0223	1.0104
	Imp. (%)	**1.97**	**5.18**	**5.43**	**5.21**	**5.08**
	Imp.* (%)	**2.89**	**7.98**	**9.23**	**9.95**	**10.38**

Table 5: On normalized PL

|V|	Method	WL deg. 0%	5%	10%	15%	20%
Small	PD-II	1.0156	1.0099	1.0065	1.0044	1.0031
	SALT	1.0113	1.0055	1.0020	1.0006	1.0002
	SALT*	1.0113	1.0055	1.0020	1.0006	1.0002
	Ours	1.0112	1.0053	1.0019	1.0005	1.0001
	Imp. (%)	**0.25**	**2.86**	**4.88**	**6.57**	**10.55**
	Imp.* (%)	**0.29**	**3.38**	**5.83**	**8.29**	**12.75**
Med.	PD-II	1.0897	1.0579	1.0373	1.0248	1.0170
	SALT	1.0778	1.0428	1.0204	1.0096	1.0044
	SALT*	1.0780	1.0440	1.0214	1.0102	1.0048
	Ours	1.0773	1.0396	1.0185	1.0086	1.0040
	Imp. (%)	**0.63**	**7.35**	**9.45**	**10.01**	**10.00**
	Imp.* (%)	**0.82**	**9.90**	**13.70**	**15.74**	**16.65**
Large	PD-II	1.1968	1.1146	1.0671	1.0413	1.0267
	SALT	1.1665	1.0815	1.0365	1.0172	1.0086
	SALT*	1.1690	1.0854	1.0390	1.0187	1.0095
	Ours	1.1616	1.0726	1.0318	1.0150	1.0076
	Imp. (%)	**2.95**	**10.92**	**12.81**	**12.91**	**12.49**
	Imp.* (%)	**4.35**	**15.02**	**18.29**	**19.70**	**20.35**
Huge	PD-II	1.2472	1.1415	1.0830	1.0513	1.0328
	SALT	1.2120	1.1054	1.0489	1.0224	1.0105
	SALT*	1.2160	1.1106	1.0522	1.0242	1.0112
	Ours	1.2045	1.0917	1.0413	1.0190	1.0088
	Imp. (%)	**3.54**	**13.03**	**15.54**	**15.54**	**16.25**
	Imp.* (%)	**5.31**	**17.12**	**20.97**	**21.52**	**21.87**
All	PD-II	1.0658	1.0398	1.0244	1.0157	1.0105
	SALT	1.0550	1.0278	1.0125	1.0056	1.0026
	SALT*	1.0555	1.0289	1.0132	1.0061	1.0029
	Ours	1.0538	1.0253	1.0111	1.0050	1.0023
	Imp. (%)	**2.05**	**9.17**	**11.35**	**11.94**	**12.16**
	Imp.* (%)	**3.01**	**12.43**	**16.04**	**17.98**	**19.11**

runtime ratio $361/2762 = 0.1307$. Figure 9 shows the average runtime comparison with SALT and SALT*, where adaptive workflow is more efficient than both of them on any size scale. We further profile the runtime of the framework, as shown in Figure 10. The inference time of TreeNet only occupies 24.39% of the total runtime.

5 Conclusion

In this work, we design a novel deep net architecture, TreeNet, to consider special properties of the point cloud for the tree construction. TreeNet is used to obtain the cloud embedding. Then, we propose an adaptive workflow to construct the routing tree based on the cloud embedding obtained by TreeNet. The results show that our TreeNet is far stronger than other deep learning-based models and the proposed framework achieves superior trade-off between WL and PL with less runtime, compared with the state-of-the-art routing tree construction methods.

Acknowledgment

This work is partially supported by The Research Grants Council of Hong Kong SAR (No. CUHK14209420).

References

[1] C. J. Alpert, T. Hu, J. Huang, A. B. Kahng, and D. Karger, "Prim-Dijkstra tradeoffs for improved performance-driven routing tree design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 7, pp. 890–896, 1995.

[2] C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu, and S. Venkatesh, "Prim-Dijkstra revisited: Achieving superior timing-driven routing trees," in *ACM International Symposium on Physical Design*, 2018, pp. 10–17.

[3] M. Elkin and S. Solomon, "Steiner shallow-light trees are exponentially lighter than spanning ones," in *IEEE Symposium on Foundations of Computer Science*, 2011, pp. 373–382.

[4] R. Scheifele, "Steiner trees with bounded RC-delay," *Algorithmica*, vol. 78, no. 1, pp. 86–109, 2017.

[5] G. Chen and E. F. Young, "Salt: provably good routing topology by a novel steiner shallow-light tree algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[6] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2008.

[7] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[8] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in neural information processing systems*, 2017, pp. 5099–5108.

[9] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5, pp. 1–12, 2019.

[10] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," in *Advances in neural information processing systems*, 2018, pp. 820–830.

[11] G. Chen, P. Tu, and E. F. Young, "SALT: provably good routing topology by a novel Steiner shallow-light tree algorithm," in *IEEE/ACM International Conference on Computer-Aided Design*, 2017, pp. 569–576.

[12] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[13] J. M. Schwarz, D. N. Cooper, M. Schuelke, and D. Seelow, "Mutationtaster2: mutation prediction for the deep-sequencing age," *Nature methods*, vol. 11, no. 4, pp. 361–362, 2014.

[14] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD Contest in incremental timing-driven placement and benchmark suite," in *IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 921–926.