Learning Point Clouds in EDA

Wei Li, Guojin Chen, Haoyu Yang, Ran Chen, Bei Yu

The Chinese University of Hong Kong
More Considerations

- Existing attempts still rely on regular format of data, like images;
- Netlists and layouts are naturally represented as graphs;
- Few DL solutions for graph-based problems in EDA.
An example of graph embeddings of layout graphs, where the graphs are transformed into vector space.
Irregular data representation in EDA: Point Cloud

An example of point-cloud embeddings of a placement.
## Graph vs. Point Cloud

<table>
<thead>
<tr>
<th>Graph</th>
<th>Point Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>A set of vertices and edges;</td>
<td>A set of data points in space;</td>
</tr>
<tr>
<td>Strictly constrains inter-connected relationships: requires the definition of connections (edges) among objects (nodes);</td>
<td>Directly preserves the original geometric information without any discretization or misinterpretation;</td>
</tr>
</tbody>
</table>
Previous works: Deep learning in EDA

By topics

- Routability estimation;
- Clock-tree synthesis;
- Placement & floorplanning;
- Lithography hotspot detection and mask optimization;

Graph Neural Networks

- Message-passing scheme;
- Netlist;
- Layout;
Multi-view-based methods:

- Transform a 3D point cloud into multiple views through projection;
- Extracted view-based features are fused together to generate a cloud embedding;
Previous works: Point Cloud Learning with Neural Networks

**Volumetric-based Methods:**

- **Voxelize** a point cloud into regular grids;
- A 3D Convolutional Neural Network is used for the embedding extraction;
Previous works: Point Cloud Learning with Neural Networks

**Point-based Methods:**

- Directly handle with raw points to avoid information loss.
- Include three procedures to obtain the embedding: *Sampling*, *Grouping* and *Encoding*.
  - *Sampling*: select centroids from the original point;
  - *Grouping*: select neighbors (also called agglomerates) for each centroid;
  - *Encoding*: encode the new centroid feature using the features from the neighbors and itself;
Challenges in EDA applications

► Order invariance;
  - Both multi-view based methods and volumetric-based methods: transformation
  - point-based methods: some symmetric functions like max-pooling or summation
    or special trainable network

► Irregularity:
  - Both multi-view based methods and volumetric-based methods transform the
    irregular point cloud into regular grid-like data such as image or voxel.
  - point-based methods directly work on points and propose networks specifically for
    irregular data like GNNs.

► Sparsity;

► Dimension: 3D vs. 2D
Outline

Case Study 1: Routing Tree Construction

Case Study 2: Hotspot Detection

Conclusion
Routing Tree Construction: Given an input net $V = \{v_0, V_s\}$, where $v_0$ is the source (red node) and $V_s$ is the set of sinks (black node), construct a tree optimizing both wire length and path length.

Examples of routing tree construction. Left: spanning tree; right: Steiner tree.
Wire length (WL) and path length (PL)

Wire length (WL) metric: lightness

- WL ratio with that of minimum spanning tree (MST).
- \(\text{lightness} = \frac{w(T)}{w(MST(G))}\), \(w(\cdot)\) is the total weight.

Path length (PL) metric: shallowness or normalized path length

- \(\text{Shallowness} = \max\{\frac{d_T(v_0, v)}{d_G(v_0, v)}|v \in V_s\}\), \(G\) is the connected weighted routing graph.
- Normalized path length = \(\frac{\sum_{v \in V} d_T(v_0, v)}{\sum_{v \in V} d_G(v_0, v)}\).
Non-trivial questions in the routing tree construction

### Best algorithm?

- Neither PD-II nor SALT, two most prominent ones, always dominates the other one in terms of both WL and PL for all nets.

### Best parameter?

- Both PD-II and SALT use a parameter to help balance WL and PL.
- Given one WL constraint, what is the best parameter to obtain the best PL?
Cloud embeddings for tree construction, where point clouds are transformed into unified 2-D Euclidean space.
Problem formulation

Given a set of 2-D pins and two routing tree construction algorithms, SALT\textsuperscript{1} and PD-II\textsuperscript{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 algorithm for the given point cloud;
2. the embedding can be used to estimate the best parameter $\epsilon$ of SALT for the given point cloud;
3. the embedding can be used to estimate the best parameter $\alpha$ of PD-II for the given point cloud.

\textsuperscript{1}Gengjie Chen and Evangeline FY Young (2019). “SALT: provably good routing topology by a novel steiner shallow-light tree algorithm”. In: IEEE TCAD.

Property 1: Down-sampling

Property

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

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.
Property 2 & 3: Permutation

**Property**

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

**Property**

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 $V$.

Examples of the routing trees with the same node coordinates but different source (highlighted by red).
Property 4: Inequality of the same $V_s$

**Property**

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.

Examples of the node with the same coordinates and local neighbors but different parent-child relationships. Here root is highlighted in red.
Property 5: Graph construction methods

Let $G_{\text{ball}}$, $G_{\text{knn}}$ and $G_{\text{bbox}}$ 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 $G_{\text{ball}}$ or $G_{\text{nn}}$, but always the subgraph of $G_{\text{bbox}}$.

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.
TreeConv

- **Sampling** selects a set of centroids from the original point cloud.
  - Omitted considering Property 1.
  - Each node is selected as the centroid.

- **Grouping** selects a set of neighbors for each centroid.
  - Selecting $k$ nearest *bbox-neighbors* of $u_i$ as the neighbors.
  - Grouping returns a list of neighbors $E_i \in \mathbb{R}^k$ for each centroid $u_i$.

- **Encoding** is to encode the new centroid feature using the original one and the local feature aggregated from the neighbors of the centroid.
  - $v'_{ic} = \max_{j \in E_i} \sigma(\theta_c \cdot \text{CONCAT}(v_i, v_i - v_j, v_i - v_r))$
  - followed by a Squeeze-and-Excitation (SE) block

---

TreeConv

Illustration of TreeConv. Brighter blocks indicate Grouping and darker blocks indicate Encoding.
TreeConv vs. existing methods.

<table>
<thead>
<tr>
<th>Sampling</th>
<th>Grouping</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet⁴</td>
<td>-</td>
<td>( v'_{ic} = \sigma(\theta_c v_i) )</td>
</tr>
<tr>
<td>PointNet++⁵</td>
<td>Fathest Point Sampling (FPS)</td>
<td>( v'<em>{ic} = \max</em>{j \in E_i} \sigma(\theta_c v_j) )</td>
</tr>
<tr>
<td>PointCNN⁶</td>
<td>Random/FPS</td>
<td>( v'_{ic} = \text{Conv}(X \times \theta(v_i - v_j)) )</td>
</tr>
<tr>
<td>DGCNN⁷</td>
<td>k nearest neighbor</td>
<td>( v'<em>{ic} = \max</em>{j \in E_i} \sigma(\theta_c \cdot \text{CONCAT}(v_i, v_i - v_j)) )</td>
</tr>
<tr>
<td>Our work</td>
<td>k bounding box neighbor</td>
<td>( v'<em>{ic} = \max</em>{j \in E_i} \sigma(\theta_c \cdot \text{CONCAT}(v_i, v_i - v_j, v_i - v_r)) )</td>
</tr>
</tbody>
</table>

---


Illustration of TreeNet Architecture for the cloud embedding.

Normalization: \( \tilde{v}_i = \frac{v_i - v_r}{d_{\text{max}}} \).
**Algorithm selection & parameter prediction**

### Algorithm selection

\[ y = \text{softmax}(W_3 \sigma(W_2 \sigma(W_1 H_c + b_1) + b_2)) \],

### Parameter prediction

- 20 valid parameter \( \epsilon_i, i \in \{1, \ldots, 20\} \) candidates for SALT.
- Following similar structure with algorithm selection to obtain the output \( y \in \mathbb{R}^{20} \).
- Given the output \( y \), the predicted parameter \( \epsilon \) is calculated by an element-wise summation and can be formulated as \( \epsilon = \sum_{i=1}^{20} \epsilon_i \cdot y_i \).
- The predicted parameter guides the routing tree construction by a simple heuristic rule.
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.
Comparison to existing methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall*</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet</td>
<td>54.13</td>
<td>53.95</td>
<td>1.91</td>
</tr>
<tr>
<td>PointNet++</td>
<td>81.31</td>
<td>82.50</td>
<td>2.65</td>
</tr>
<tr>
<td>PointCNN</td>
<td>62.18</td>
<td>64.24</td>
<td>1.16</td>
</tr>
<tr>
<td>DGCNN</td>
<td>92.24</td>
<td>94.62</td>
<td>11.84</td>
</tr>
<tr>
<td>TreeNet w.o. Nor</td>
<td>87.22</td>
<td>88.62</td>
<td>15.69</td>
</tr>
<tr>
<td>TreeNet w.o. global</td>
<td>92.40</td>
<td>94.63</td>
<td>25.53</td>
</tr>
<tr>
<td>TreeNet w. knn</td>
<td>92.58</td>
<td>94.79</td>
<td>26.76</td>
</tr>
<tr>
<td>TreeNet</td>
<td><strong>94.09</strong></td>
<td><strong>95.38</strong></td>
<td><strong>50.74</strong></td>
</tr>
</tbody>
</table>
Comparison to SALT & PD-II (shallowness & normalized PL)

<table>
<thead>
<tr>
<th>V</th>
<th>Method</th>
<th>WL deg.</th>
<th>0%</th>
<th>5%</th>
<th>10%</th>
<th>15%</th>
<th>20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>PD-II</td>
<td>1.0606</td>
<td>1.0369</td>
<td>1.0240</td>
<td>1.0161</td>
<td>1.0114</td>
<td></td>
</tr>
<tr>
<td>SALT</td>
<td>1.0462</td>
<td>1.0216</td>
<td>1.0078</td>
<td>1.0022</td>
<td>1.0006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>1.0461</td>
<td>1.0210</td>
<td>1.0074</td>
<td>1.0021</td>
<td>1.0005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp. (%)</td>
<td>0.28</td>
<td>2.62</td>
<td>4.40</td>
<td>5.42</td>
<td>8.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp.∗ (%)</td>
<td>0.32</td>
<td>3.04</td>
<td>5.14</td>
<td>6.75</td>
<td>9.94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Med.</td>
<td>PD-II</td>
<td>1.3849</td>
<td>1.2518</td>
<td>1.1688</td>
<td>1.1176</td>
<td>1.0851</td>
<td></td>
</tr>
<tr>
<td>SALT</td>
<td>1.3456</td>
<td>1.1775</td>
<td>1.0838</td>
<td>1.0391</td>
<td>1.0181</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SALT∗</td>
<td>1.3463</td>
<td>1.1815</td>
<td>1.0868</td>
<td>1.0410</td>
<td>1.0192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>1.3435</td>
<td>1.1689</td>
<td>1.0790</td>
<td>1.0370</td>
<td>1.0172</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp. (%)</td>
<td>0.62</td>
<td>4.85</td>
<td>5.72</td>
<td>5.57</td>
<td>5.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp.∗ (%)</td>
<td>0.80</td>
<td>6.95</td>
<td>8.99</td>
<td>9.92</td>
<td>10.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large</td>
<td>PD-II</td>
<td>2.1660</td>
<td>1.7169</td>
<td>1.4771</td>
<td>1.3438</td>
<td>1.2603</td>
<td></td>
</tr>
<tr>
<td>SALT</td>
<td>1.9796</td>
<td>1.3549</td>
<td>1.1568</td>
<td>1.0727</td>
<td>1.0358</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SALT∗</td>
<td>1.8083</td>
<td>1.3689</td>
<td>1.1648</td>
<td>1.0771</td>
<td>1.0382</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>1.7755</td>
<td>1.3339</td>
<td>1.1481</td>
<td>1.0690</td>
<td>1.0341</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp. (%)</td>
<td>2.77</td>
<td>5.91</td>
<td>5.35</td>
<td>5.11</td>
<td>4.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp.∗ (%)</td>
<td>4.06</td>
<td>9.50</td>
<td>10.12</td>
<td>10.52</td>
<td>10.77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Huge</td>
<td>PD-II</td>
<td>2.2921</td>
<td>1.8221</td>
<td>1.6193</td>
<td>1.5037</td>
<td>1.5064</td>
<td></td>
</tr>
<tr>
<td>SALT</td>
<td>2.0511</td>
<td>1.4938</td>
<td>1.2083</td>
<td>1.0987</td>
<td>1.0466</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SALT∗</td>
<td>2.0291</td>
<td>1.4567</td>
<td>1.2183</td>
<td>1.1039</td>
<td>1.0489</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>1.9793</td>
<td>1.4152</td>
<td>1.1975</td>
<td>1.0941</td>
<td>1.0444</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp. (%)</td>
<td>3.15</td>
<td>5.61</td>
<td>5.17</td>
<td>4.69</td>
<td>4.64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp.∗ (%)</td>
<td>4.85</td>
<td>9.09</td>
<td>9.50</td>
<td>9.47</td>
<td>9.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>PD-II</td>
<td>1.0156</td>
<td>1.0099</td>
<td>1.0065</td>
<td>1.0044</td>
<td>1.0031</td>
<td></td>
</tr>
<tr>
<td>SALT</td>
<td>1.0113</td>
<td>1.0055</td>
<td>1.0020</td>
<td>1.0006</td>
<td>1.0002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SALT∗</td>
<td>1.0113</td>
<td>1.0055</td>
<td>1.0020</td>
<td>1.0006</td>
<td>1.0002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>1.0112</td>
<td>1.0053</td>
<td>1.0019</td>
<td>1.0005</td>
<td>1.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp. (%)</td>
<td>0.25</td>
<td>2.86</td>
<td>4.88</td>
<td>6.57</td>
<td>10.55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imp.∗ (%)</td>
<td>0.29</td>
<td>3.38</td>
<td>5.83</td>
<td>8.29</td>
<td>12.75</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Imp. (%) = Improved by % compared to PD-II
Imp.∗ (%) = Improved by % compared to SALT

**Note:** The table compares the performance of different methods (PD-II, SALT, Ours) across various dataset sizes (Small, Med., Large, Huge, All) and WL degrees (0% - 20%). The table highlights improvements in shallowness and normalized PL metrics.
## Runtime

Runtime comparison with SALT and SALT*.

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>SALT</th>
<th>SALT *</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>58.46%</td>
<td>40.05%</td>
<td>1.05%</td>
</tr>
<tr>
<td>Med.</td>
<td>69.83%</td>
<td>75.97%</td>
<td>11.13%</td>
</tr>
<tr>
<td>Large</td>
<td>68.44%</td>
<td>14.09%</td>
<td>17.47%</td>
</tr>
<tr>
<td>Huge</td>
<td>24.39%</td>
<td>9.67%</td>
<td>21.04%</td>
</tr>
<tr>
<td>All</td>
<td>65.94%</td>
<td>68.44%</td>
<td>17.47%</td>
</tr>
</tbody>
</table>

Runtime breakdown of our framework.
Outline

Case Study 1: Routing Tree Construction

Case Study 2: Hotspot Detection

Conclusion
Pattern Matching based Hotspot Detection

Fast and accurate

[Yu+, ICCAD'14] [Nosato+, JM3'14] [Su+, TCAD'15]

Fuzzy pattern matching [Wen+, TCAD'14]

Hard to detect non-seen pattern
Pattern Matching based Hotspot Detection

- Fast and accurate
- [Yu+, ICCAD’14] [Nosato+, JM3’14] [Su+, TCAD’15]
- Fuzzy pattern matching [Wen+, TCAD’14]
- Hard to detect non-seen pattern
Classification based Hotspot Detection

- Predict new patterns
  - Decision-tree, ANN, SVM, Boosting...
- Extract layout features
- Hotspot detection model
- Classification

[Drmanac+, DAC'09] [Ding+, TCAD'12] [Yu+, JM3'15] [Matsunawa+, SPIE'15] [Yu+, TCAD'15]

- Hard to balance accuracy and false-alarm
Classification based Hotspot Detection

- Predict new patterns
- Decision-tree, ANN, SVM, Boosting ...
- [Drmanac+,DAC’09] [Ding+,TCAD’12] [Yu+,JM3’15] [Matsunawa+,SPIE’15] [Yu+,TCAD’15]
- Hard to balance accuracy and false-alarm
(a) Density-based encoding [SPIE’15] 8

(b) Concentric circle sampling [ICCAD’16] 9

(c) Squish pattern [ASPDAC’19] 10

---


Simplified CNN Architecture [DAC’17]

Feature Tensor Generation:

- Clip Partition
- Discrete Cosine Transform
- Discarding High Frequency Components
- Feature Tensor

---

Simplified CNN Architecture [DAC’17]\(^\text{11}\)

**Feature Tensor Generation:**

- Clip Partition
- Discrete Cosine Transform
- Discarding High Frequency Components
- Feature Tensor

---

Simplified CNN Architecture [DAC’17]^{11}

Feature Tensor Generation:

- Clip Partition
- Discrete Cosine Transform
- Discarding High Frequency Components
- Feature Tensor

---

Simplified CNN Architecture [DAC’17]

Feature Tensor

- $k$-channel hyper-image
- Compatible with CNN
- Storage and computational efficiency

<table>
<thead>
<tr>
<th>Layer</th>
<th>Kernel Size</th>
<th>Stride</th>
<th>Output Node #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1-1</td>
<td>3</td>
<td>1</td>
<td>$12 \times 12 \times 16$</td>
</tr>
<tr>
<td>conv1-2</td>
<td>3</td>
<td>1</td>
<td>$12 \times 12 \times 16$</td>
</tr>
<tr>
<td>maxpooling1</td>
<td>2</td>
<td>2</td>
<td>$6 \times 6 \times 16$</td>
</tr>
<tr>
<td>conv2-1</td>
<td>3</td>
<td>1</td>
<td>$6 \times 6 \times 32$</td>
</tr>
<tr>
<td>conv2-2</td>
<td>3</td>
<td>1</td>
<td>$6 \times 6 \times 32$</td>
</tr>
<tr>
<td>maxpooling2</td>
<td>2</td>
<td>2</td>
<td>$3 \times 3 \times 32$</td>
</tr>
<tr>
<td>fc1</td>
<td>N/A</td>
<td>N/A</td>
<td>250</td>
</tr>
<tr>
<td>fc2</td>
<td>N/A</td>
<td>N/A</td>
<td>2</td>
</tr>
</tbody>
</table>
Examples of the transformation from layout to point cloud. left: original GDSII layout, the hotspot is marked as red rectangle. right: transformed point cloud.
Workflow

Overall flow of point cloud hotspot detection model.

- Hotspot box proposal generation
  - Obtain point-wise features by PointNet++;
  - One segmentation head for predicting foreground points information and one box regression head for generating hotspot proposals;

- Hotspot box refinement
  - The embedding is further used to refine hotspot proposals and predict confidence for each proposal;
## Preliminary results

<table>
<thead>
<tr>
<th>Bench</th>
<th>Faster R-CNN(^{12}) Accu (%)</th>
<th>Faster R-CNN(^{12}) FA</th>
<th>Faster R-CNN(^{12}) Time (s)</th>
<th>TCAD'19(^{13}) Accu (%)</th>
<th>TCAD'19(^{13}) FA</th>
<th>TCAD'19(^{13}) Time (s)</th>
<th>TCAD'20(^{14}) Accu (%)</th>
<th>TCAD'20(^{14}) FA</th>
<th>TCAD'20(^{14}) Time (s)</th>
<th>PCloud-HSD Accu (%)</th>
<th>PCloud-HSD FA</th>
<th>PCloud-HSD Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 2</td>
<td>1.8</td>
<td>3</td>
<td>1.0</td>
<td>77.78</td>
<td>48</td>
<td>60.0</td>
<td>93.02</td>
<td>17</td>
<td>2.0</td>
<td>83.1</td>
<td>36</td>
<td>1.6</td>
</tr>
<tr>
<td>Case 3</td>
<td>57.1</td>
<td>74</td>
<td>11.0</td>
<td>91.20</td>
<td>263</td>
<td>265.0</td>
<td>94.5</td>
<td>34</td>
<td>10.0</td>
<td>88.4</td>
<td>89</td>
<td>8.2</td>
</tr>
<tr>
<td>Case 4</td>
<td>6.9</td>
<td>69</td>
<td>8.0</td>
<td>100</td>
<td>511</td>
<td>428.0</td>
<td>100</td>
<td>201</td>
<td>6.0</td>
<td>100</td>
<td>294</td>
<td>5.5</td>
</tr>
<tr>
<td>Average</td>
<td>21.9</td>
<td>48.7</td>
<td>6.67</td>
<td>89.66</td>
<td>274</td>
<td>251</td>
<td>95.8</td>
<td>84</td>
<td>6</td>
<td>90.5</td>
<td>139.6</td>
<td>5.1</td>
</tr>
<tr>
<td>Ratio</td>
<td>0.23</td>
<td>0.58</td>
<td>1.11</td>
<td>0.94</td>
<td>3.26</td>
<td>41.83</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.95</td>
<td>1.66</td>
<td><strong>0.85</strong></td>
</tr>
</tbody>
</table>


\(^{13}\)Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, et al. (2019). “Layout hotspot detection with feature tensor generation and deep biased learning”. In: *IEEE TCAD* 38.6, pp. 1175–1187.

Outline

Case Study 1: Routing Tree Construction

Case Study 2: Hotspot Detection

Conclusion
Conclusion

- We formalize **special properties** of the point cloud for the routing tree construction;
- We propose an **adaptive flow** for the routing tree construction, which uses the cloud embedding to **select the best approach** and **predict the best parameter**;
- We further study the possibility of point cloud based hotspot detection.
- More applications to explore...
Thank You!