International Symposium on Physical Design

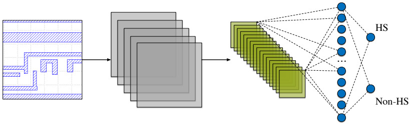
Learning Point Clouds in EDA

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

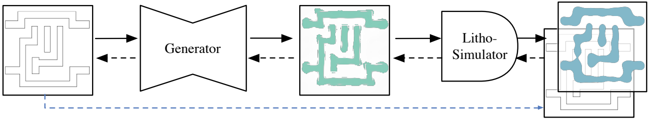
The Chinese University of Hong Kong

Challenge: Irregular Structure Learning

- Verification [Yang et.al TCAD'2018]

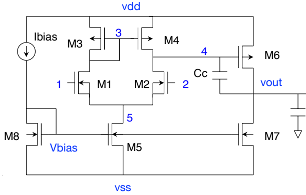


- Mask optimization [Yang et.al DAC'2018]

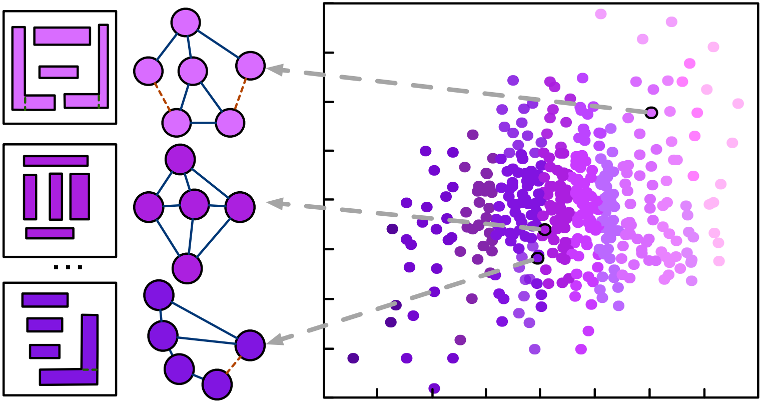


More Considerations

- 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.

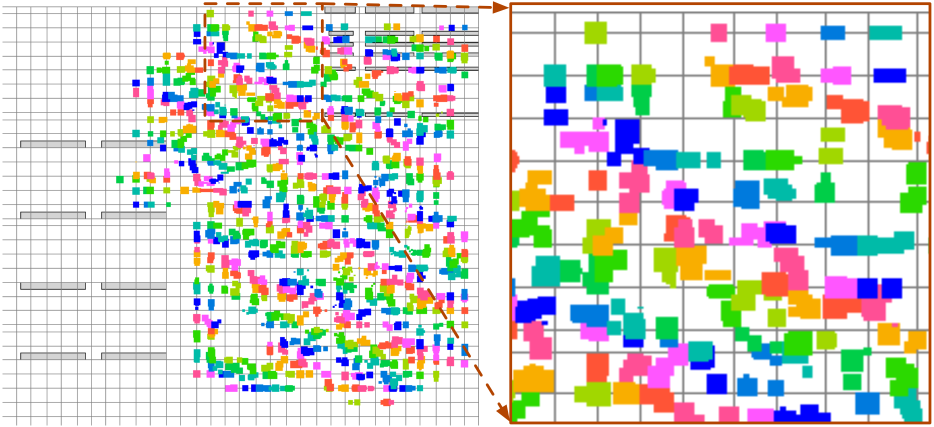


Irregular data representation in EDA: Graph



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

Graph

- A set of vertices and edges;
- Strictly constrains inter-connected relationships: requires the definition of connections (edges) among objects (nodes);

Point Cloud

- A set of data points in space;
- Directly preserves the original geometric information without any discretization or misinterpretation;

Previous works: Deep learning in EDA

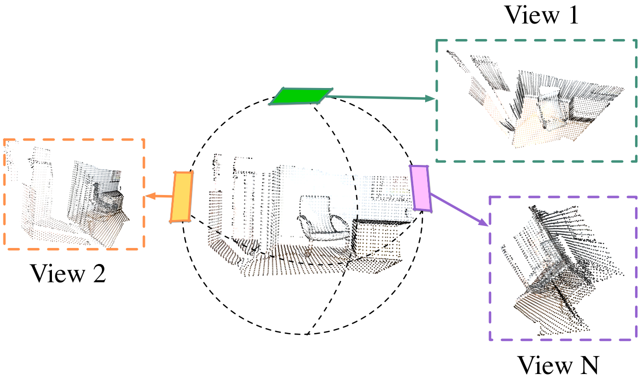
By topics

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

Graph Neural Networks

- Message-passing scheme;
- Netlist;
- Layout;

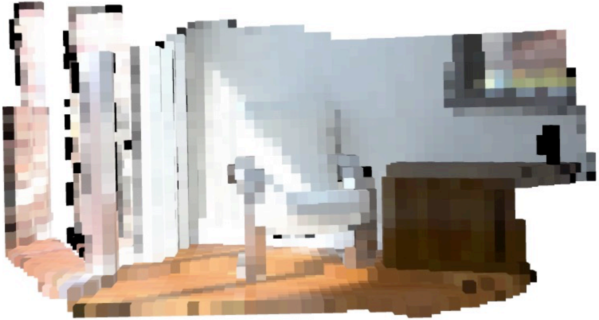
Previous works: Point Cloud Learning with Neural Networks



Multi-view-based methods:

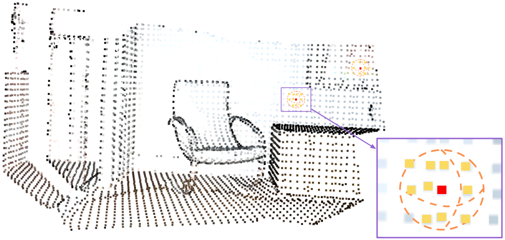
- Transform a 3D point cloud into multiple views through **projection**;
- Extracted view-based features are fused together to generate a cloud embedding;

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;



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

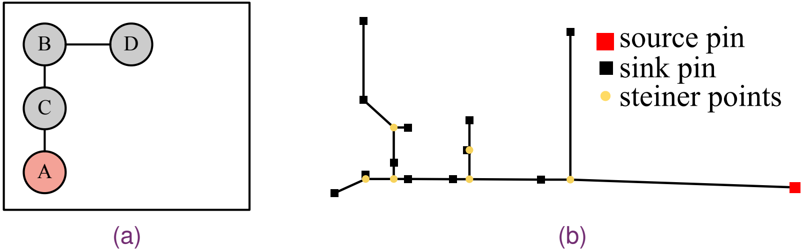
Case Study 1: Routing Tree Construction

Case Study 2: Hotspot Detection

Conclusion

Case study 1: Routing Routing Tree Construction

Routing Tree Construction: Given a input net $\boldsymbol{V} = \{v_0, \boldsymbol{V}_s\}$, v_0 is the source (red node) and $\boldsymbol{V}_s$ 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).
- $lightness = \frac{w(T)}{w(MST(G))}$, $w(\cdot)$ is the total weight.

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

- Shallowness $= \max\{\frac{d_T(v_0, v)}{d_G(v_0, v)} | v \in \boldsymbol{V_s}\}$, G is the connected weighted routing graph.
- Normalized path length $= \frac{\sum_{v \in \boldsymbol{V}} d_T(v_0, v)}{\sum_{v \in \boldsymbol{V}} d_G(v_0, v)}$.

Non-trivial questions in the routing tree construction

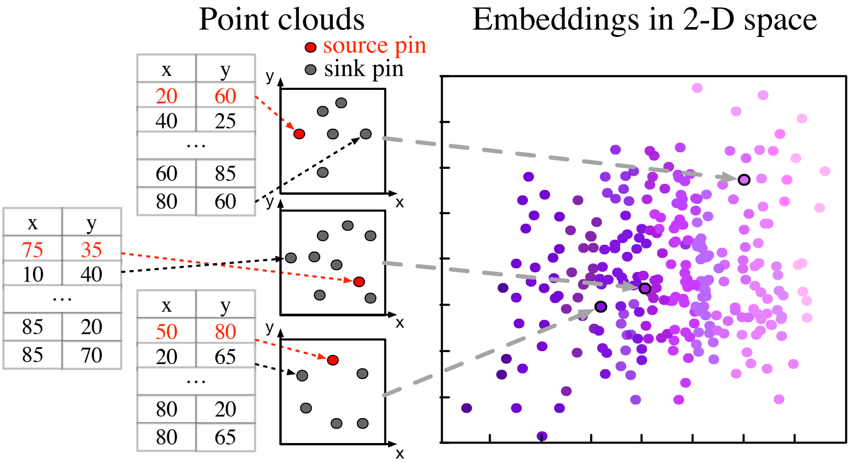
Best algorithm?

- 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?

Point cloud and its embedding



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[1] 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 algorithm** 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.

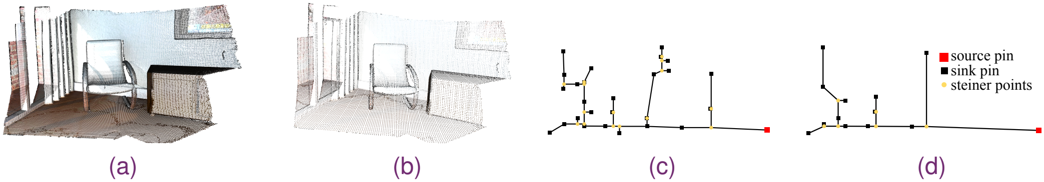
[1] Gengjie Chen and Evangeline FY Young (2019). "SALT: provably good routing topology by a novel steiner shallow-light tree algorithm". In: *IEEE TCAD*.

[2] Charles J Alpert et al. (2018). "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees". In: *Proc. ISPD*, pp. 10–17.

Property 1: Down-sampling

Property

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 $f(d(\boldsymbol{V}))$.



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.

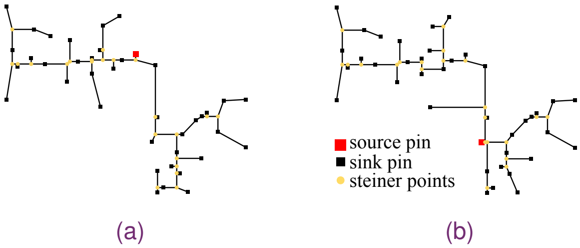
Property 2 & 3: Permutation

Property

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

Property

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

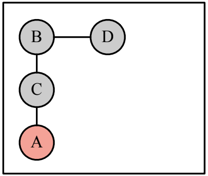


Examples of the routing trees with the same node coordinates but different source (highlighted by red).

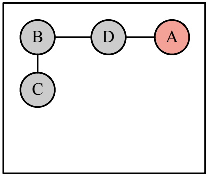
Property 4: Inequality of the same V_s

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.



(a)



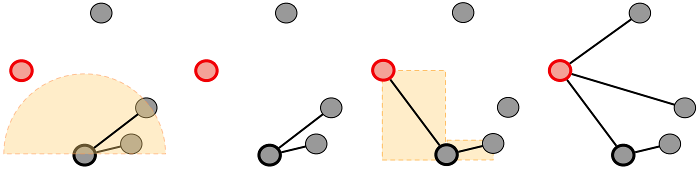
(b)

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

Property

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}.



(a) ball (b) k-nn (c) k-bbox (d) routing tree

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
  - Omited 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 $\boldsymbol{E}_i \in \mathbb{R}^k$ 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.
  - $v'_{ic} = \max_{j \in E_i} \sigma(\boldsymbol{\theta}_c \cdot \texttt{CONCAT}(\boldsymbol{v}_i, \boldsymbol{v}_i - \boldsymbol{v}_j, \boldsymbol{v}_i - \boldsymbol{v}_r))$
  - followed by a Squeeze-and-Excitation (SE) block[3]

[3] Jie Hu, Li Shen, and Gang Sun (2018). "Squeeze-and-excitation networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141.

TreeConv

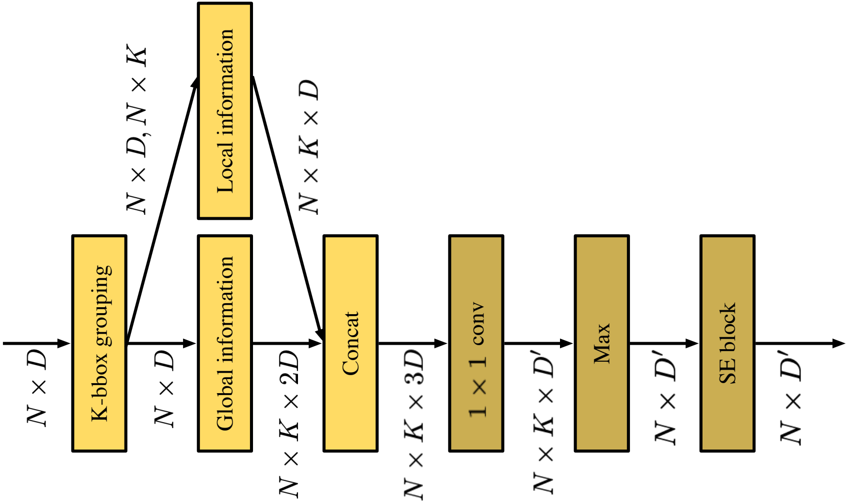


Illustration of TreeConv. Brighter blocks indicate `Grouping` and darker blocks indicate `Encoding`.

TreeConv vs. existing methods.

	Sampling	Grouping	Encoding
PointNet[4]	-	-	$v'_{ic} = \sigma(\boldsymbol{\theta}_c \boldsymbol{v}_i)$
PointNet++[5]	Fathest Point Sampling (FPS)	ball query's local neighborhood	$v'_{ic} = \max_{j\in E_i} \sigma(\boldsymbol{\theta}_c \boldsymbol{v}_j)$
PointCNN[6]	Random/FPS	k nearest neighbor	$\boldsymbol{v}'_i = \mathtt{Conv}(X \times \boldsymbol{\theta}(\boldsymbol{v}_i - \boldsymbol{v}_j))$
DGCNN[7]	-	k nearest neighbor	$v'_{ic} = \max_{j\in E_i} \sigma(\boldsymbol{\theta}_c \cdot \mathtt{CONCAT}(\boldsymbol{v}_i, \boldsymbol{v}_i - \boldsymbol{v}_j))$
Our work	-	k bounding box neighbor	$v'_{ic} = \max_{j\in E_i} \sigma(\boldsymbol{\theta}_c \cdot \mathtt{CONCAT}(\boldsymbol{v}_i, \boldsymbol{v}_i - \boldsymbol{v}_j, \boldsymbol{v}_i - \boldsymbol{v}_r))$

[4] Charles R Qi et al. (2017). "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proc. CVPR*, pp. 652–660.

[5] Charles Ruizhongtai Qi et al. (2017). "PointNet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in Neural Information Processing Systems*, pp. 5099–5108.

[6] Yangyan Li et al. (2018). "PointCNN: Convolution on x-transformed points". In: *Advances in Neural Information Processing Systems*, pp. 820–830.

[7] Yue Wang et al. (2019). "Dynamic graph CNN for learning on point clouds". In: *ACM Transactions on Graphics* 38.5, pp. 1–12.

TreeNet

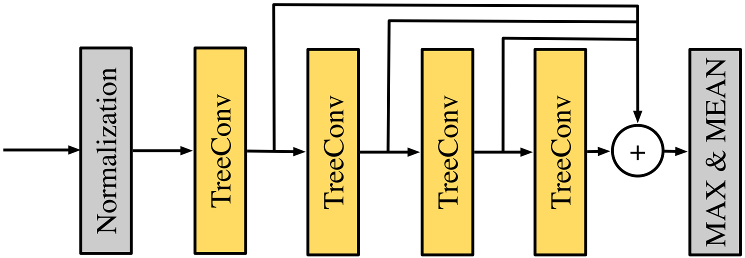


Illustration of TreeNet Architecture for the cloud embedding.

- Normalization: $\tilde{\boldsymbol{v}}_i = \dfrac{\boldsymbol{v}_i - \boldsymbol{v}_r}{d_{max}}$.

Algorithm selction & parameter predition

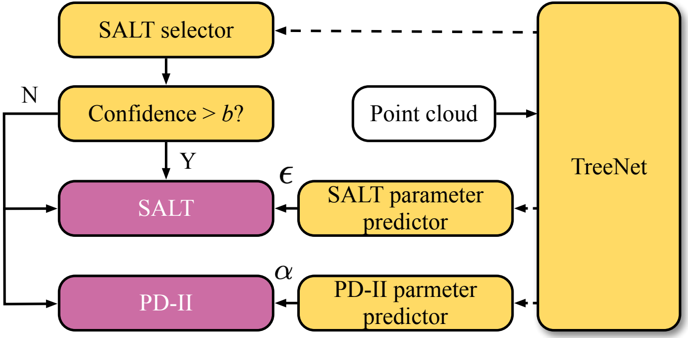
Algorithm selction

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

Parameter predition

- 20 valid parameter $\epsilon_i, i \in \{1, ..., 20\}$ candidates for SALT
- Following similar structure with algorithm selection to obtain the output $\boldsymbol{y} \in \mathbb{R}^{20}$.
- Given the output $\boldsymbol{y}$, the predicted parameter ϵ 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

Framework



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

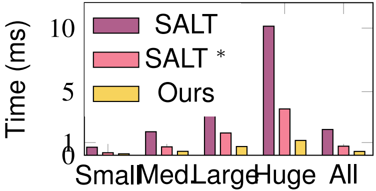
Method	Accuracy	Precision	Recall*
PointNet	54.13	53.95	1.91
PointNet++	81.31	82.50	2.65
PointCNN	62.18	64.24	1.16
DGCNN	92.24	94.62	11.84
TreeNet w.o. Nor	87.22	88.62	15.69
TreeNet w.o. global	92.40	94.63	25.53
TreeNet w. knn	92.58	94.79	26.76
TreeNet	**94.09**	**95.38**	**50.74**

Comparison to SALT & PD-II (shallowness & normalized PL)

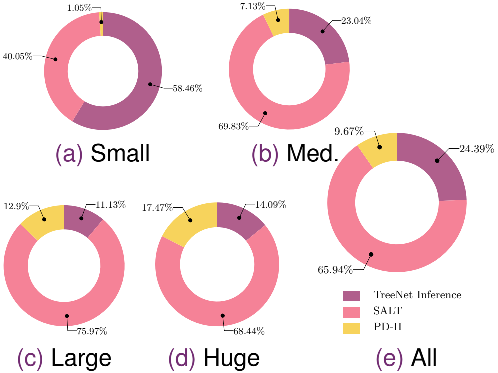
|V|	Method	WL deg.				
		0%	5%	10%	15%	20%
Small	PD-II	1.0606	1.0369	1.0240	1.0161	1.0114
	SALT	1.0462	1.0216	1.0078	1.0022	1.0006
	SALT*	1.0462	1.0216	1.0079	1.0023	1.0006
	Ours	1.0461	1.0210	1.0074	1.0021	1.0005
	Imp. (%)	**0.28**	**2.62**	**4.40**	**5.42**	**8.25**
	Imp.* (%)	**0.32**	**3.04**	**5.14**	**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**

|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



Runtime comparison with SALT and SALT*.



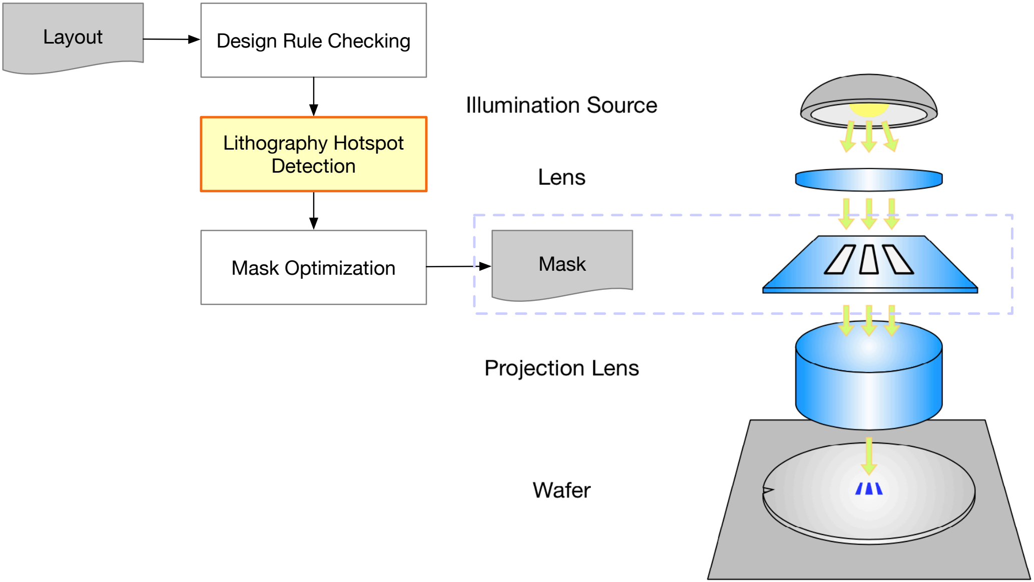
Runtime breakdown of our framework.

Outline

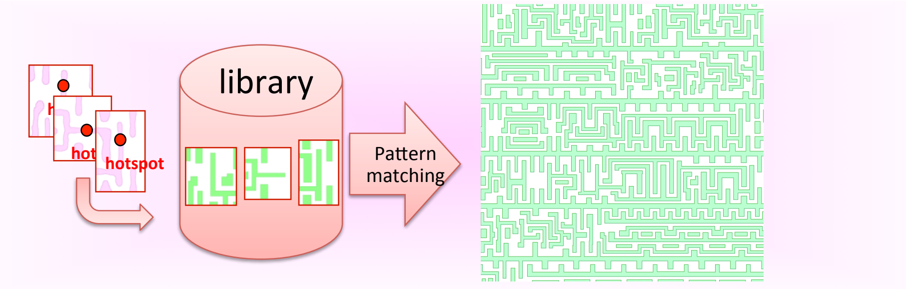
Case Study 1: Routing Tree Construction

Case Study 2: Hotspot Detection

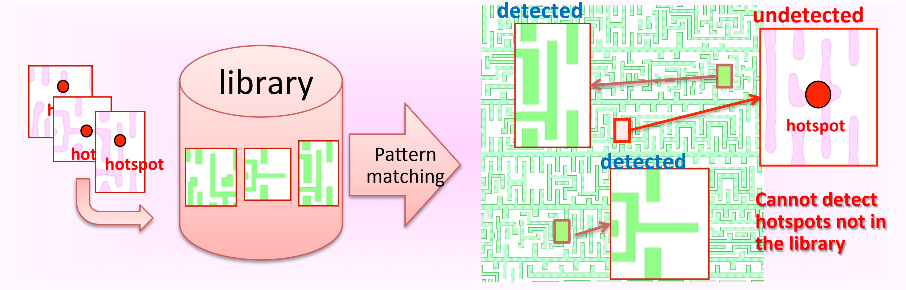
Conclusion


Layout
Design Rule Checking
Lithography Hotspot Detection
Mask Optimization
Mask
Illumination Source
Lens
Projection Lens
Wafer

Pattern Matching based Hotspot Detection

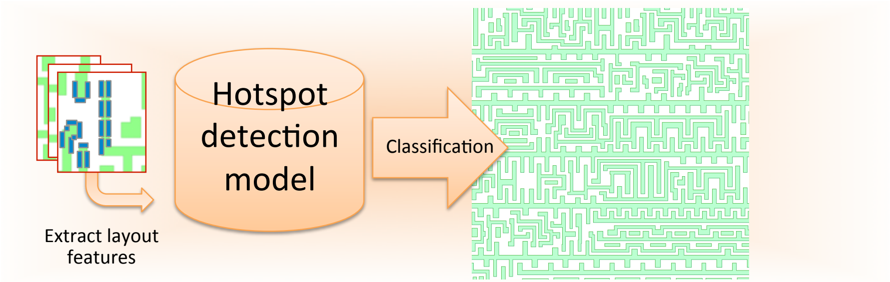


Pattern Matching based Hotspot Detection

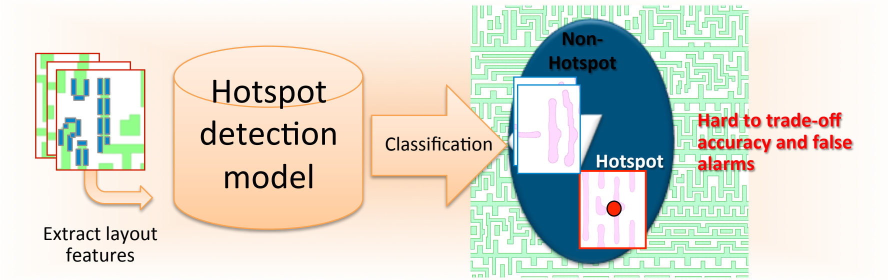


- Fast and accurate
- [Yu+,ICCAD'14] [Nosato+,JM3'14] [Su+,TCAD'15]
- Fuzzy pattern matching [Wen+,TCAD'14]
- Hard to detect non-seen pattern

Classification based Hotspot Detection

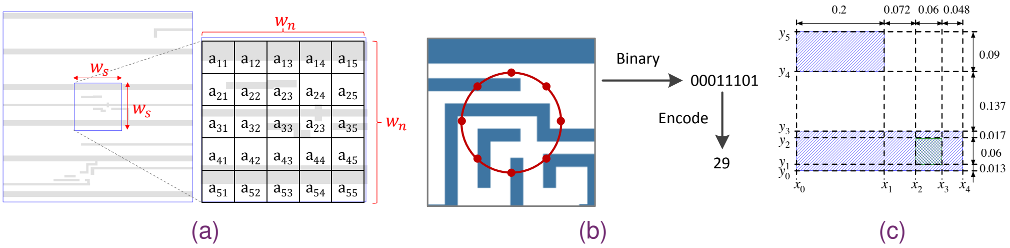


Classification based Hotspot Detection



- 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

HSD-Research: New Representation



- (a) Density-based encoding [SPIE'15][8]
- (b) Concentric circle sampling [ICCAD'16][9]
- (c) Squish pattern [ASPDAC'19][10]

---

[8] Tetsuaki Matsunawa et al. (2015). "A new lithography hotspot detection framework based on AdaBoost classifier and simplified feature extraction". In: *Proc. SPIE*. vol. 9427.

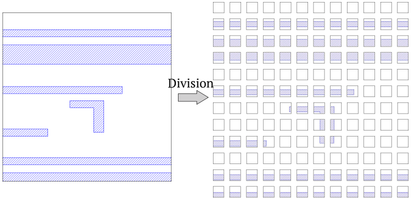
[9] Hang Zhang, Bei Yu, and Evangeline F. Y. Young (2016). "Enabling Online Learning in Lithography Hotspot Detection with Information-Theoretic Feature Optimization". In: *Proc. ICCAD*, 47:1–47:8.

[10] Haoyu Yang, Piyush Pathak, et al. (2019). "Detecting multi-layer layout hotspots with adaptive squish patterns". In: *Proc. ASPDAC*, pp. 299–304.

Simplified CNN Architecture [DAC'17][11]

Feature Tensor Generation:

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

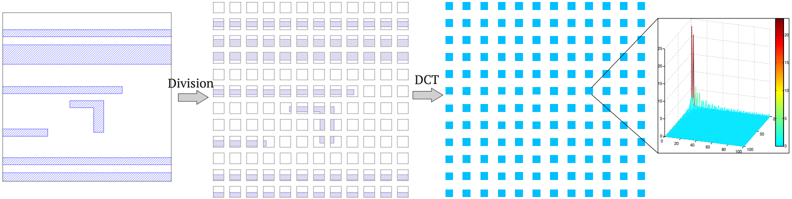


[11] Haoyu Yang, Jing Su, Yi Zou, Bei Yu, et al. (2017). "Layout Hotspot Detection with Feature Tensor Generation and Deep Biased Learning". In: *Proc. DAC*, 62:1–62:6.

Simplified CNN Architecture [DAC'17][11]

Feature Tensor Generation:

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

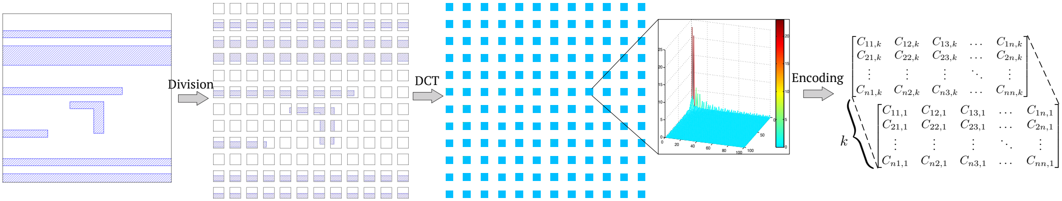


[11] Haoyu Yang, Jing Su, Yi Zou, Bei Yu, et al. (2017). "Layout Hotspot Detection with Feature Tensor Generation and Deep Biased Learning". In: *Proc. DAC*, 62:1–62:6.

Simplified CNN Architecture [DAC’17][11]

Feature Tensor Generation:

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



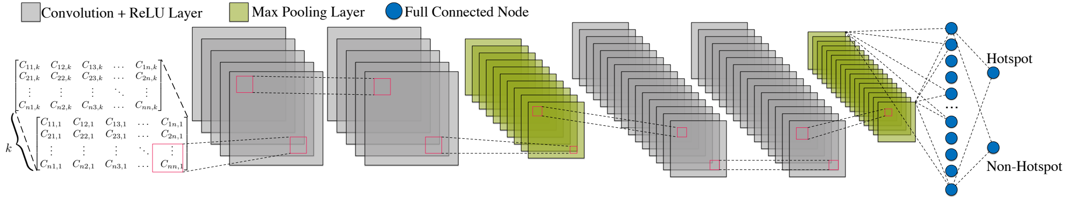
[11] Haoyu Yang, Jing Su, Yi Zou, Bei Yu, et al. (2017). “Layout Hotspot Detection with Feature Tensor Generation and Deep Biased Learning”. In: *Proc. DAC*, 62:1–62:6.

Simplified CNN Architecture [DAC’17]

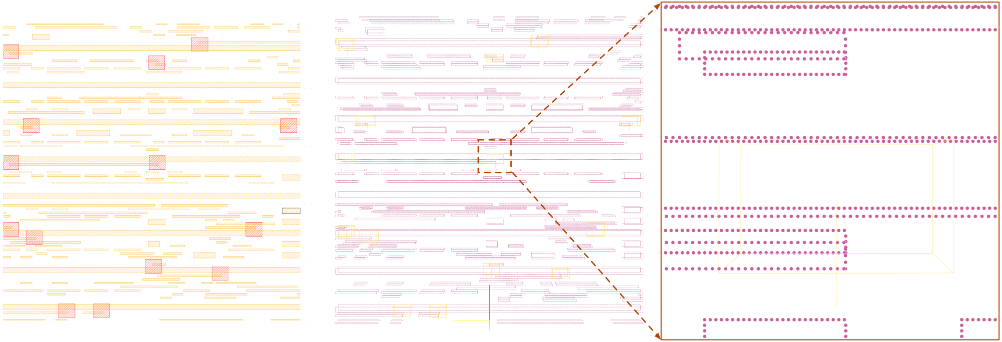
Feature Tensor

- k-channel hyper-image
- Compatible with CNN
- Storage and computional efficiency

Layer	Kernel Size	Stride	Output Node #
conv1-1	3	1	$12 \times 12 \times 16$
conv1-2	3	1	$12 \times 12 \times 16$
maxpooling1	2	2	$6 \times 6 \times 16$
conv2-1	3	1	$6 \times 6 \times 32$
conv2-2	3	1	$6 \times 6 \times 32$
maxpooling2	2	2	$3 \times 3 \times 32$
fc1	N/A	N/A	250
fc2	N/A	N/A	2



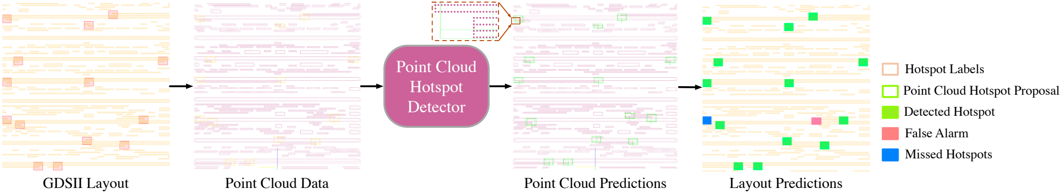
Case Study 2: Point-Cloud based Hotspot Detection



(d) (e)

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

Bench	Faster R-CNN[12]			TCAD'19[13]			TCAD'20[14]			PCloud-HSD		
	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)
`Case2`	1.8	3	1.0	77.78	48	60.0	93.02	17	2.0	83.1	36	1.6
`Case3`	57.1	74	11.0	91.20	263	265.0	94.5	34	10.0	88.4	89	8.2
`Case4`	6.9	69	8.0	100	511	428.0	100	201	6.0	100	294	5.5
Average	21.9	48.7	6.67	89.66	274	251	95.8	84	6	90.5	139.6	5.1
Ratio	0.23	0.58	1.11	0.94	3.26	41.83	**1**	**1**	1	0.95	1.66	**0.85**

[12] Shaoqing Ren et al. (2015). "Faster R-CNN: Towards real-time object detection with region proposal networks". In: *Proc. NIPS*, pp. 91–99.

[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.

[14] Ran Chen et al. (2019). "Faster Region-based Hotspot Detection". In: *Proc. DAC*, 146:1–146:6.

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!