Adaptive Layout Decomposition with Graph Embedding Neural Networks

Wei Li\textsuperscript{1}, Jialu Xia\textsuperscript{1}, Yuzhe Ma\textsuperscript{1}, Jialu Li\textsuperscript{1}, Yibo Lin\textsuperscript{2}, Bei Yu\textsuperscript{1}

\textsuperscript{1}The Chinese University of Hong Kong
\textsuperscript{2}Peking University
Outline

Background & Introduction

Algorithms

Results

Conclusion
Outline

Background & Introduction

Algorithms

Results

Conclusion
Multiple Patterning Lithography Decomposition

An example of the layout and corresponding decomposition results
Uncolorable case: Conflict

An example of the uncolorable case
One possible solution for the uncolorable case: Stitch

An example of the stitch candidate and stitch
Problem Formulation

\[
\min_x \sum c_{ij} + \alpha \sum s_{ij},
\]

s.t. \( c_{ij} = (x_i == x_j), \quad \forall e_{ij} \in CE, \tag{1a} \)
\[
s_{ij} = (x_i \neq x_j), \quad \forall e_{ij} \in SE, \tag{1b} \)
\[
x_i \in \{0, 1, \ldots, k\}, \quad \forall x_i \in x, \tag{1c} \)

\( x \): color assigned to each node, \( CE \): conflict edge set, \( SE \): stitch edge set.
Integer Linear Programming (ILP)*

\[
\begin{align*}
&\min \sum_{e_{ij} \in \text{CE}} c_{ij} + \alpha \sum_{e_{ij} \in \text{SE}} s_{ij} \\
\text{s.t.} &\quad x_{i1} + x_{i2} \leq 1, x_{ij} \in \{0, 1\}. \\
&\quad x_{i1} + x_{j1} \leq 1 + c_{ij1}, x_{i2} + x_{j2} \leq 1 + c_{ij2}, \\
&\quad (1 - x_{i1}) + (1 - x_{j1}) \leq 1 + c_{ij1}, \\
&\quad (1 - x_{i2}) + (1 - x_{j2}) \leq 1 + c_{ij2}, \\
&\quad c_{ij1} + c_{ij2} \leq 1 + c_{ij}, \\
&\quad |x_{j1} - x_{i1}| \leq s_{ij1}, |x_{j2} - x_{i2}| \leq s_{ij2}, \\
&\quad s_{ij} \geq s_{ij1}, s_{ij} \geq s_{ij2},
\end{align*}
\]

(2a)

(2b)

(2c)

(2d)

(2e)

(2f)

(2g)

(2h)

---

An example of the exact cover-based algorithm

Pros and cons analysis

- **ILP**
  - Pros: Optimal
  - Cons: Bad runtime performance

- **EC**
  - Pros: High efficiency
  - Cons: Degradation of the solution quality

- **Graph matching**
  - Pros: Good performance in both efficiency and quality for small graphs
  - Cons: Graph library size is limited

---

Graph Embedding

An example of graph embeddings of layout graphs, where the graphs are transformed into vector space.
Graph Convolutional Network

\[ u_i^{(l+1)} = ReLU \left( \sum_{j \in N_i} W^{(l)} u_j^{(l)} + u_i^{(l)} \right), \quad (3) \]

- \( u^{(l)} \): node representation at the \( l \)th layer, \( N_i \): neighbours of node \( i \).
- Composed of two modules, aggregator and encoder
- Node embedding: node representation at the final layer
- Graph embedding: obtained from node embedding through some operations such as summation and mean
- Not applicable for heterogeneous graphs
Outline

Background & Introduction

Algorithms

Results

Conclusion
Framework Overview

The online workflow of our framework.

- **Online**: Shown in the figure.
- **Offline**: Model training & Graph library construction.
Relational Graph Convolutional Networks (RGCN)

\[ u_i^{(l+1)} = ReLU \left( \sum_{e \in E} \sum_{j \in N_i^e} W_e^{(l)} u_j^{(l)} + u_i^{(l)} \right) , \]  \hspace{1cm} (4) 

\( E: \{CE \text{ (conflict edge set)}, SE \text{ (stitch edge set)}\} \)

- Neighbours connected by different kinds of edges are assigned to different encoder tracks.
- Applicable for heterogeneous graphs
Graph Embedding Workflow by RGCN

Overview of the process for graph embedding
Offline: Graph Library Construction

What we need?

- Enumerate all possible graphs under a size constraint
- Avoid isomorphic graphs

Rough Algorithm

1. Enumerate all valid graphs under the given size constraint
2. For each graph enumerated, calculate the graph embedding and normalize it
3. Multiply it with the graph embeddings in the library
4. If the maximal value is less than one, insert the graph and corresponding optimal solution by ILP into the library
Online: Graph Matching & Decomposer Selection

Graph Matching

- Similar idea with graph library construction
- Return the optimal solution of the corresponding matched graph whose graph embedding multiplication result is exactly one

Decomposer Selection

\[ y = \arg \max(W_s h + b_s), \]  

\( W_s, b_s: \) trainable weight and bias, \( h: \) graph embedding

Two-class classification problem: ILP or EC
Outline

Background & Introduction

Algorithms

Results

Conclusion
Effectiveness of RGCN

<table>
<thead>
<tr>
<th>Label</th>
<th>Predicted</th>
<th>ILP</th>
<th>EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>100.0%</td>
<td>13</td>
<td>682</td>
</tr>
<tr>
<td>F1-score</td>
<td>0.0367</td>
<td>0</td>
<td>5900</td>
</tr>
</tbody>
</table>

(a) Proposed RGCN

<table>
<thead>
<tr>
<th>Label</th>
<th>Predicted</th>
<th>ILP</th>
<th>EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>15.4%</td>
<td>2</td>
<td>244</td>
</tr>
<tr>
<td>F1-score</td>
<td>0.0154</td>
<td>11</td>
<td>6338</td>
</tr>
</tbody>
</table>

(b) Conventional GCN

- Classify all ‘ILP’ cases correctly and such achieves the optimality
- $2 \times$ F1-score, $6 \times$ Recall
Comparison with state-of-the-art

<table>
<thead>
<tr>
<th>Circuit</th>
<th>ILP</th>
<th>SDP</th>
<th>EC</th>
<th>RGCN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>st#</td>
<td>cn#</td>
<td>cost</td>
<td>time (s)</td>
</tr>
<tr>
<td>C432</td>
<td>4</td>
<td>0</td>
<td>0.4</td>
<td>0.486</td>
</tr>
<tr>
<td>C499</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.063</td>
</tr>
<tr>
<td>C880</td>
<td>7</td>
<td>0</td>
<td>0.7</td>
<td>0.135</td>
</tr>
<tr>
<td>C1355</td>
<td>3</td>
<td>0</td>
<td>0.3</td>
<td>0.121</td>
</tr>
<tr>
<td>C1908</td>
<td>1</td>
<td>0</td>
<td>0.1</td>
<td>0.129</td>
</tr>
<tr>
<td>C2670</td>
<td>6</td>
<td>0</td>
<td>0.6</td>
<td>0.158</td>
</tr>
<tr>
<td>C3540</td>
<td>8</td>
<td>1</td>
<td>1.8</td>
<td>0.248</td>
</tr>
<tr>
<td>C5315</td>
<td>9</td>
<td>0</td>
<td>0.9</td>
<td>0.226</td>
</tr>
<tr>
<td>C6288</td>
<td>205</td>
<td>1</td>
<td>21.5</td>
<td>5.569</td>
</tr>
<tr>
<td>C7552</td>
<td>21</td>
<td>1</td>
<td>3.1</td>
<td>0.872</td>
</tr>
<tr>
<td>S1488</td>
<td>2</td>
<td>0</td>
<td>0.2</td>
<td>0.147</td>
</tr>
<tr>
<td>S38417</td>
<td>54</td>
<td>19</td>
<td>24.4</td>
<td>7.883</td>
</tr>
<tr>
<td>S35932</td>
<td>40</td>
<td>44</td>
<td>48</td>
<td>13.692</td>
</tr>
<tr>
<td>S38584</td>
<td>117</td>
<td>36</td>
<td>47.7</td>
<td>13.494</td>
</tr>
<tr>
<td>S15850</td>
<td>97</td>
<td>34</td>
<td>43.7</td>
<td>11.380</td>
</tr>
</tbody>
</table>

- Obtain the optimal solution in the benchmark
- Runtime is reduced to 12.3% compared to another optimal ILP-based algorithm
The decomposition runtime by the selected decomposer is the major bottleneck.

RGCN inference and graph matching runtime of our framework are actually trivial.

Our method has strong scalability and can be applied to select other more efficient decomposers in the future.
Outline

Background & Introduction

Algorithms

Results

Conclusion
Conclusion

- Graph embedding by RGCN
  - Build the isomorphism-free graph library
  - Match graphs in the library
  - Adaptively select decomposer

- The results show that:
  - The obtained graph embeddings have powerful representation capability
  - Excellent balance between decomposition quality and efficiency
  - Our framework has strong scalability for future incremental selection
Thank You

Wei Li (wli@cse.cuhk.edu.hk)
Yuzhe Ma (yzma@cse.cuhk.edu.hk)
Yibo Lin (yibolin@pku.edu.cn)
Bei Yu (byu@cse.cuhk.edu.hk)