FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm

Chris Chu
Department of Electrical and Computer Engineering
Iowa State University
Rectilinear Steiner minimal tree (RSMT) problem:

- Given pin positions, find a rectilinear Steiner tree with minimum wirelength (WL)

- Very useful in routing of VLSI circuits
- NP-complete
Previous RSMT Algorithms

- **Optimal algorithms:**
  - Hwang, Richards, Winter [ADM 92]
  - Warme, Winter, Zachariasen [AST 00] *GeoSteiner package*

- **Near-optimal algorithms:**
  - Griffith et al. [TCAD 94] *Batched 1-Steiner heuristic (BI1S)*
  - Mandoiu, Vazirani, Ganley [ICCAD 99]

- **Low-complexity algorithms:**
  - Borah, Owens, Irwin [TCAD 94] *Edge-based heuristic, O(n log n)*
  - Kahng et al. [ASPDAC 03] *Batch Greedy Algorithm, O(n log^2 n)*
  - Zhou [ISPD 03] *Spanning graph based, O(n log n)*

- **Algorithms targeting low-degree nets (VLSI applications):**
  - Soukup [Proc. IEEE 81] *Single Trunk Steiner Tree (STST)*
  - Chen et al. [SLIP 02] *Refined Single Trunk Tree (RST-T)*
FLUTE Overview

- **FLUTE** -- Fast LookUp Table Estimation

- Basic idea:
  - LUT to handle nets with a few pins
  - Net breaking technique to recursively break large nets

- Low degree nets are handled extremely well:
  - Optimal and extremely efficient for nets up to 9 pins
  - Still very accurate and fast for nets up to 100 pins

- So FLUTE is especially suitable for VLSI applications:
  - Over all 1.57 million nets in 18 IBM circuits [ISPD 98]
    - More accurate than Batched 1-Steiner heuristic
    - Almost as fast as minimum spanning tree construction
Practical Impact of FLUTE

Nine companies either have or are planning to incorporate FLUTE into their tools:

- Intel, IBM, Magma, Calypto Design Systems, Atoptech, Dorado Design Automation, Lightspeed Semiconductor Corporation, Lizotech, Pulsic Limited

Thirteen academic EDA tools have incorporated FLUTE:

- Physical synthesis tools: SafeResynth
- Placement tools: Rooster, IPR
- Global routing tools: BoxRouter, FastRoute, DpRouter, FGR, Maizerouter, Archer, NTHU-Route, IGOR, HSR, Simple Router
FLUTE Publications

- Chris Chu, “FLUTE: Fast Lookup Table Based Wirelength Estimation Technique”, ICCAD 2004. (FLUTE 1.0)
- Chris Chu and Y.-C. Wong, “Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”, ISPD 2005. (FLUTE 2.0)
- Chris Chu and Y.-C. Wong, “FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”, TCAD 2008. (FLUTE 2.5)
Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets

- Experimental results
Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets
- Experimental results
A net is a set of $n$ pins

Degree of a net is the number of pins in it

Consider routing along Hanan grid

- Hanan proved an optimal RSMT always exists along the Hanan grid

Observation: An optimal RSMT can always be broken down into a set of horizontal edges and vertical edges

Define edge lengths $h_i$ and $v_i$: 

[Diagram showing horizontal and vertical edges with labels $h_i$ and $v_i$.]
Wirelength Vector (WV)

- **Observation:** WL can be written as a linear combination of edge lengths with positive integral coefficients.

\[
WL = h_1 + 2h_2 + h_3 + v_1 + v_2 + 2v_3 \\
WL = h_1 + h_2 + h_3 + v_1 + 2v_2 + 3v_3 \\
WL = h_1 + 2h_2 + h_3 + v_1 + v_2 + v_3
\]

\[(1, 2, 1, 1, 1, 2) \quad (1, 1, 1, 1, 2, 3) \quad (1, 2, 1, 1, 1, 1)\]

- WL can be expressed as a vector of the coefficients.
- Called **Wirelength Vector**
Potentially Optimal WV (POWV)

- To find optimal wirelength, can enumerate all WVs
- However, most WVs can never produce optimal WL
  - \((1, 2, 1, 1, 1, 2)\) is redundant as it always produces a larger WL than \((1, 2, 1, 1, 1, 1)\)

- Potentially Optimal Wirelength Vector (POWV) is a WV that *may* produce the optimal wirelength
# of POWVs is Very Small

- For any net,
  - # of possible routing solutions is huge
  - # of WVs is much less
  - # of POWVs is very small

- For example, only 2 POWVs for the net below:

  POWV
  (1,2,1,1,1,1)

  POWV
  (1,1,1,1,2,1)
Sharing of POWVs Among Nets

- To find optimal WL, we can pre-compute all POWVs and store them in a lookup table.
- However, there are infinite number of different nets.
- We try to group together nets that can share the same set of POWVs.
- For example, these two nets share the same set of POWVs:
Grouping by Position Sequence

- Define **position sequence** $s_1s_2...s_n$ to be the list of rank of pins in x-coordinate

  Position sequence $= 3142$

- **Lemma:** The set of all degree-$n$ nets can be divided into $n!$ groups according to the position sequence such that all nets in each group share the same set of POWVs
Steps of FLUTE for WL Estimation

- Given a net:
  1. Find the position sequence
  2. Get the POWVs from LUT
  3. Find the edge lengths
  4. Find WL for each POWV and return the best

**Position sequence:**

```
3 1 4 2
```

**POWVs:**

```
(1,2,1,1,1,1)  (1,1,1,1,2,1)
```

**Edge lengths:**

```
3 2 5
```

**WL calculation:**

```
HPWL + 2 = 22  HPWL + 6 = 26
```

Return
Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets

- Experimental results
POWVs Generation -- First Attempt

- For each small net degree and for each group (i.e., position sequence),
  - generate all routing topologies
  - find the corresponding WVs
  - prune away the redundant ones.

- Extremely expensive

- A better algorithm based on boundary compaction
Boundary Compaction Technique

Left Boundary Compaction

One possible routing topology

Left Boundary Expansion
Boundary compaction can be considered as a specific way to perform routing.

Different order in compacting the 4 boundaries will generate different routing topologies.

Some routing topologies and hence some WVVs may be missed.

**Theorem:** Boundary compaction can enumerate all POWVs for nets up to degree 6.

By including some extra topologies, we can enumerate all POWVs up to degree 9.
Statistics on POWV Table

- Table size for all nets up to degree 9 is 2.75MB

<table>
<thead>
<tr>
<th>Degree $n$</th>
<th># of groups $n!$</th>
<th># of POWVs in a group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>720</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>5040</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>40320</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>362880</td>
<td>1</td>
</tr>
</tbody>
</table>
Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets
- Experimental results
Minimum Wirelength Computation

- For a given net, we need to compute the WL corresponding to a set of POWVs
- Can compute each POWV independently
- However, most POWVs in a group are similar to one another
- Can speed up computation by exploring dependency among POWVs
- Example:

  \[ WL_1: (1,2,1,1, 1,2,3,1) \]  \[ WL_2: (1,2,2,1, 1,2,2,1) \]  \[ WL_3: (1,3,2,1, 1,1,2,1) \]

  \[ WL_1 = HPWL + h_2 + v_2 + v_3 + v_3 \]
  \[ WL_2 = WL_1 + h_3 - v_3 \]
  \[ WL_3 = WL_2 + h_2 - v_2 \]
MST-Based Approach

- The WL computation problem of a set of POWVs can be transformed into a MST problem.
- Cost of MST = # of Add/Sub to compute all POWVs.

- One node for each POWV.
- Edge weight = # of add/sub required to convert the WL of one WV to the other.
# of Add/Sub for MST-Based Approach

<table>
<thead>
<tr>
<th>Degree $n$</th>
<th>Average # of ADD/SUB per group</th>
<th>Average # of ADD/SUB per POWV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Independent</td>
<td>MST</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1.333</td>
<td>1.333</td>
</tr>
<tr>
<td>5</td>
<td>4.267</td>
<td>4.267</td>
</tr>
<tr>
<td>6</td>
<td>14.422</td>
<td>10.333</td>
</tr>
<tr>
<td>7</td>
<td>39.651</td>
<td>20.025</td>
</tr>
<tr>
<td>8</td>
<td>109.136</td>
<td>38.561</td>
</tr>
<tr>
<td>9</td>
<td>288.060</td>
<td>74.155</td>
</tr>
</tbody>
</table>
Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets
- Experimental results
High-Degree Nets by Net Breaking

- Lookup table is practical only for low-degree nets
- Have a user-defined parameter $D$
  - $D=9$ in our implementation

- For nets up to degree $D$, use lookup table
- For nets with degree > $D$, recursively break net until degree $\leq D$
  - Optimal net breaking algorithm
  - Net Breaking Heuristic #1
  - Net Breaking Heuristic #2
  - Net Breaking Heuristic #3
  - Net Breaking Heuristic #4
  - Accuracy control scheme
Optimal Net Breaking Algorithm

Condition:
Pins on opposite quadrants.

Theorem:
By combining the two optimal sub-trees, the Steiner tree constructed is optimal.
Net Breaking Heuristics

- Apply if condition for optimal net breaking is not satisfied
- A score for each direction and each pin
- Break in several ways which give the highest scores

\[
S_\gamma (r) = S_1 (r) - \alpha S_2 (r) - \beta S_3 (r) - \gamma S_4 (r)
\]
Net Breaking Heuristic #1

- A score for each direction and each pin
- Break in several ways which give the highest scores

\[ S_y(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r) \]
Net Breaking Heuristic #2

- A score for each direction and each pin
- Break in several ways which give the highest scores

\[ S_v(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r) \]

Subnet 1

Subnet 2

\[ S_2(r) \]

\[ \alpha = 0.3 \]
Net Breaking Heuristic #3

- A score for each direction and each pin
- Break in several ways which give the highest scores

\[ S_v(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r) \]

\[ \bar{v} = \text{Average vertical segment length} \]

\[ \bar{h} = \text{Average horizontal segment length} \]

\[ S_3(r) = n_x \cdot \bar{h} + n_y \cdot \bar{v} \]

\[ \beta = \frac{7.4}{n + 10} \]
Net Breaking Heuristic #4

- A score for each direction and each pin
- Break in several ways which give the highest scores

\[ S_y(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r) \]

\[ S_4(r) = HPWL_1 + HPWL_2 \]

\[ \gamma = \frac{4.8}{n - 1} \]
Accuracy Control Scheme

- Accuracy parameter $A$
- Break a net in $A$ ways with the highest scores
- Subnets are handled with accuracy $\max\{\lfloor A/2 \rfloor, 1\}$
- Runtime complexity $= O\left(A^{\log A+1/n \log n}\right)$
- Default $A=3$
Extension for RSMT Construction

- If degree $\leq D$, store 1 routing topology for each POWV
  
  **POVV**
  
  (1,2,1,1,1,1)

  **POVV**
  
  (1,1,1,1,2,1)

- If degree $> D$, Steiner trees of two sub-nets are combined
  
  - The local sub-tree around the merging pin can be refined by FLUTE

(a) Extra Steiner node

(b) pin r

(c) Refined subtree
Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets

- Experimental results
Experimental Setup

- Comparing five techniques:
  - **RMST** – Prim’s RMST algorithm
    - Prim [BSTJ 57]
  - **RST-T** – Refined Single Trunk Tree
    - Chen et al. [SLIP 02]
  - **SPAN** – Spanning graph based algorithm
    - Zhou [ISPD 03]
  - **BGA** – Batched Greedy Algorithm
    - Kahng et al. [ASPDAC 03]
  - **BI1S** – Batched Iterated 1-Steiner heuristic
    - Griffith et al. [TCAD 94]
  - **FLUTE** (version 2.5) with D=9 and A=3

- 18 IBM circuits in the ISPD98 benchmark suite
- Placement by FastPlace [ISPD 04]
- Optimal solutions by GeoSteiner 3.1 (Warme et al.)
### Benchmark Information

<table>
<thead>
<tr>
<th>Circuit</th>
<th># of nets</th>
<th>Ave. degree</th>
<th>Max. degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>14111</td>
<td>3.58</td>
<td>42</td>
</tr>
<tr>
<td>ibm02</td>
<td>19584</td>
<td>4.15</td>
<td>134</td>
</tr>
<tr>
<td>ibm03</td>
<td>27401</td>
<td>3.41</td>
<td>55</td>
</tr>
<tr>
<td>ibm04</td>
<td>31970</td>
<td>3.31</td>
<td>46</td>
</tr>
<tr>
<td>ibm05</td>
<td>28446</td>
<td>4.44</td>
<td>17</td>
</tr>
<tr>
<td>ibm06</td>
<td>34826</td>
<td>3.68</td>
<td>35</td>
</tr>
<tr>
<td>ibm07</td>
<td>48117</td>
<td>3.65</td>
<td>25</td>
</tr>
<tr>
<td>ibm08</td>
<td>50513</td>
<td>4.06</td>
<td>75</td>
</tr>
<tr>
<td>ibm09</td>
<td>60902</td>
<td>3.65</td>
<td>39</td>
</tr>
<tr>
<td>ibm10</td>
<td>75196</td>
<td>3.96</td>
<td>41</td>
</tr>
<tr>
<td>ibm11</td>
<td>81454</td>
<td>3.45</td>
<td>24</td>
</tr>
<tr>
<td>ibm12</td>
<td>77240</td>
<td>4.11</td>
<td>28</td>
</tr>
<tr>
<td>ibm13</td>
<td>99666</td>
<td>3.58</td>
<td>24</td>
</tr>
<tr>
<td>ibm14</td>
<td>152772</td>
<td>3.58</td>
<td>33</td>
</tr>
<tr>
<td>ibm15</td>
<td>186608</td>
<td>3.84</td>
<td>36</td>
</tr>
<tr>
<td>ibm16</td>
<td>190048</td>
<td>4.10</td>
<td>40</td>
</tr>
<tr>
<td>ibm17</td>
<td>189581</td>
<td>4.54</td>
<td>36</td>
</tr>
<tr>
<td>ibm18</td>
<td>201920</td>
<td>4.06</td>
<td>66</td>
</tr>
<tr>
<td><strong>All</strong></td>
<td><strong>1570355</strong></td>
<td><strong>3.92</strong></td>
<td><strong>134</strong></td>
</tr>
</tbody>
</table>
## Accuracy Comparison

<table>
<thead>
<tr>
<th>Circuit</th>
<th>RMST</th>
<th>RST-T</th>
<th>SPAN</th>
<th>BGA</th>
<th>BIIS</th>
<th>FLUTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>4.092</td>
<td>1.933</td>
<td>0.251</td>
<td>0.129</td>
<td>0.106</td>
<td>0.074</td>
</tr>
<tr>
<td>ibm02</td>
<td>5.849</td>
<td>3.780</td>
<td>0.331</td>
<td>0.143</td>
<td>0.115</td>
<td>0.209</td>
</tr>
<tr>
<td>ibm03</td>
<td>4.637</td>
<td>1.919</td>
<td>0.271</td>
<td>0.125</td>
<td>0.095</td>
<td>0.062</td>
</tr>
<tr>
<td>ibm04</td>
<td>4.048</td>
<td>1.255</td>
<td>0.203</td>
<td>0.084</td>
<td>0.060</td>
<td>0.051</td>
</tr>
<tr>
<td>ibm05</td>
<td>4.489</td>
<td>3.134</td>
<td>0.329</td>
<td>0.153</td>
<td>0.112</td>
<td>0.106</td>
</tr>
<tr>
<td>ibm06</td>
<td>5.964</td>
<td>2.822</td>
<td>0.381</td>
<td>0.182</td>
<td>0.134</td>
<td>0.084</td>
</tr>
<tr>
<td>ibm07</td>
<td>4.720</td>
<td>1.704</td>
<td>0.268</td>
<td>0.116</td>
<td>0.084</td>
<td>0.046</td>
</tr>
<tr>
<td>ibm08</td>
<td>4.784</td>
<td>4.445</td>
<td>0.328</td>
<td>0.162</td>
<td>0.123</td>
<td>0.261</td>
</tr>
<tr>
<td>ibm09</td>
<td>4.331</td>
<td>1.804</td>
<td>0.235</td>
<td>0.105</td>
<td>0.075</td>
<td>0.042</td>
</tr>
<tr>
<td>ibm10</td>
<td>4.104</td>
<td>1.790</td>
<td>0.252</td>
<td>0.104</td>
<td>0.080</td>
<td>0.051</td>
</tr>
<tr>
<td>ibm11</td>
<td>4.018</td>
<td>1.227</td>
<td>0.219</td>
<td>0.087</td>
<td>0.062</td>
<td>0.024</td>
</tr>
<tr>
<td>ibm12</td>
<td>3.783</td>
<td>1.908</td>
<td>0.248</td>
<td>0.106</td>
<td>0.077</td>
<td>0.054</td>
</tr>
<tr>
<td>ibm13</td>
<td>4.782</td>
<td>2.002</td>
<td>0.292</td>
<td>0.135</td>
<td>0.102</td>
<td>0.053</td>
</tr>
<tr>
<td>ibm14</td>
<td>3.908</td>
<td>1.540</td>
<td>0.221</td>
<td>0.095</td>
<td>0.068</td>
<td>0.040</td>
</tr>
<tr>
<td>ibm15</td>
<td>4.201</td>
<td>1.941</td>
<td>0.266</td>
<td>0.106</td>
<td>0.077</td>
<td>0.062</td>
</tr>
<tr>
<td>ibm16</td>
<td>4.231</td>
<td>2.421</td>
<td>0.279</td>
<td>0.124</td>
<td>0.090</td>
<td>0.068</td>
</tr>
<tr>
<td>ibm17</td>
<td>3.905</td>
<td>2.188</td>
<td>0.263</td>
<td>0.110</td>
<td>0.082</td>
<td>0.056</td>
</tr>
<tr>
<td>ibm18</td>
<td>4.432</td>
<td>3.353</td>
<td>0.300</td>
<td>0.134</td>
<td>0.100</td>
<td>0.147</td>
</tr>
<tr>
<td>All</td>
<td>4.232</td>
<td>2.261</td>
<td>0.269</td>
<td>0.117</td>
<td>0.086</td>
<td>0.075</td>
</tr>
</tbody>
</table>
## Runtime Comparison

All experiments are carried out on a 3.4-GHz Pentium 4 machine.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>RMST (s)</th>
<th>RST-T (s)</th>
<th>SPAN (s)</th>
<th>BGA (s)</th>
<th>BIS (s)</th>
<th>FLUTE (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>0.02</td>
<td>0.09</td>
<td>0.55</td>
<td>0.75</td>
<td>1.01</td>
<td>0.02</td>
</tr>
<tr>
<td>ibm02</td>
<td>0.02</td>
<td>0.14</td>
<td>1.05</td>
<td>1.50</td>
<td>4.32</td>
<td>0.03</td>
</tr>
<tr>
<td>ibm03</td>
<td>0.02</td>
<td>0.18</td>
<td>1.02</td>
<td>1.38</td>
<td>1.95</td>
<td>0.03</td>
</tr>
<tr>
<td>ibm04</td>
<td>0.04</td>
<td>0.20</td>
<td>1.07</td>
<td>1.44</td>
<td>2.24</td>
<td>0.02</td>
</tr>
<tr>
<td>ibm05</td>
<td>0.03</td>
<td>0.20</td>
<td>1.71</td>
<td>2.40</td>
<td>2.69</td>
<td>0.05</td>
</tr>
<tr>
<td>ibm06</td>
<td>0.03</td>
<td>0.23</td>
<td>1.45</td>
<td>1.95</td>
<td>2.53</td>
<td>0.04</td>
</tr>
<tr>
<td>ibm07</td>
<td>0.05</td>
<td>0.32</td>
<td>1.96</td>
<td>2.59</td>
<td>3.26</td>
<td>0.04</td>
</tr>
<tr>
<td>ibm08</td>
<td>0.06</td>
<td>0.35</td>
<td>2.63</td>
<td>3.74</td>
<td>6.60</td>
<td>0.09</td>
</tr>
<tr>
<td>ibm09</td>
<td>0.07</td>
<td>0.40</td>
<td>2.42</td>
<td>3.19</td>
<td>4.13</td>
<td>0.06</td>
</tr>
<tr>
<td>ibm10</td>
<td>0.08</td>
<td>0.53</td>
<td>3.59</td>
<td>4.77</td>
<td>5.85</td>
<td>0.09</td>
</tr>
<tr>
<td>ibm11</td>
<td>0.06</td>
<td>0.53</td>
<td>2.87</td>
<td>3.76</td>
<td>5.16</td>
<td>0.05</td>
</tr>
<tr>
<td>ibm12</td>
<td>0.10</td>
<td>0.54</td>
<td>3.94</td>
<td>5.33</td>
<td>6.25</td>
<td>0.10</td>
</tr>
<tr>
<td>ibm13</td>
<td>0.10</td>
<td>0.66</td>
<td>3.89</td>
<td>5.18</td>
<td>6.68</td>
<td>0.09</td>
</tr>
<tr>
<td>ibm14</td>
<td>0.15</td>
<td>1.02</td>
<td>5.91</td>
<td>7.84</td>
<td>10.11</td>
<td>0.14</td>
</tr>
<tr>
<td>ibm15</td>
<td>0.21</td>
<td>1.27</td>
<td>8.18</td>
<td>10.86</td>
<td>13.96</td>
<td>0.22</td>
</tr>
<tr>
<td>ibm16</td>
<td>0.23</td>
<td>1.33</td>
<td>9.33</td>
<td>12.47</td>
<td>14.75</td>
<td>0.26</td>
</tr>
<tr>
<td>ibm17</td>
<td>0.28</td>
<td>1.39</td>
<td>11.06</td>
<td>15.06</td>
<td>16.63</td>
<td>0.31</td>
</tr>
<tr>
<td>ibm18</td>
<td>0.26</td>
<td>1.40</td>
<td>9.81</td>
<td>13.28</td>
<td>17.82</td>
<td>0.30</td>
</tr>
<tr>
<td>All</td>
<td>0.93</td>
<td>5.56</td>
<td>37.34</td>
<td>50.25</td>
<td>64.92</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Normalized
Accuracy vs. Runtime Tradeoff

![Graph showing the tradeoff between error and runtime for different methods (RMST, RSTT, SPAN, BGA, BI1S). The graph plots error (%) on the y-axis and runtime (s) on the x-axis.]
Effect of Accuracy Control Parameter

RMST Runtime
(Error 4.232%)

![Graph showing the effect of accuracy control parameter on RMST runtime and error. The graph plots error (%) on the y-axis and runtime (s) on the x-axis. Different values of A (1, 2, 3, 4, 5, 6, 7) are shown, with varying degrees of error and runtime. The point A=3 (default) is highlighted, indicating a balance between error and runtime.]
Breakdown According to Net Degree

- All 1.57 million nets in 18 circuits
  - Average degree = 3.92
  - 8.13% with degree ≥ 10 (but 26.2% of WL)
  - 0.077% with degree ≥ 30
  - 0.005% with degree ≥ 60

<table>
<thead>
<tr>
<th>Degree</th>
<th>Net breakdown</th>
<th>Wirelength error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>WL</td>
</tr>
<tr>
<td>2</td>
<td>54.92%</td>
<td>27.98%</td>
</tr>
<tr>
<td>3</td>
<td>14.40%</td>
<td>10.26%</td>
</tr>
<tr>
<td>4</td>
<td>7.68%</td>
<td>7.84%</td>
</tr>
<tr>
<td>5</td>
<td>5.61%</td>
<td>8.18%</td>
</tr>
<tr>
<td>6</td>
<td>3.20%</td>
<td>5.65%</td>
</tr>
<tr>
<td>7</td>
<td>2.28%</td>
<td>4.82%</td>
</tr>
<tr>
<td>8</td>
<td>1.98%</td>
<td>4.61%</td>
</tr>
<tr>
<td>9</td>
<td>1.81%</td>
<td>4.46%</td>
</tr>
<tr>
<td>10–17</td>
<td>6.98%</td>
<td>21.72%</td>
</tr>
<tr>
<td>≥18</td>
<td>1.15%</td>
<td>4.48%</td>
</tr>
</tbody>
</table>
Improvement For High-Degree Nets

- For high-degree nets (with tens of pins or more), net breaking according to rectilinear minimum spanning tree
  - Complicated merging techniques to achieve extraordinary accuracy
Conclusion

- FLUTE:
  - Rectilinear Steiner Minimal Tree algorithm
  - Post-placement pre-routing wirelength estimation

- Very suitable for VLSI applications:
  - Optimal and extremely fast up to degree 9
  - Very accurate and fast up to degree 100
  - Nice tradeoff between accuracy and runtime

- Key ideas:
  - Pre-computed POWVs by boundary compaction
  - Store the POWVs and corresponding RSMTs in LUT
  - MST-based approach to speed up WL computation
  - Net-breaking technique to handle large nets
Extension and Future Works

Extension:
- Delay-driven Steiner tree construction

Future Works:
- Extend FLUTE for RSMT construction with obstacles
- Design LUT-based practical algorithms for other NP-complete problems

Source code available in GSRC Bookshelf:
http://home.eng.iastate.edu/~cnchu/flute.html
Thank You