Steiner Routing

ECE6133
Physical Design Automation of VLSI Systems

Prof. Sung Kyu Lim
School of Electrical and Computer Engineering
Georgia Institute of Technology
ARM A53 Placement

arm
CORTEX®-A53

CoreSight™ multicore debug and trace

Armv8-A
32b/64b CPU

NEON™ SIMD engine
with crypto ext.

Floating point unit

8-64k l-cache w/parity

8-64k D-cache w/ECC

ACP

SCU

L2 w/ECC (128kB–2MB)

Configurable AMBA® 4 ACE or AMBA 5 CHI coherent bus interface
### TSMC 28nm BEOL Spec

<table>
<thead>
<tr>
<th></th>
<th>Width (um)</th>
<th>Pitch (um)</th>
<th>Dir.</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.05</td>
<td>0.135</td>
<td>V</td>
</tr>
<tr>
<td>M2</td>
<td>0.05</td>
<td>0.100</td>
<td>H</td>
</tr>
<tr>
<td>M3</td>
<td>0.05</td>
<td>0.100</td>
<td>V</td>
</tr>
<tr>
<td>M4</td>
<td>0.05</td>
<td>0.100</td>
<td>H</td>
</tr>
<tr>
<td>M5</td>
<td>0.05</td>
<td>0.100</td>
<td>V</td>
</tr>
<tr>
<td>M6</td>
<td>0.05</td>
<td>0.100</td>
<td>H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>R (ohm/um)</th>
<th>C (fF/um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>7.24</td>
<td>0.172</td>
</tr>
<tr>
<td>M2</td>
<td>9.05</td>
<td>0.175</td>
</tr>
<tr>
<td>M3</td>
<td>9.06</td>
<td>0.181</td>
</tr>
<tr>
<td>M4</td>
<td>9.05</td>
<td>0.177</td>
</tr>
<tr>
<td>M5</td>
<td>9.06</td>
<td>0.180</td>
</tr>
<tr>
<td>M6</td>
<td>9.05</td>
<td>0.177</td>
</tr>
</tbody>
</table>
Full-Chip Routing
Full-Chip Routing

M4

M5

M6
M1 Layer (Mostly Intra-Cell Routing)

yellow: signal
M2 Layer

yellow: signal
magenta: clock, red: power/ground
M3 Layer

yellow: signal
magenta: clock
yellow: signal
magenta: clock
yellow: signal
magenta: clock, red: power/ground
M7 and M8

magenta: power/ground
Routing

- Generates a "loose" route for each net.
- Assigns a list of routing regions to each net without specifying the actual layout of wires.

Global routing
- Finds the actual geometric layout of each net within the assigned routing regions.

Detailed routing

Compaction
Routing Constraints

- 100% routing completion + area minimization, under a set of constraints:
  - Placement constraint: usually based on fixed placement
  - Number of routing layers
  - Geometrical constraints: must satisfy design rules
  - Timing constraints (performance-driven routing): must satisfy delay constraints
  - Crosstalk?
  - Process variations?

*Two-layer routing*
Graph Models for Global Routing: Grid Graph

- Each cell is represented by a vertex.
- Two vertices are joined by an edge if the corresponding cells are adjacent to each other.
- The occupied cells are represented as filled circles, whereas the others are as clear circles.
Global-Routing Problem

• Given a netlist \( N = \{N_1, N_2, \ldots, N_n\} \), a routing graph \( G = (V, E) \), find a Steiner tree \( T_i \) for each net \( N_i \), \( 1 \leq i \leq n \), such that \( U(e_j) \leq c(e_j) \), \( \forall e_j \in E \) and \( \sum_{i=1}^{n} L(T_i) \) is minimized, where
  
  - \( c(e_j) \): capacity of edge \( e_j \);
  - \( x_{ij} = 1 \) if \( e_j \) is in \( T_i \); \( x_{ij} = 0 \) otherwise;
  - \( U(e_j) = \sum_{i=1}^{n} x_{ij} \): \# of wires that pass through the channel corresponding to edge \( e_j \);
  - \( L(T_i) \): total wirelength of Steiner tree \( T_i \).

• For high-performance, the maximum wirelength \( (\max_{i=1}^{n} L(T_i)) \) is minimized (or the longest path between two points in \( T_i \) is minimized).
**Classification of Global-Routing Algorithm**

- **Sequential approach:** Assigns priority to nets; routes one net at a time based on its priority (net ordering?).
- **Concurrent approach:** All nets are considered at the same time (complexity?)
Spanning Tree

Problem Formulation:
Given a graph $G = (V, E)$, select a subset $V' \subseteq V$, such that $V'$ has property $\mathcal{P}$.

Minimum Spanning Tree

Problem Formulation:
Given an edge-weighted graph $G = (V, E)$, select a subset of edges $E' \subseteq E$ such that $E'$ induces a tree and the total cost of edges $\sum_{e_i \in E'} wt(e_i)$, is minimum over all such trees, where $wt(e_i)$ is the cost or weight of the edge $e_i$.

– Used in routing applications.
Steiner Trees

1. Problem formulation:

Given an edge weighted graph \( G = (V, E) \) and a subset \( D \subseteq V \), select a subset \( V' \subseteq V \), such that \( D \subseteq V' \) and \( V' \) induces a tree of minimum cost over all such trees.

The set \( D \) is referred to as the set of demand points and the set \( V' - D \) is referred to as Steiner points.

- Used in the global routing of multi-terminal nets.
Min Spanning Trees vs. Steiner Trees

- Both problems try to “span” nodes in the given graph
  - Goal is to minimize the total edge weight
  - MST: span all nodes
  - Steiner tree: span only a designated subset of nodes. We can use “extra” nodes (= steiner nodes) if they help.

![Input Graph](image1)

![Minimum Spanning Tree](image2)

minimum spanning tree
total cost = 10

![Steiner Tree](image3)

steiner tree for \{a, c, e\}
steiner point = \{d\}
total cost = 9
Underlying Grid Graph

The underlying grid graph is defined by the intersections of the horizontal and vertical lines drawn through the demand points.

Hanan's Thm (69'): There exists an optimal RST with all Steiner points (set S) chosen from the intersection points of horizontal and vertical lines drawn from points of D.
Different Steiner trees constructed from a MST

Hwang's Thm (76'):
The ratio of the cost of a rectilinear MST to that of an optimal RST is no greater than 3/2.
Steiner Routing: 3D vs. 2D

3D Steiner Routing

2D Steiner Routing + Layer Assignment
The 1-Steiner Problem

Definition

We denote the minimum spanning tree over a point set $P$ by $MST(P)$, and use $c(MST(P))$ to denote the cost of the MST on point set $P$. Given a point set $P = \{p_1, \cdots, p_n\}$, a 1-Steiner point is any point $x$ such that $c(MST(P \cup \{x\}))$ is minimized, with $c(MST(P \cup \{x\})) < c(MST(P))$. A 1-Steiner tree is the minimum spanning tree over $P \cup \{x\}$. 
Why 1-Steiner Insertion?

- Can Reduce Wirelength

Fig. 3. Execution of iterated 1-Steiner on a four-point example.
1-Steiner by Kahng/Robins

- Iterative 1-Steiner Insertion Algorithm
  - Keep adding 1-Steiner point one-by-one until no more gain
  
  By the result of Hanan, we can find a 1-Steiner point by constructing a new MST on \( n + 1 \) points for each element in the Steiner candidate set, then picking the candidate which results in the shortest MST.

- Naïve implementation: \( O(n^2 \times n \log n \times n) \)
- Sophisticated implementation: \( O(n^3) \)
1-Steiner Routing by Kahng/Robins

- Perform 1-Steiner Routing by Kahng/Robins
  - Need an initial MST: wirelength is 20
  - 16 locations for Steiner points
First 1-Steiner Point Insertion

- There are six 1-Steiner points
  - Two best solutions: we choose (c) randomly

![Diagram showing before insertion](image)

(a) WL=19  (b) WL=19  (c) WL=18
First 1-Steiner Point Insertion (cont)

before insertion

(d) WL=18
(e) WL=19
(f) WL=19
Second 1-Steiner Point Insertion

- Need to break tie again
  - Note that (a) and (b) do not contain any more 1-Steiner point: so we choose (c)
Third 1-Steiner Point Insertion

- Tree completed: all edges are rectilinearized
  - Overall wirelength reduction $= 20 - 16 = 4$
Sample Kahng/Robins Routing (1/3)

- 5 points in 10x10 grid
  - 2 Steiner points used

MST (WL = 21)  
final tree (WL = 18)
Sample Kahng/Robins Routing (2/3)

- 50 points in 30x30 grid
  - 20 Steiner points used

MST (WL = 183)  final tree (WL = 163)
Sample Kahng/Robins Routing (3/3)

- 100 points in 30x30 grid
  - 22 Steiner points used, it took 15ms to route

MST (WL = 242)  \hspace{1cm}  \text{final tree (WL = 220)}
Kahng/Robins Speedup Techniques

• **Random variant**
  – Instead of choosing the best gain Steiner point in each iteration, just pick the first one found.
  – Time spent on each step is less, but more Steiner points need to be added.

• **Prune out bad candidates**
  – After the first iteration, the Hanan grid points that gave no gain were removed.
  – This improved practical time complexity.

• **Any other thoughts?**
Interesting Observation

Our edge-based algorithm is based on connecting a node to the nearest point on the rectangular layout of an edge in the tree and removing the longest edge in the loop thus formed.
Gain Computation

- Things to do
  1) Add node $p$
  2) Remove edge $e_1$
  3) Remove edge $e_2$
  4) Add edge connecting $p$ to $p_1$
  5) Add edge connecting $p$ to $p_2$
  6) Add edge connecting $p$ to $p_3$.

- Thus, the gain is

$$gain = \text{length}(e_2) - \text{length}(p, p_1)$$
Overall Algorithm

- Multi-pass Heuristic
  - Entire algorithm can be repeated

Algorithm Edge-based-Steiner()

Begin
1. Compute the rectilinear minimum spanning tree of the set of nodes
2. Compute all possible <node, edge> pairs that give positive gain
3. Sort all the pairs in descending order of gain
4. While (there are pairs with positive gain) do
   If (the two edges to be replaced exist in the tree) then
     Replace the pair of edges with three new edges and a new node.
   End-if
End-while
End
1-Steiner Routing by Borah/Owens/Irwin

- Perform a single pass of Borah/Owens/Irwin
  - Initial MST has 5 edges with wirelength of 20
  - Need to compute the max-gain (node, edge) pair for each edge in this MST
Best Pair for \((a, c)\)

We first let \(p_1 = b\) and \(e_1 = (a, c)\). Next, we compute the shortest Manhattan distance between \(p_1\) and a “rectilinear layout” of \(e_1\), which is 2 in this case. The node \(p\) is the nearest point on this rectilinear layout of \(e_1\) to \(p_1\). Next, we look for \(e_2\), the longest edge on \(p_1\)-to-\(a\) path, which is \(e_2 = (b, c)\). Thus,

\[
gain\{b, (a, c)\} = length(e_2) - length(p, p_1) = 4 - 2 = 2
\]
Best Pair for $(b,c)$

- Three nodes can pair up with $(b,c)$

\[
\begin{align*}
gain\{a, (b, c)\} &= length(a, c) - length(p, a) = 4 - 2 = 2 \\
gain\{d, (b, c)\} &= length(b, d) - length(p, d) = 5 - 4 = 1 \\
gain\{e, (b, c)\} &= length(c, e) - length(p, e) = 4 - 3 = 1
\end{align*}
\]

\[l(a,c) - l(p,a) = 4 - 2\]
Best Pair for \((b,c)\) (cont)

- All three pairs have the same gain
  - Break ties randomly

\[
l(b,d) - l(p,d) = 5 - 4
\]

\[
l(c,e) - l(p,e) = 4 - 3
\]
Best Pair for \((b,d)\)

- Two nodes can pair up with \((b,d)\)
  - both pairs have the same gain

\[
l(b,c) - l(p,c) = 4 - 3
\]

\[
l(b,c) - l(p,e) = 4 - 3
\]
Best Pair for \((c,e)\)

- Three nodes can pair up with \((c,e)\)

\[
\begin{align*}
l(b,c) - l(p,b) &= 4 - 3 \\
l(b,d) - l(p,d) &= 5 - 4
\end{align*}
\]
Best Pair for \((c, e)\) (cont)

\[ l(e, f) - l(p, f) = 3 - 2 \]
Best Pair for \((e,f)\)

- Can merge with \(c\) only

\[
l(c,e) - l(p,c) = 4 - 3
\]
Summary

- Max-gain pair table
  - Sort based on gain value

<table>
<thead>
<tr>
<th>pair</th>
<th>gain</th>
<th>$e_1$</th>
<th>$e_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${b, (a, c)}$</td>
<td>2</td>
<td>$(a, c)$</td>
<td>$(b, c)$</td>
</tr>
<tr>
<td>${a, (b, c)}$</td>
<td>2</td>
<td>$(b, c)$</td>
<td>$(a, c)$</td>
</tr>
<tr>
<td>${c, (b, d)}$</td>
<td>1</td>
<td>$(b, d)$</td>
<td>$(b, c)$</td>
</tr>
<tr>
<td>${b, (c, e)}$</td>
<td>1</td>
<td>$(c, e)$</td>
<td>$(b, c)$</td>
</tr>
<tr>
<td>${c, (e, f)}$</td>
<td>1</td>
<td>$(e, f)$</td>
<td>$(c, e)$</td>
</tr>
</tbody>
</table>
First 1-Steiner Point Insertion

- Choose \{b, (a,c)\} (max-gain pair)
  - Mark \(e_1 = (a,c), e_2 = (b,c)\)
  - Skip \{a, (b,c)\}, \{c, (b,d)\}, \{b, (c,e)\} since their \(e_1/e_2\) are already marked
  - Wirelength reduces from 20 to 18

<table>
<thead>
<tr>
<th>pair</th>
<th>gain</th>
<th>(e_1)</th>
<th>(e_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{b, (a, c)}</td>
<td>2</td>
<td>(a, c)</td>
<td>(b, c)</td>
</tr>
<tr>
<td>{a, (b, c)}</td>
<td>2</td>
<td>(b, c)</td>
<td>(a, c)</td>
</tr>
<tr>
<td>{c, (b, d)}</td>
<td>1</td>
<td>(b, d)</td>
<td>(b, c)</td>
</tr>
<tr>
<td>{b, (c, e)}</td>
<td>1</td>
<td>(c, e)</td>
<td>(b, c)</td>
</tr>
<tr>
<td>{c, (e, f)}</td>
<td>1</td>
<td>(e, f)</td>
<td>(c, e)</td>
</tr>
</tbody>
</table>
Second 1-Steiner Point Insertion

- Choose \{c, (e,f)\} (last one remaining)
  - Wirelength reduces from 18 to 17

<table>
<thead>
<tr>
<th>pair</th>
<th>gain</th>
<th>(e_1)</th>
<th>(e_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{b, (a, c)}</td>
<td>2</td>
<td>(a, c)</td>
<td>(b, c)</td>
</tr>
<tr>
<td>{a, (b, c)}</td>
<td>2</td>
<td>(b, c)</td>
<td>(a, c)</td>
</tr>
<tr>
<td>{c, (b, d)}</td>
<td>1</td>
<td>(b, d)</td>
<td>(b, c)</td>
</tr>
<tr>
<td>{b, (c, e)}</td>
<td>1</td>
<td>(c, e)</td>
<td>(b, c)</td>
</tr>
<tr>
<td>{c, (e, f)}</td>
<td>1</td>
<td>(e, f)</td>
<td>(c, e)</td>
</tr>
</tbody>
</table>
Sample Borah Routing

- 100 points in 30x30 grid
  - 22 Steiner points used, it took 59ms to route

MST (WL = 242)    final tree (WL = 219)
Comparison

- Kahng/Robins vs Borah/Owens/Irwin
  - Kahng/Robins tends to give better results
  - Borah/Owens/Irwin runs much faster: $O(n^4 \log n)$ vs $O(n^2)$
Bounded Radius Routing

- Why Radius?
  - Longest source-sink path length among all sinks
  - Smaller path resistance: better performance

- Both Radius and Cost?
  - Cost = wirelength
  - Radius (= R) and wirelength (= C) are both important for RC-delay reduction

- Bounded PRIM vs Bounded Radius/Cost
Radius vs Wirelength

Fig. 1. An example where the cost of a shortest path tree (right) is $\Omega(|N|)$ times larger than the cost of a minimum spanning tree (left).

Fig. 2. An example in the Manhattan plane of how increasing the value of $\epsilon$ may result in decreased tree cost, but increased radius $r(T)$: (a) $\epsilon = 0$, cost($T$) = 17, $n(T)$ = 6; (b) $\epsilon = 1$, cost($T$) = 15, $n(T)$ = 10; (c) $\epsilon = \infty$, cost($T$) = 14, $r(T)$ = 14.
BPRIM Under $\varepsilon = \infty$

Radius bound = $\infty$
= regular PRIM
BPRIM Under $\varepsilon = \infty$ (cont)
Bounded PRIM Algorithm

Variation of PRIM's MST algorithm

\[ T = (V', E') = (\{s\}, \emptyset) \]

while \(|V'| < |N|\)

Select two terminals \(x \in V'\) and \(y \in N - V'\) minimizing \(\text{dist}(x, y)\)

if \(\text{dist}_T(s, x) + \text{dist}(x, y) \leq (1 + \epsilon) \cdot R\) then \(x' = x\)

else find the first terminal \(x'\) along the path in \(T\) from \(x\) to \(s\)

such that \(\text{dist}_T(s, x') + \text{dist}(x', y) \leq R\)

\(V' = V' \cup \{x'\}\)

\(E' = E' \cup \{(x', y)\}\)

Fig. 4. Algorithm BPRIM: computing a bounded-radius spanning tree, \(T\), for a given set of terminals, \(N\), with source, \(s \in N\), and radius, \(R\), using parameter \(\epsilon\).
Why Tighter Radius?

- BPRIM uses tighter radius bound during backtracing
  - $R$ instead of $(1+\epsilon)R$

Note that in backtracing we could choose $x'$ such that $\text{dist}_T(s, x') + \text{dist}(x', y) \leq (1 + \epsilon) \cdot R$. However, our choice of appropriate edges leads to fewer backtracing operations, while guaranteeing that backtracing is still always possible. In other words, we intentionally introduce some "slack" at $y$ so that terminals within an $\epsilon R$ neighborhood of $y$ will not cause additional backtracing. Limiting the amount of backtracing in this way will keep the cost of the resulting tree close to that of the minimum spanning tree.
Bounded PRIM Algorithm

- Comparison (e = 0, 0.5, infinity)
  - Radius bound/value increase
  - Wirelength decreases

![Diagram](a) ![Diagram](b) ![Diagram](c)
Bounded Radius Routing

- Perform bounded PRIM algorithm
  - Under $\epsilon = 0$, $\epsilon = 0.5$, and $\epsilon = \infty$
  - Compare radius and wirelength
  - Radius = 12 for this net
BPRIM Under $\varepsilon = 0$ (cont)
BPRIM Under $\varepsilon = 0$ (cont)
BPRIM Under $\varepsilon = 0.5$ (cont)
BPRIM Under $\varepsilon = 0.5$ (cont)
Comparison

- As the bound increases ($12 \rightarrow 18 \rightarrow \infty$)
  - Radius value increases ($12 \rightarrow 17 \rightarrow 22$)
  - Wirelength decreases ($56 \rightarrow 49 \rightarrow 36$)