Algorithms for Global Routing

Objectives:
- Total WL minimization
- Congestion minimization
- Performance constraints/Timing minimization

Routing Topology:
- Channel Connectivity graph

Figure 6.12 Channel connectivity graph. (a) A building-block layout. (b) Corresponding channel connectivity graph.

- Vertices are channels & adjacent channels have edges bet. corresponding vertices
Bottleneck graph.

- Channel defined by the min. 7 dimension of two adjacent block edges.

- Vertices or switchboxes defined at channel intersections.

- The corresponding "bottleneck graph" (a type of channel graph) has nodes as the switchboxes and edges for channels between switchboxes.

- Edge weights can be based on:
  (i) channel length
  (ii) channel occupancy/capacity
  (updated after each global net route if needed)

Figure 6.13 Bottleneck graph. (a) A building-block layout. (b) Corresponding bottleneck graph.
Figure 3.6. Grid graph.
(a) A two-dimensional grid. (b) Corresponding grid graph where each
grid cell (a x 2 grid cell) is modeled by a vertex.
Special Case: Standard-cell Layout

- No prescribed vertical channels
- Can determine a grid of vertical spacing = horizontal spacing (hor. lines go through the middle of cell rows)

Figure 9.17 The partitioning of the standard-cell layout area by a grid for the sake of global routing (a) and a rectilinear Steiner tree embedded in this grid (b).

- Grid lines: m x 0, 1, ..., m
- m x 0, 1, ..., n

- Local vertical density \( d_v(i,j) \) = \# of wires crossing the vertical grid & line \( j \) between hor. lines \( i-1 \) & \( i \) \((1 \leq i \leq m, 1 \leq j \leq n-1)\)

- Local hor. density \( d_h(i,j) \) = \# of wires crossing hor. grid line \( i \) between vertical grid lines \( j-1 \) & \( j \) \((1 \leq i \leq m-1, 1 \leq j \leq n)\)
- Channel density \( D_v(i) \) for grid line \( i \) \((1 \leq i \leq n)\) is:

\[
D_v(i) = \max_{j=1}^{n-1} d_v((i, j))
\]

- Congestion minimization goal of global routing is equivalent to:

\[
\text{Minimize total channel density given by } \sum_{i=1}^{m} D_v(i)
\]

- Satisfy constraints \( d_x((i, j)) \leq M_{ij}, (1 \leq i \leq m-1, 1 \leq j \leq n) \)

where \( M_{ij} = \max_{k} \text{ feed through cell width (of feed through wires) in cell row } i \) between vertical grid lines \( j-1 \) \& \( j \)

- \( M_{ij}'s \) may be apportioned depending on the difference of cell row \( i \) for the max cell row
Global Routing Approaches

- Flat
  - Sequential Steiner tree constructions, followed after each net route by update of edge/vertex weights to reflect current density

- Hierarchical
  - Start w/ coarse 2x2 grid. Place all pins at center of each grid cell.
  - Perform S-Tree routing of external nets to evenly distribute the zones.

- Solve routing problem of internal vs. external nets of each cell by subdividing it further into a 2x2 grid, etc.

- Can stop either when the first grid is reached or when sufficiently detailed net distribution is achieved for a detailed router to take over
Approximate Steiner Tree Construction Algorithms

Seen before:
1) Multi-terminal Tree or Multi-terminal Dijkstra
2) Minimum Rect. Spanning Tree (MRST)

- Since \( l(MRST) \leq 1.5 l(MRST) \)
- Min Rect. Steiner Tree

algorithms for that start w/ an MRST &
improve it have been developed.

We will discuss 2 such algorithms here.

The algorithms are best work w/ a
bottle-neck graph for nano-cell layout
(here the tree is not longer necessarily rectangular)
or the grid graph developed earlier for a
standard cell layout.
Algorithm 1: Flipping & Merging

- In the MRSOT, identify all connections between two points \( u, v \) such that there are parallel wires from \( u \) to the other point \( v \), hence breaking the segment of the wire.

- If so flip the segment of the connection (i.e., use the alternate \( L \) connection which looks like \( T \), and vice versa) and merge of the parallel wires.

- Repeat the process until no further improvement.

- A local search (iterative improvement process)
Algorithm 2: Steiner Heuristic

- Input: Set \( P \) of \( n \) points \( p_1, \ldots, p_n \) in a 2-D plane.

- Output: A minimum-length tree that interconnects all points of \( P \) making use of other non-\( P \) points in the plane (i.e., these non-\( P \) points have degree \( \geq 3 \), not just points in interconnects between \( P \) points on the tree) if these points can reduce the tree length. Such non-\( P \) points (of degree \( \geq 3 \)) are called Steiner points and denoted by \( S \).

[Hanonn, SIAM J. Appl. Math. '65]

[Candidates for Steiner's points: Optimal MRSOT can be embedded in a grid composed of]
only those grid lines that go through points of P. Points on this grid are called Hanan points.

(Note: An ST can have non-Hanan points: the MST is not unique.)

Figure 9.20 A set of points P for which the Steiner-tree should be constructed (a) and its Hanan points (b).

- Thus there are potentially $O(n^2)$ Steiner points.

- It has, however, been shown that there are at most $n-2$ Steiner points for a point set P, based on the fact that each Steiner point is connected to at least $n-1$ edges in the MST.
The 1-Steiner Heuristic

**Initial MRS\textsubscript{pT} of point set P**

Find min-length \{MRS\textsubscript{T}(P∪\{s\}) : S = set of s

by incremental update of MRS\textsubscript{T}(P)

(1-Steiner tree heuristic)

Let \(S_{min}\) be that \(S\) that is the corresponding to the min-length \(MRS\textsubscript{T}(P∪\{s\})\).

\textbf{P = P∪\{S_{min}\}}

\textbf{S = S - \{S_{min}\}}

\textbf{Length[MRS\textsubscript{T}(P∪\{S_{min}\})] < Length[MRS\textsubscript{T}(P)] ?}

MRS\textsubscript{T}(P) is approx. Steiner-tree.
(set of struct vertex, set of struct edge) steiner(set of struct vertex P)
{
    set of struct vertex T;
    set of struct edge E, F;
    int gain;

    E ← prim(P);
    (T, F, gain) ← 1-steiner(P, E);
    while (gain > 0) |
    P ← T;
    E ← F;
    (T, F, gain) ← 1-steiner(P, E);
    return (P, E);
}

Figure 9.19 The iterated 1-Steiner heuristic for Steiner-tree construction.

(set of struct vertex, set of struct edge, int)
1-steiner(set of struct vertex V, set of struct edge E)
{
    set of struct vertex W;
    set of struct edge F;
    struct vertex maxpoint;
    int gain, maxgain;

    maxgain ← 0;
    for each s ∈ "Hannan points of V" |
    (W, F, gain) ← spanning_update(V, E, s);
    if (gain > maxgain) |
        maxgain ← gain;
        maxpoint ← s;
    |
    if (maxgain > 0) |
    (W, F, gain) ← spanning_update(V, E, s);
    return (W, F, maxgain);
    |
    else return (V, E, 0);
}

Figure 9.21 The pseudo-code of the 1-Steiner algorithm.

(set of struct vertex, set of struct edge, int)
spanning_update(set of struct vertex V, set of struct edge E, struct vertex s)
{
    int delta;
    struct vertex u, v, w;

    delta ← 0;
    V ← V U {s};
    for each d ∈ {north, east, south, west} |
        u ← closest_point(V, s, d);
        delta ← delta - distance(s, u);
        E ← E U {(s, u)};
        if (cycle(V, E)) |
            (v, w) ← largest_cycle_segment(V, E);
            E ← E \ {(v, w)};
            delta ← delta + distance(v, w);
        |
    return (V, E, delta);
}

Figure 9.23 The different steps in the incremental computation of a spanning tree.

Figure 9.22 The function that incrementally computes a spanning tree when a new point is added to the original point set.
Figure 9.23  The different steps in the incremental computation of a spanning tree.

Figure 9.24  The solution found by the iterated 1-Steiner heuristic for the problem instance.
Time Complexity:

- Spanning Update: $\Theta(n)$

- 1 - Steiner Tree $\rightarrow \Theta(n)$ calls to Spanning Update (for $n-2$ Steiner points)
  $\Rightarrow \Theta(n^2)$

- Steiner (approx) $\rightarrow \Theta(n)$ calls to 1 - Steiner
  $\Rightarrow \Theta(n^3)$
Local Transformations of S-Trees for Congestion Reduction

Figure 9.25  Local transformations for Steiner trees.