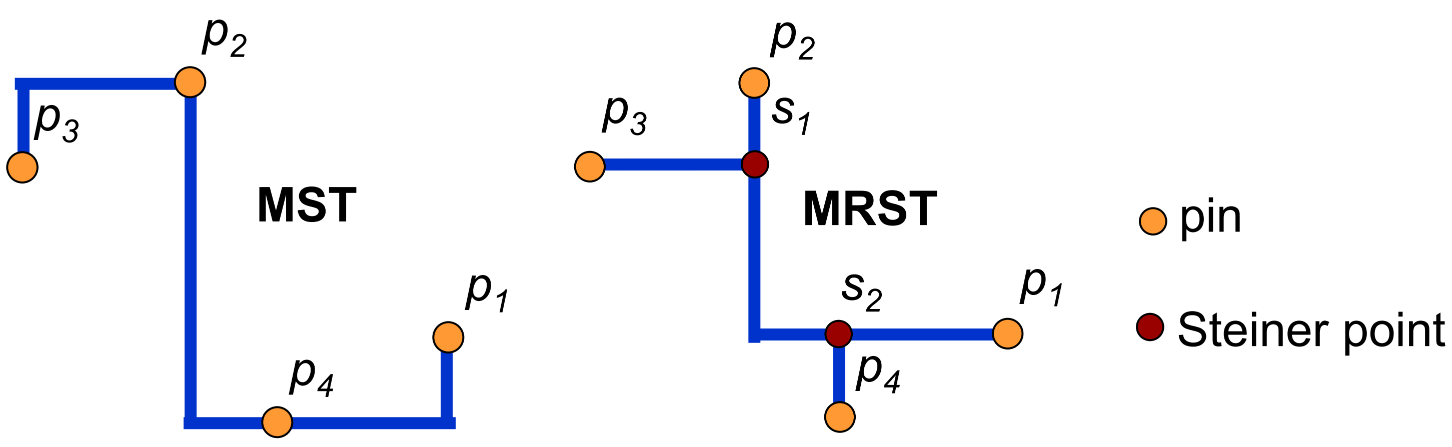
Routing Tree

- If all nets are two-pin ones, we can apply a general-purpose routing algorithm to handle the problem, such as maze, line-search, and A*-search routing.
- For three or more multi-pin nets, one approach is to *decompose* each net into a set of two-pin connections, and then routes the connections one-by-one.

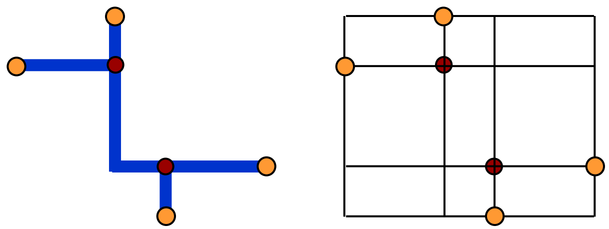
The Routing-Tree Problem

- **Problem:** Given a set of pins of a net, interconnect the pins by a "routing tree."
- **Minimum Spanning Tree (MST):** a minimum-length tree of edges connecting all the pins
- **Minimum Rectilinear Steiner Tree (MRST) Problem:** Given n points in the plane, find a minimum-length tree of rectilinear edges which connects the points. (Very useful in routing of VLSI circuits, but NP-hard)
- $MRST(P) = MST(P \cup S)$, where P and S are the sets of original points and Steiner points, respectively.



Theoretic Results for the RSMT Problem

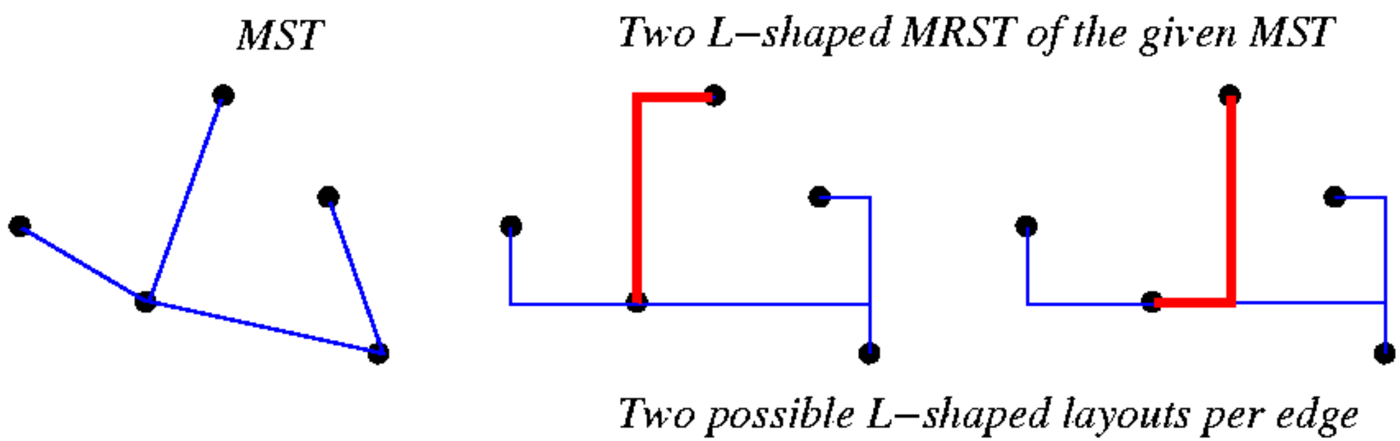
- **Hanan's Thm:** There exists an MRST with all Steiner points (set *S*) chosen from the intersection points of horizontal and vertical lines drawn from points of *P*.
  - Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Applied Math.*, 1966.



- **Hwang's Theorem:** For any point set *P*, $\frac{Cost(MST(P))}{Cost(MRST(P))} \leq \frac{3}{2}$.
  - Hwang, "On Steiner minimal tree with rectilinear distance," *SIAM J. Applied Math.*, 1976.
- Better approximation algorithm with the performance bound 61/48
  - Foessmeier *et al*, "Fast approximation algorithm for the rectilinear Steiner problem," Wilhelm Schickard-Institut für Informatik, TR WSI-93-14, 93.

Minimum Spanning Tree Based Heuristic

- Ho, Vijayan, and Wong, “New algorithms for the rectilinear Steiner problem,” TCAD-90.
    1. Construct an RST from an MST.
    2. Each edge is straight or L-shaped.
    3. Maximize overlaps by dynamic programming.
- About 8% smaller than *Cost*(*MST*).



Repeated 1-Steiner Tree Heuristic

- A. B. Kahng and G. Robins, "A new class of Steiner tree heuristics with good performance: the iterated 1-Steiner approach," ICCAD, 1990.
- **1-Steiner tree problem**: the minimal Steiner tree problem with the restriction that the tree contains only a single Steiner point.
- The optimal solution of the 1-Steiner tree problem can be found efficiently.
- The **repeated 1-Steiner tree heuristic** gives provably good results (but no optimal solution is guaranteed).

```
(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);
}
```

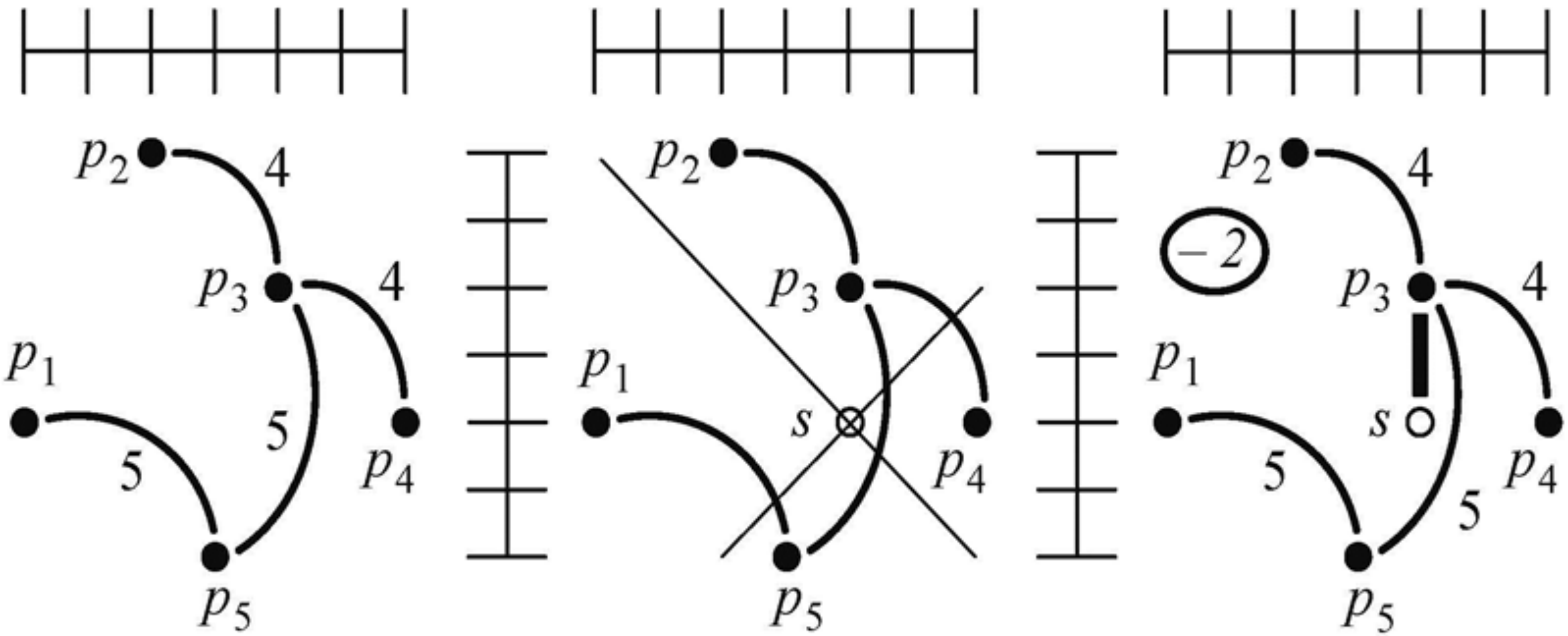
1-Steiner: Try All Hanan Points

```
(set of struct vertex, set of struct edge, int)
  1-steiner(set of struct vertex V, set of struct edge E)

  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);
}
```

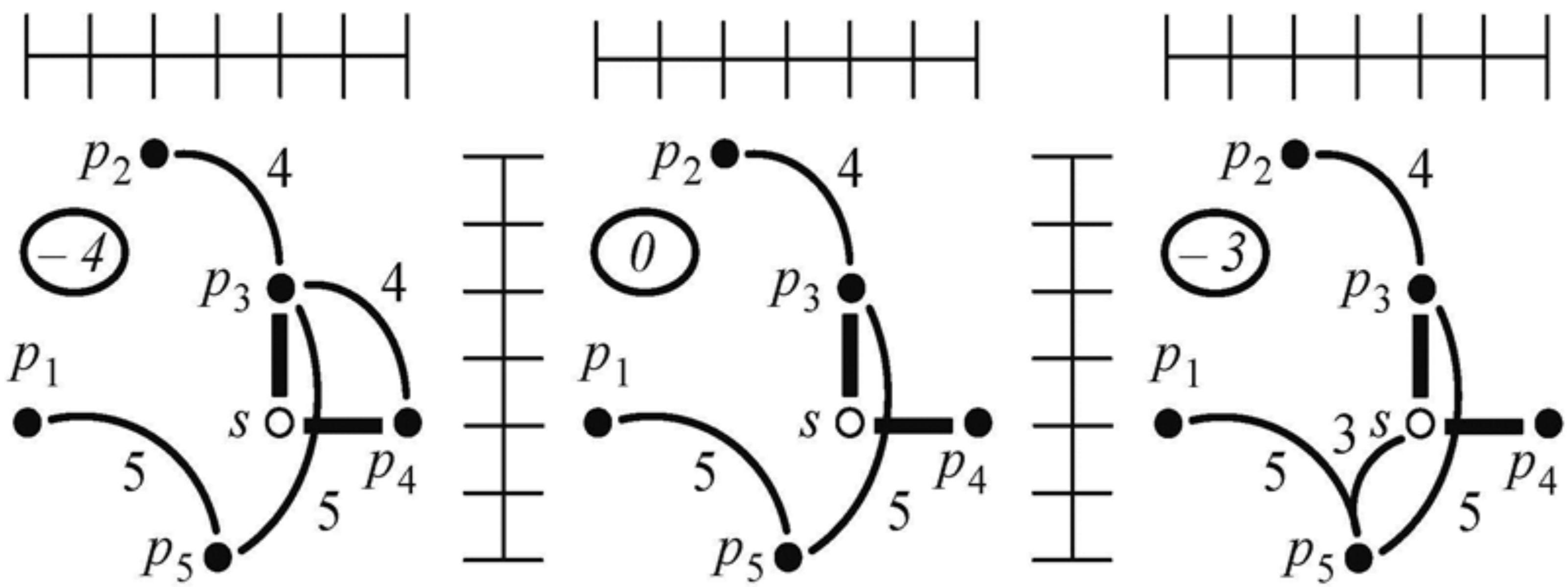
Getting Spanning Tree Incrementally

```
(set of struct vertex, set of struct edge, int)
  spanning_update(set of struct vertex V, set of struct edge E, struct vertex s)

  delta ← 0;
  V ← V ∪ {s};
  for each d ∈ {north, east, south, west} {
    u ← closest_point(V, s, d);
    delta ← delta - distance(s, u);
    E ← E ∪ {(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);
}
```

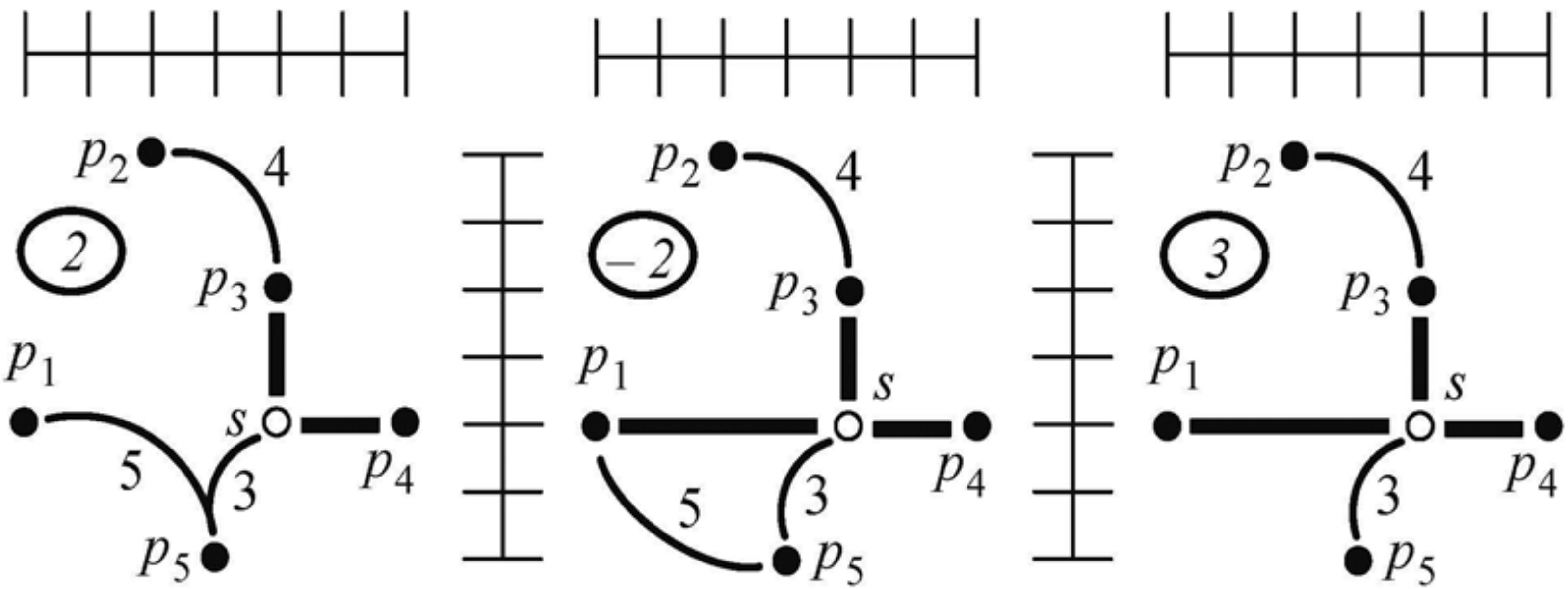
Example



Example (cont'd)



Example (cont'd)



FLUTE: Fast LookUp Table Estimation

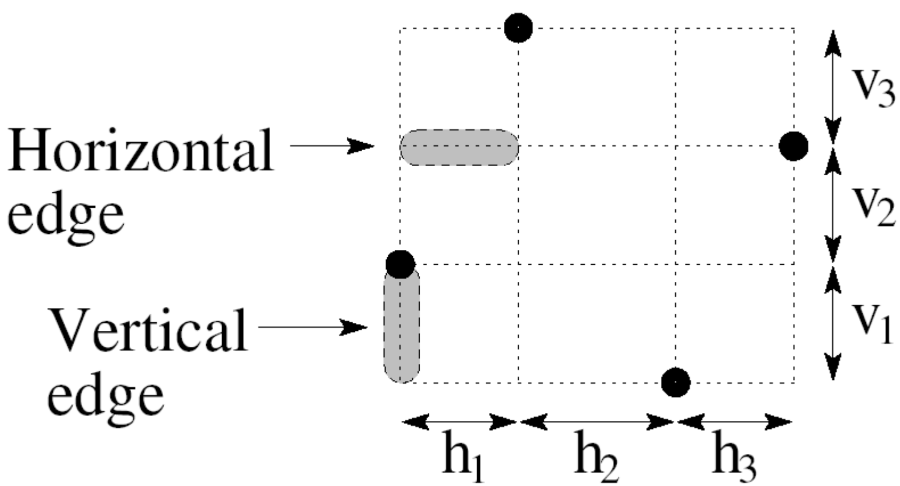
- Related publications
  - C. Chu, “FLUTE: Fast Lookup Table Based Wirelength Estimation Technique”, ICCAD 2004. (FLUTE 1.0)
  - C. Chu and Y.-C. Wong, “Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”, ISPD 2005. (FLUTE 2.0)
  - C. Chu and Y.-C. Wong, “FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”, TCAD 2008. (FLUTE 2.5)
  - Y.-C. Wong and Chris Chu, “A Scalable and Accurate Rectilinear Steiner Minimal Tree Algorithm”, VLSI-DAT 2008. (FLUTE 3.0)

Overview

- Basic idea
  - Lookup table 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
- 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

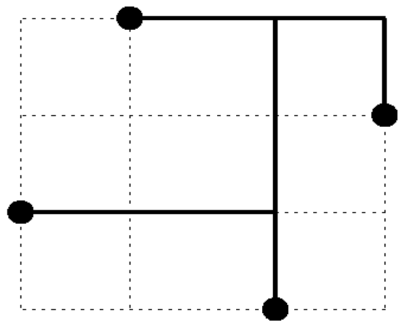
Preliminaries

- Consider routing along 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



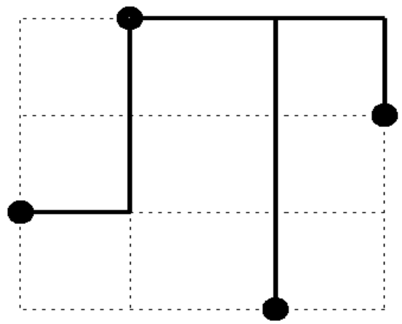
Wirelength Vector (WV)

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



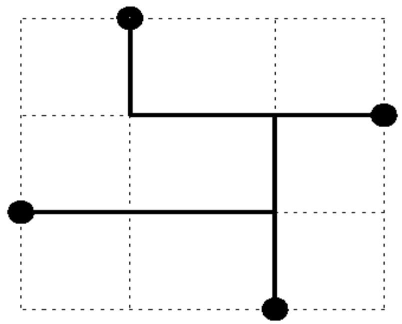
$$WL = h_1 + 2h_2 + h_3 + v_1 + v_2 + 2v_3$$

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



$$WL = h_1 + h_2 + h_3 + v_1 + 2v_2 + 3v_3$$

(1, 1, 1, 1, 2, 3)



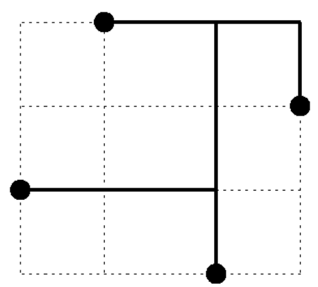
$$WL = h_1 + 2h_2 + h_3 + v_1 + v_2 + v_3$$

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

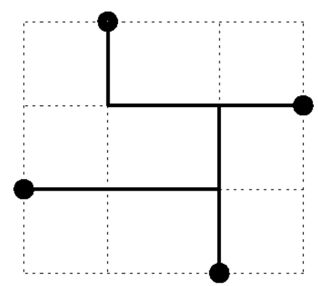
- WL can be expressed as a vector of the coefficients called Wirelength Vector (WV)

Potentially Optimal WV (POWV)

- Optimal WL can be found by enumerating all WVs
- However, most WVs can never produce optimal WL
  - (1, 2, 1, 1, 1, <u>2</u>) is redundant as it always produces a larger WL than (1, 2, 1, 1, 1, <u>1</u>)



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



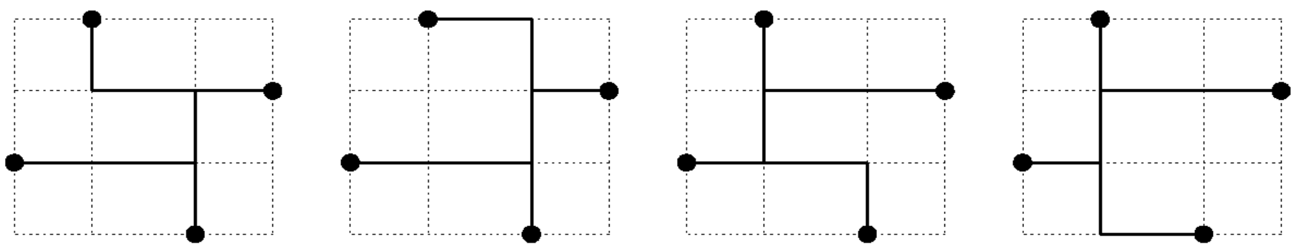
(1, 2, 1, 1, 1, 1)

- Potentially Optimal Wirelength Vector (POWV) is a WV that *may* produce optimal WL

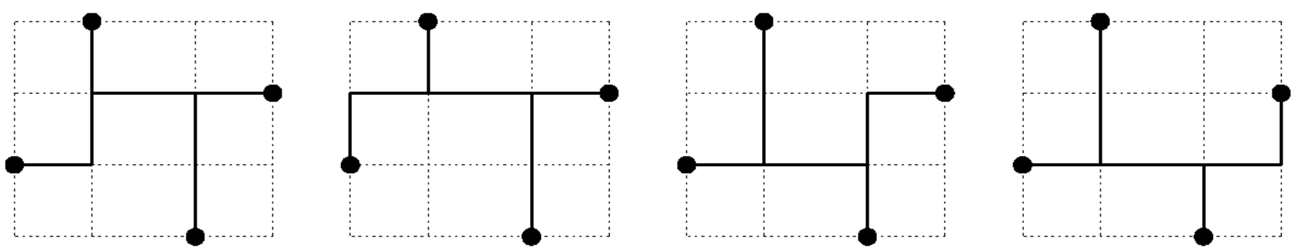
Number of POWVs

- 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)

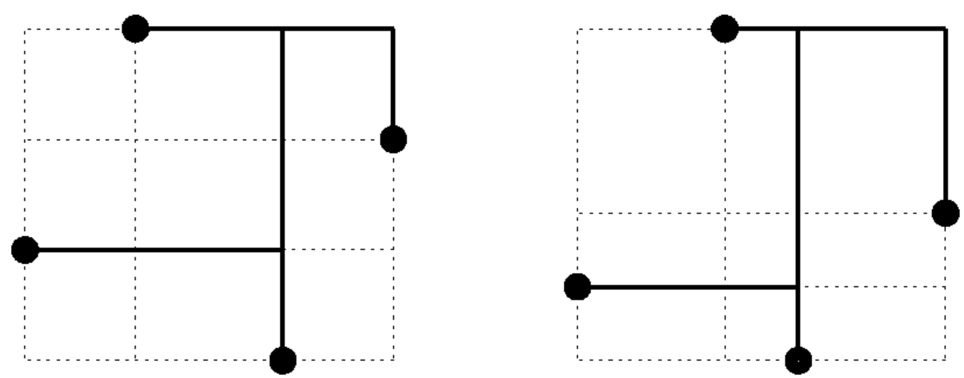


POWV
(1,1,1,1,2,1)



Sharing of POWVs Among Nets

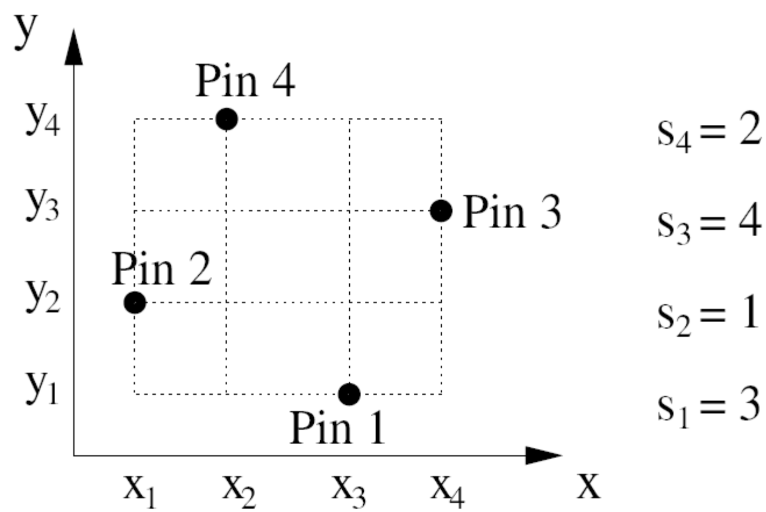
- To find optimal WL, we can pre-compute all POWVs and store them in a lookup table
- However, there are infinite number of different nets
- Try to group together nets that can share the same set of POWVs
- For example, these following two nets share the same set of POWVs:



Grouping by Position Sequence

- Define position sequence $s_1 s_2 \ldots s_n$ to be the list of ranks of pins in x-coordinate

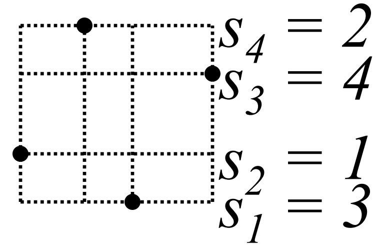
Position sequence = 3142



- **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 for WL Estimation

- Given a net
  1. Find the position sequence

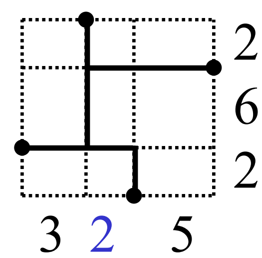


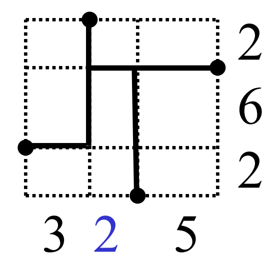
Position sequence: *3142*

  2. Get the POWVs from LUT

POWVs:
(1,2,1,1,1,1) (1,1,1,1,2,1)

  3. Find the edge lengths





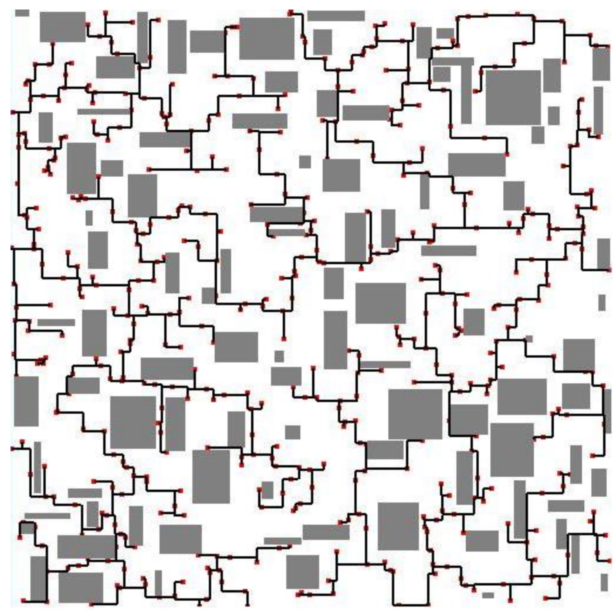
  4. Find WL for each POWV and return the best

HPWL + 2 = 22 HPWL + 6 = 26

Return

Obstacle-Avoiding Rectilinear Steiner Minimal Tree

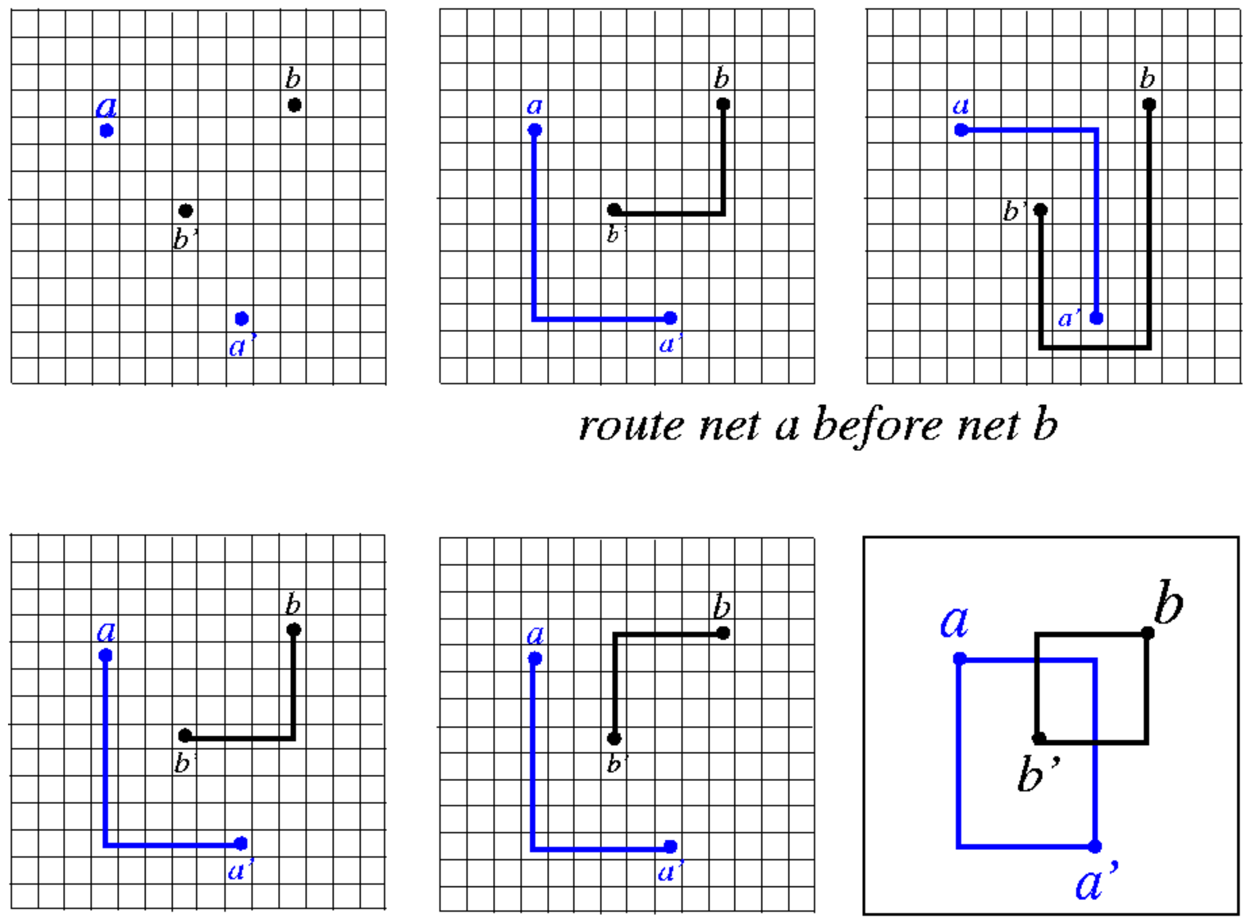
- Obstacles: macro/IP blocks, power/ground network, etc.
  - Rectangular shapes vs. rectilinear shapes
- Single routing layer



- Multiple routing layers
  - Preferred directions vs. non-preferred directions

Net Ordering

- Net ordering greatly affects routing solutions.
- In the example, we should route net *b* before net *a*.

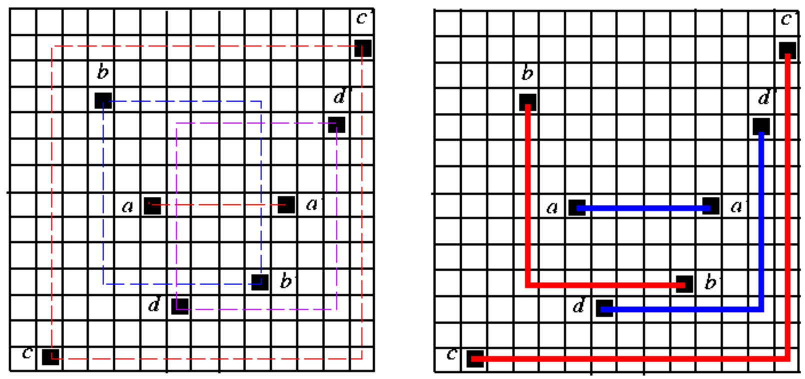


route net a before net b

route net b before net a

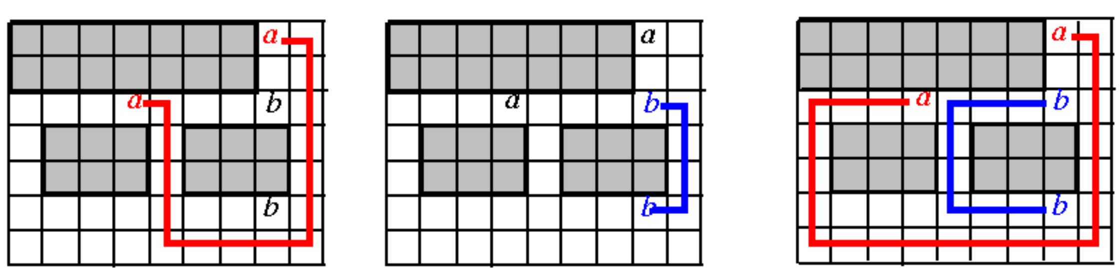
Net Ordering (cont'd)

- Order the nets in the ascending order of the # of pins within their bounding boxes.



routing ordering: a (0) –> b (1) –> d (2) –> c (6)

- Order the nets in the ascending (or descending??) order of their length.
- Order the nets based on their timing criticality.
- A mutually intervening case:



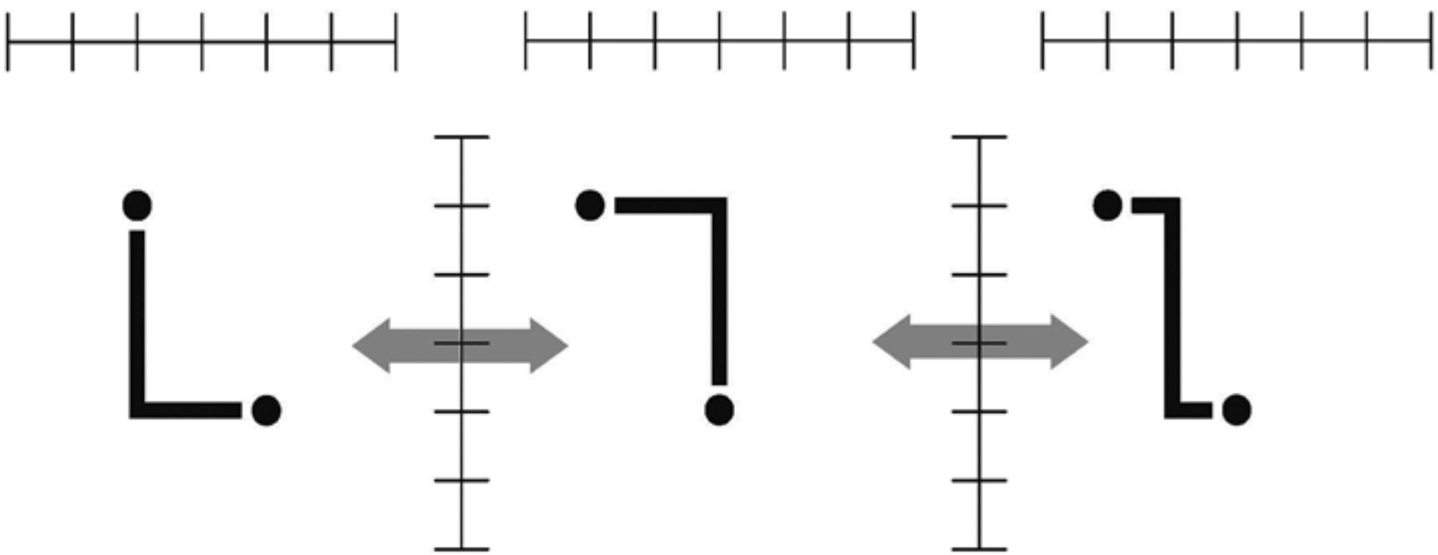
a prevents routing of b   *b prevents routing of a*   *a feasible routing*

Rip-Up and Re-routing

- Rip-up and re-routing is required if a global or detailed router fails in routing all nets.
- Two steps in rip-up and re-routing
  1. Identify bottleneck regions, rip off some already routed nets.
  2. Route the blocked connections, and re-route the ripped-up connections.
- Repeat the above steps until all connections are routed or a time limit is exceeded.

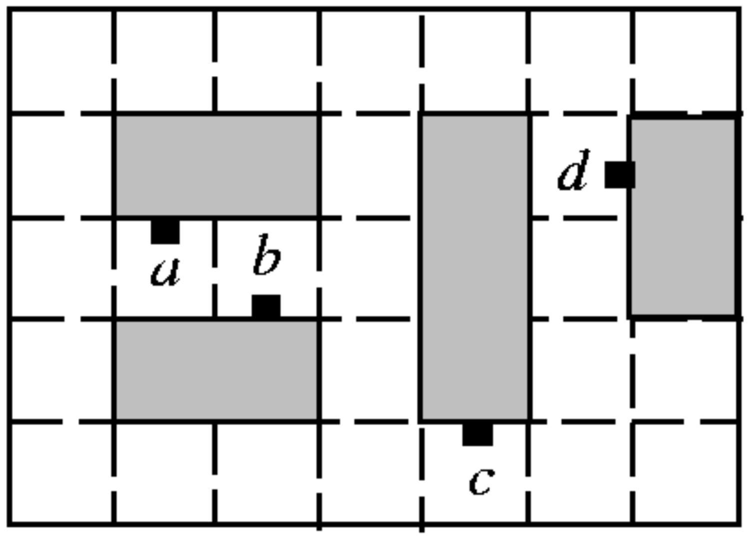
Local Transformations

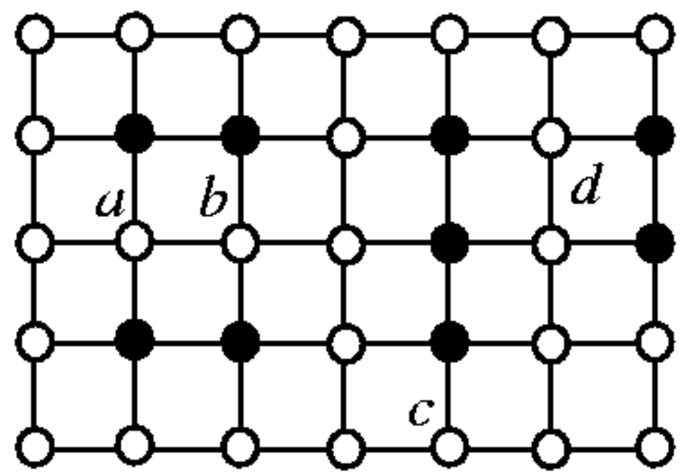
- Over-congestion after net-independent Steiner-tree construction can be eliminated by local transformations (e.g. guided by simulated annealing) or by ripping up a net and rerouting it by maze routing



Graph Models for Global Routing: Grid Graph

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





Graph Model: Channel Intersection Graph

- Channels are represented as edges.
- Channel intersections are represented as vertices.
- Edge weight represents channel capacity.
- Extended channel intersection graph: terminals are also represented as vertices.