Global Routing

Placement

↓

Global Routing

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

↓

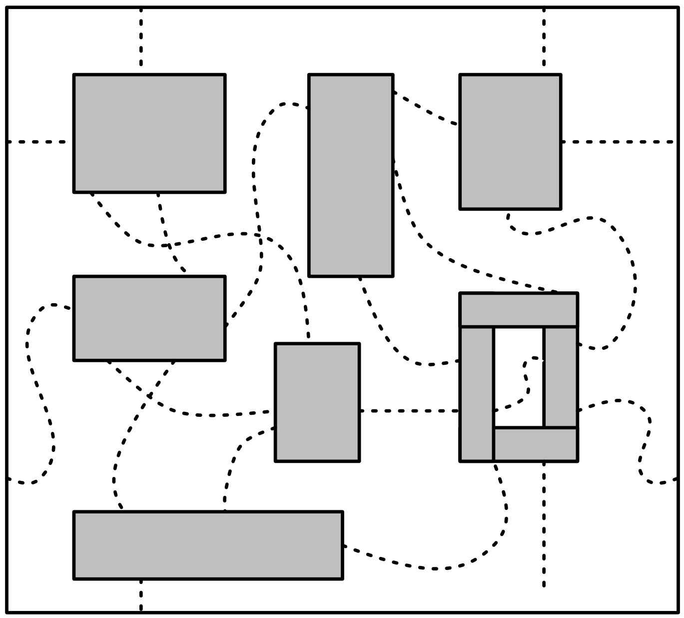
Detailed Routing

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

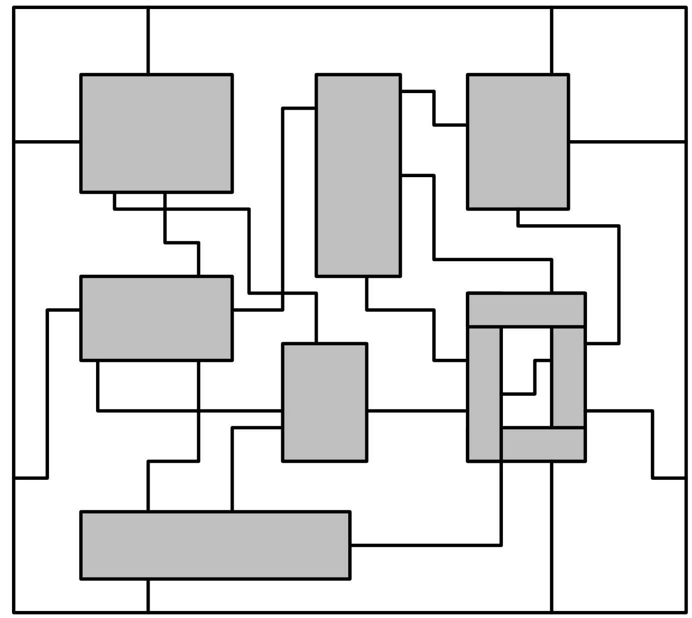
↓

Compaction

A Routing Example



Global Routing
(a)



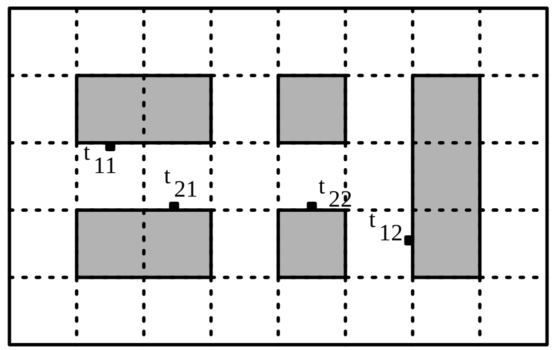
Detailed Routing
(b)

Graph Models used in Global Routing

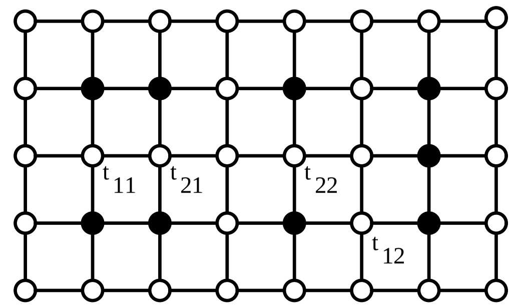
- Graphs are used to represent the geometric information.

  1. Grid Graph Model
  2. Checker Board Model
  3. Channel Intersection Graph Model

Grid Graph Model



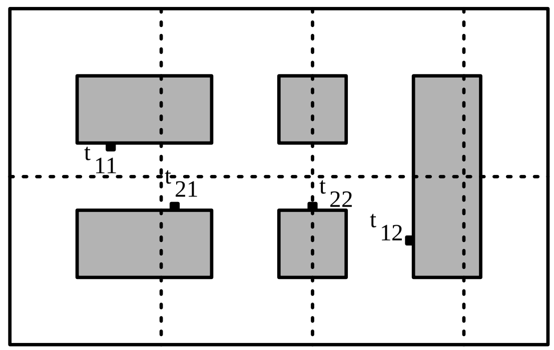
(a)



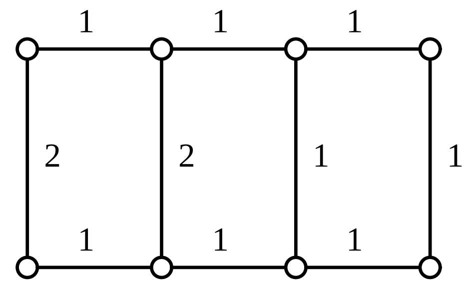
(b)

- 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.

Checker Board Graph



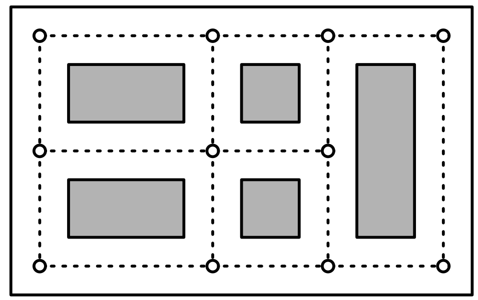
(a)

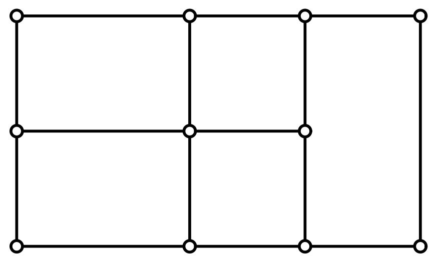


(b)

- Checker board graph is generated in a manner similar to the grid graph except that the superimposed grid is a 'coarse' grid.

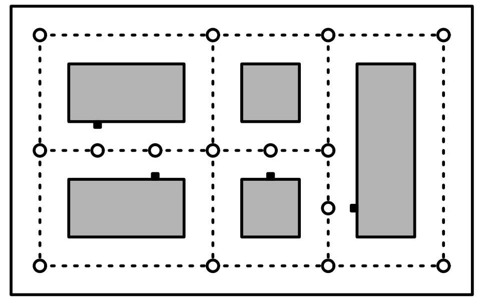
Channel Intersection Graph


(a)

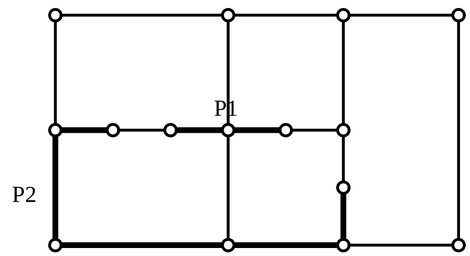

(b)

- Channels are represented as edges.
- Channel intersections are represented as vertices.
- Edge weight represents channal capacity.

Extented Channel Intersection Graph



(a)



(b)

- Terminals are also represented as vertices.

Problem Formulation

- Given, a net-list $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$, the routing graph $G = (V, E)$, find a Steiner tree T_i for each net $N_i, 1 \leq i \leq n$, such that,

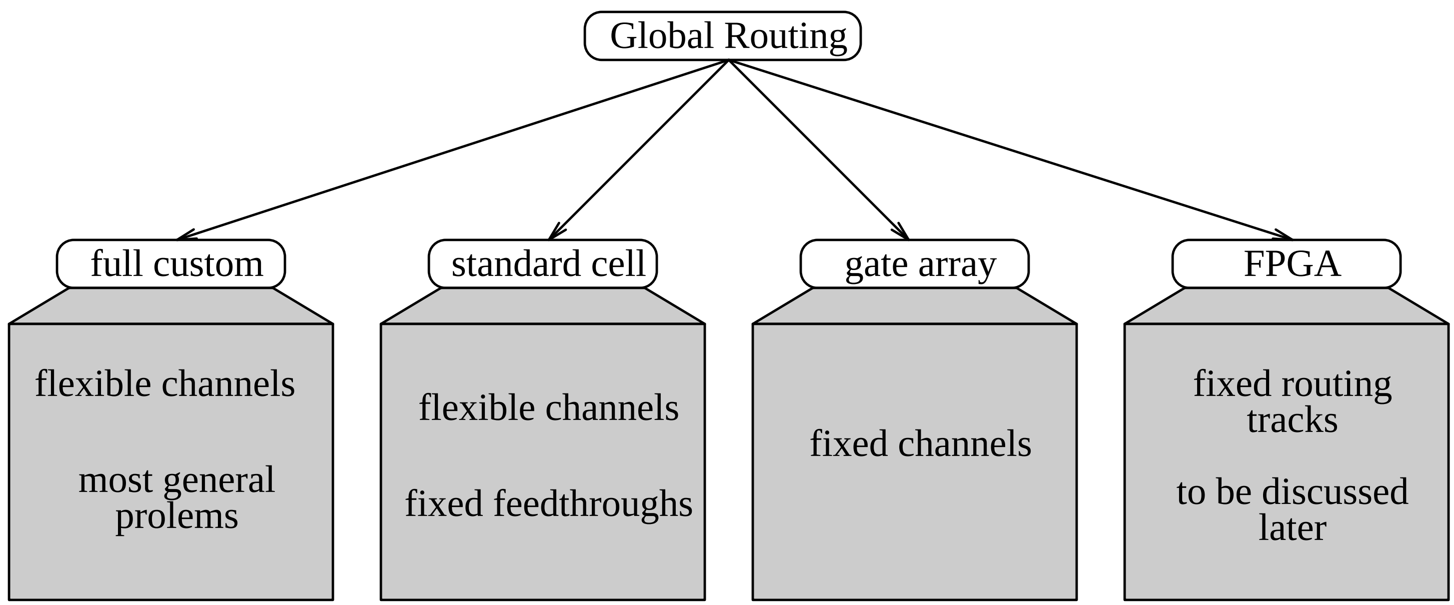
$$U(e_j) \leq c(e_j) \text{ for all } e_j \in E$$
$$\Sigma_{i=1}^{n} L(T_i) \text{ is minimized}$$

Where

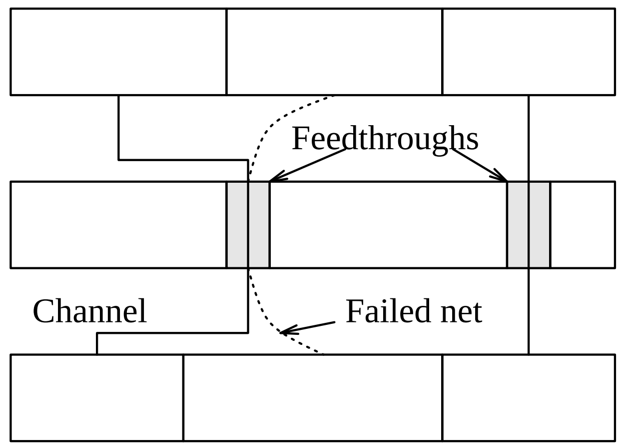
1. $c(e_j)$ = capacity of edge e_j,
2. $x_{ij} = 1$ if e_j is in T_i, $x_{ij} = 0$ otherwise,
3. $U(e_j) = \Sigma_{i=1}^{n} x_{ij}$ = the number of wires that pass through the channel corresponding to edge e_j,
4. $L(T_i)$ = the length of Steiner tree T_i.

- In case of high-performance, the maximum wire length ($\max_{i=1}^{n} L(T_i)$) is minimized.

Global Routing in Different Design Styles



Global Routing in Standard Cell



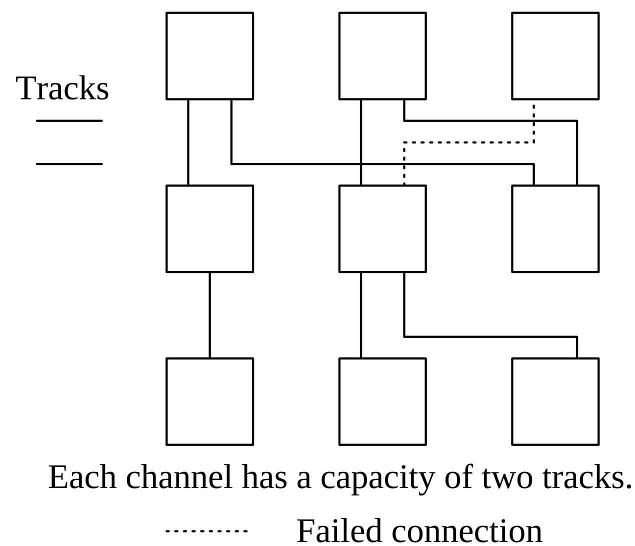
Objective:

- Minimize total channel height.
- Assignment of feedthroughs?
  - Placement
  - Global routing

In case of high-performance:

- Minimize the maximum wire length.
- Minimize the maximum path length.

Global Routing in Gate Array



Objective:

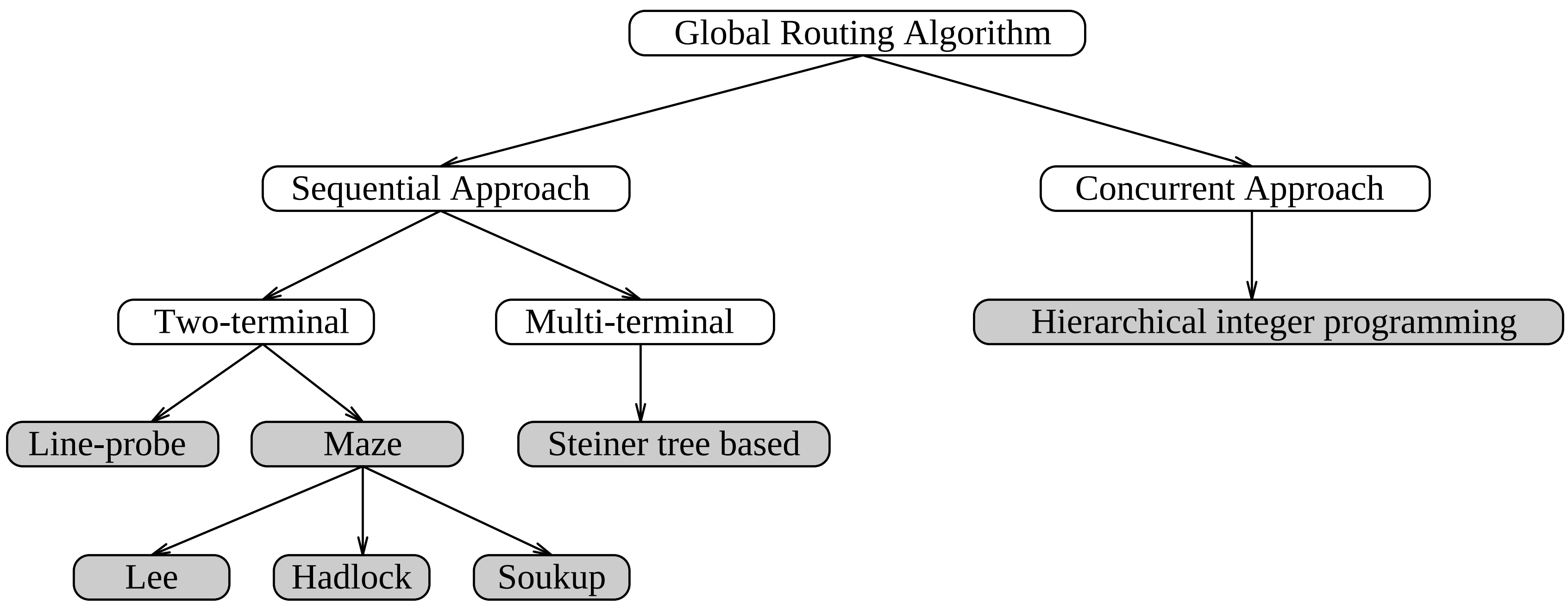
- Gurantee routability.

In case of high-performance:

- Minimize the maximum wire length.
- Minimize the maximum path length.

Classification of Global Routing Algorithms

1. Sequential Approach: Assign priority to nets. Routes one net at a time based on its priority.
2. Concurrent Approach: All nets are considered at the same time.



Features of Lee's Algorithm

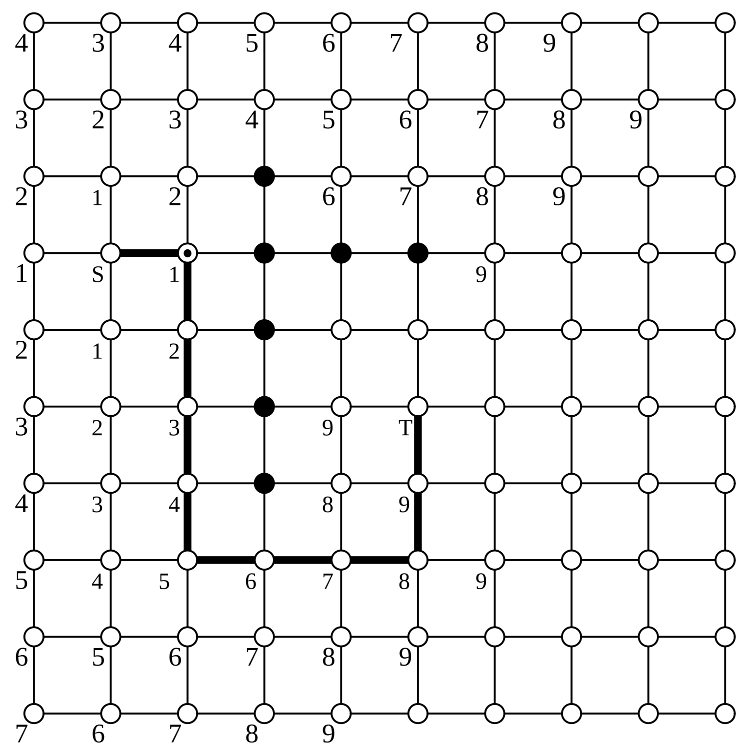
- Breadth-first search.
- Works on grid nodes.
- Can be visualized as a wave propagating from the source.
- Time and space complexities are $O(h \times w)$ for a grid of dimensions $h \times w$.
- Finds the shortest path between source and target.

C. Y. Lee *TCOMP* 1961

Lee's Algorithm

```
Algorithm LEE-ROUTER (B, s, t, P)
    input: B, s, t
    output: P
begin
    plist = s; nlist = φ; temp = 1;
    path_exists = FALSE;
    while plist ≠ φ do
        for each vertex v_i in plist do
            for each vertex v_j neighboring v_i do
            if B[v_j] = UNBLOCKED then
                L[v_j] = temp; INSERT(v_j,nlist);
                if v_j = t then
                    path_exists = TRUE; exit while;
        temp=temp + 1; plist = nlist; nlist = φ;
    if path_exists = TRUE then RETRACE (L, P);
    else path does not exist;
end.
```

Example of Lee's Algorithm



Features of Soukup's Router

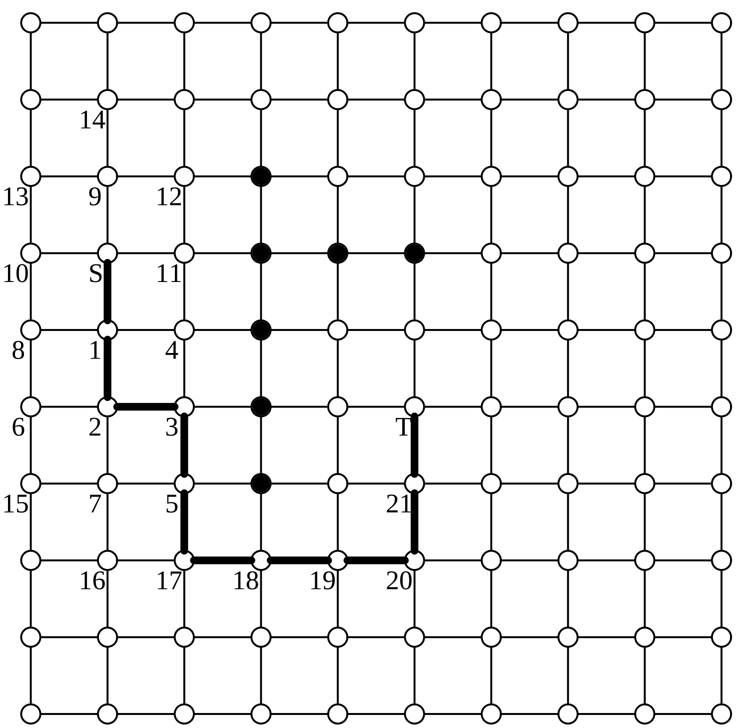
- Combined breadth-first and depth-first search.
- Works on grid nodes.
- Depth-first search is used to explore in the direction toward target until an obstacle or the target is reached.
- Breadth-first search is used if an obstacle is reached.
- Time and space complexities are $O(h \times w)$
- Finds a path between source and target.
- May not find the shortest path.

J. Soukup *DAC* 1978

Soukup's Router

```
Algorithm SOUKUP-ROUTER (B, s, t, P)
begin
  plist = s; nlist = φ; temp = 1; path_exists = FALSE;
  while plist ≠ φ do
    for each vertex v_i in plist do
      for each vertex v_j neighboring v_i do
        if v_j = t then
          L[v_j] = temp; path_exists = TRUE; exit while;
        if B[v_j] = UNBLOCKED then
        (* If the direction of the search is toward the target,
          the search continues in this direction *)
          if DIR(v_i, v_j) = TO-TARGET
          then L[v_j] = temp; temp = temp + 1; INSERT (v_j, plist);
            while B[NGHBR-IN-DIR(v_i, v_j)] = UNBLOCKED do
              v_j = NGHBR-IN-DIR(v_i, v_j); L[v_j] = temp;
              temp = temp + 1; INSERT (v_j, plist);
          else
            L[v_j] = temp; temp = temp + 1; INSERT (v_j, nlist);
    plist = nlist; nlist = φ;
  if path_exists = TRUE then RETRACE (L, P); else path does not exist;
end.
```

Example of Soukup's Router



Features of Hadlock's Algorithm

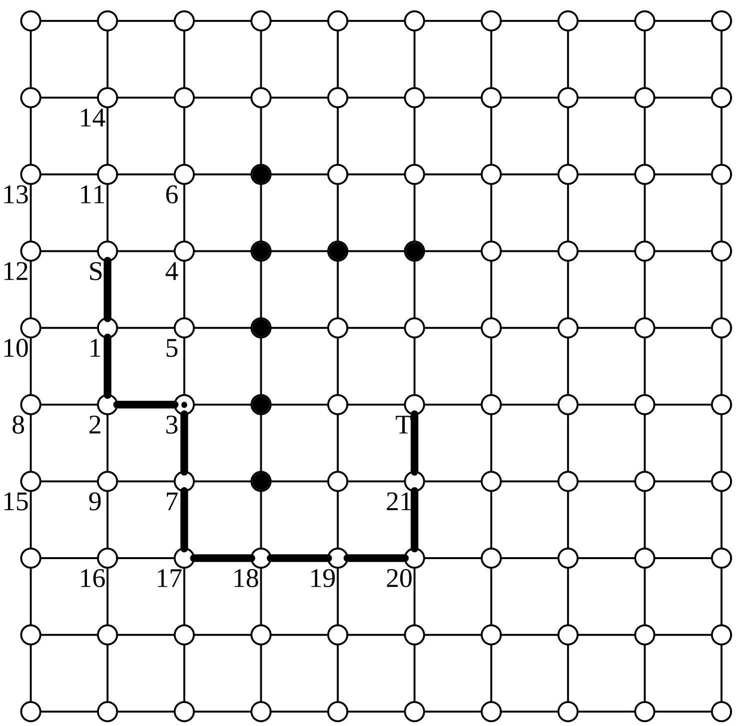
- A* search.
- Works on grid nodes.
- It uses detour number instead of labelling wavefront in Lee's router.
- Minimizes the detour number.
- Time and space complexities are $O(h \times w)$.
- Finds the shortest path between source and target.

F. O. Hadlock *Networks* 1977

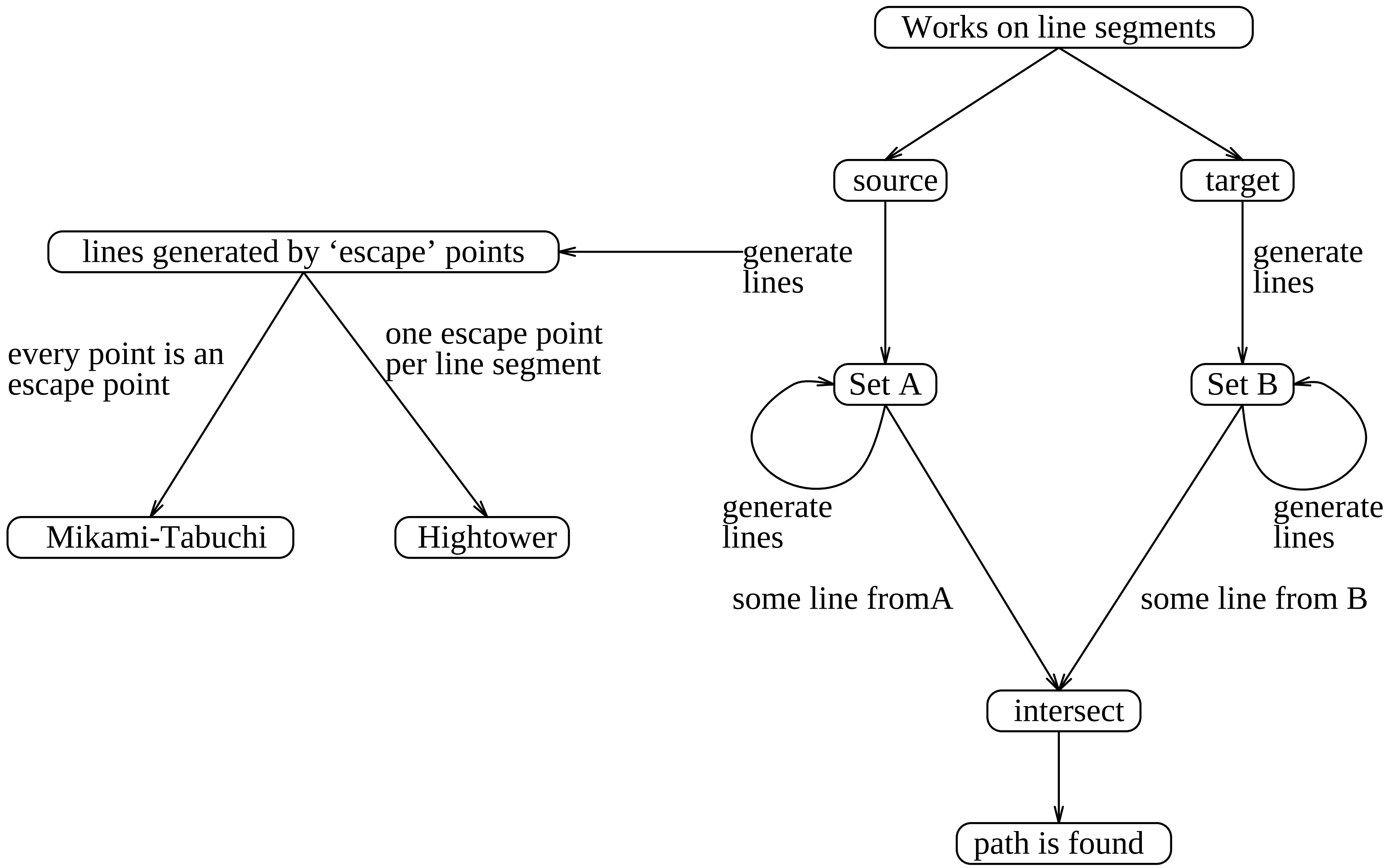
Hadlock's Algorithm

```
Algorithm HADLOCK-ROUTER(B, s, t, P)
begin
    plist = s; nlist = φ; detour = 0; path_exists = FALSE;
    while plist ≠ φ do
        for each vertex v_i in plist do
            for all vertices v_j neighboring v_i do
                if B[v_j] = UNBLOCKED then
                    D[v_j] = DETOUR-NUMBER(v_j); INSERT (v_j,nlist);
                    if v_j = t then
                        path_exists = TRUE; exit while;
        if nlist = φ then
            path_exists = FALSE; exit while;
        detour = MINIMUM-DETOUR( nlist );
        for each vertex v_k in nlist do
            if D[v_k] = detour then INSERT(v_k, plist);
        DELETE (nlist, plist);
    if path_exists = TRUE then RETRACE (L, P);
    else path does not exist;
end.
```

A net routed by Hadlock's Algorithm



Features of Line Probe Algorithm

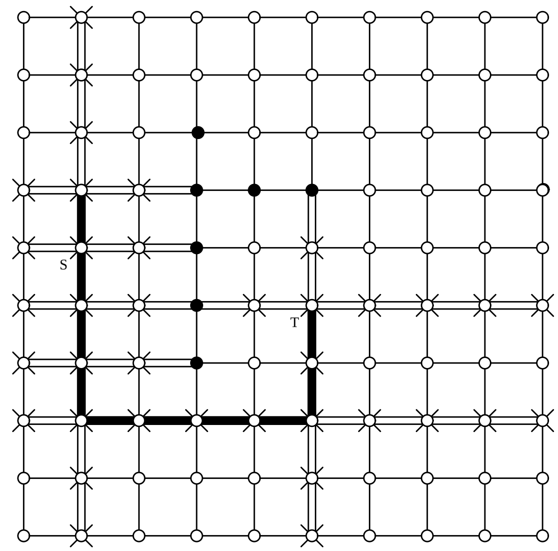


- Time and space complexities are $O(L)$ where L is the total number of line segments generated.

Line Probe Algorithm

Algorithm LINE-PROBE-ROUTER(s, t, P)
begin
  new_slist = line segments generated from s;
  new_tlist = line segments generated from t;
  while $new\_slist \neq \phi$ **and** $tlist \neq \phi$ **do**
    $slist = new\_slist$; $tlist = new\_tlist$;
    for each line segment l_i in $slist$ **do**
      for each line segment l_j in $tlist$ **do**
        if INTERSECT(l_i, l_j)=TRUE **then**
          $path\_exists$ = TRUE; exit while;
    $new\_slist = \phi$;
    for each line segment l_i in $slist$ **do**
      for each escape point e on l_i **do**
        GENERATE(l_k, e); INSERT(l_k, new_slist);
    $new\_tlist = \phi$;
    for each line segment l_i in $tlist$ **do**
      for each escape point e on l_i **do**
        GENERATE(l_k, e); INSERT(l_k, new_tlist);
  if $path\_exists$=TRUE **then** RETRACE;
  else a path can not be found;
end.

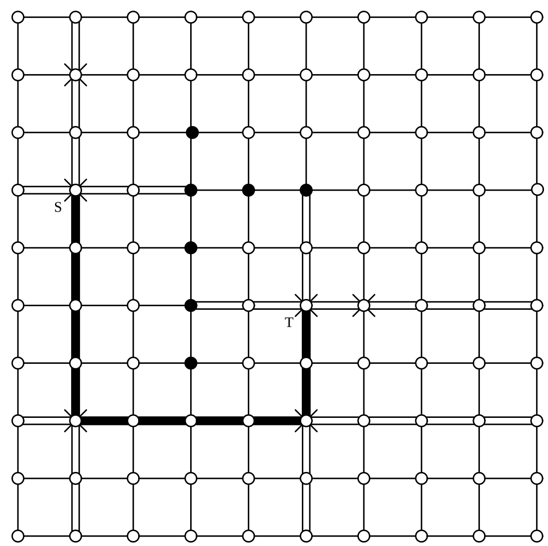
Example of Mikami-Tabuchi's Algorithm



- Every grid point is an escape point.

K. Mikami and K. Tabuchi *IFIPS Proc.* 1968

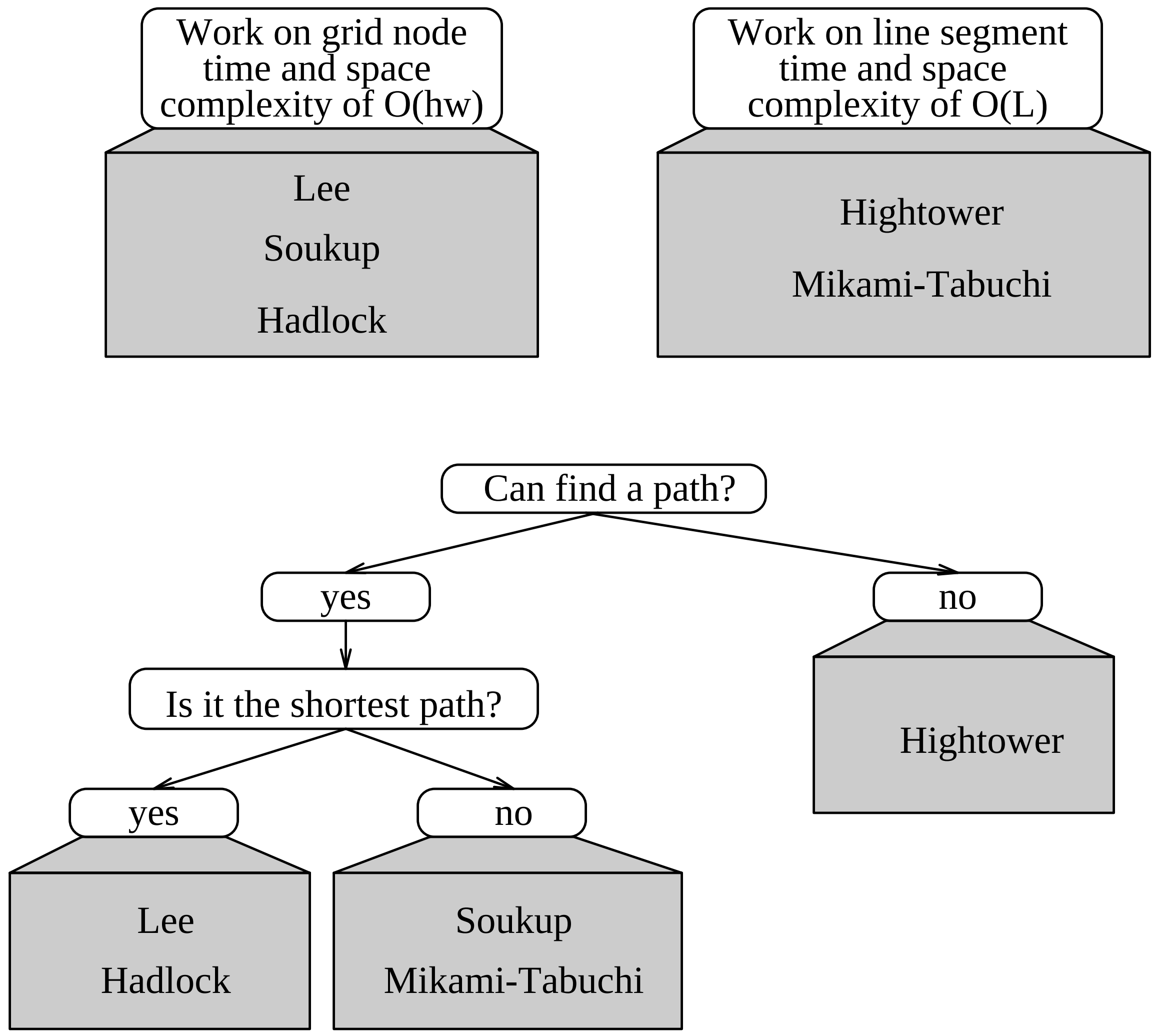
Example of Hightower's Algorithm



- A single 'escape' point on each line segment.
- In the simple case of a probe parallel to the blocked vertices, the escape point is placed just past the endpoint of the segment.

D. W. Hightower *Proc. 6th Design Automation Workshop* 1969

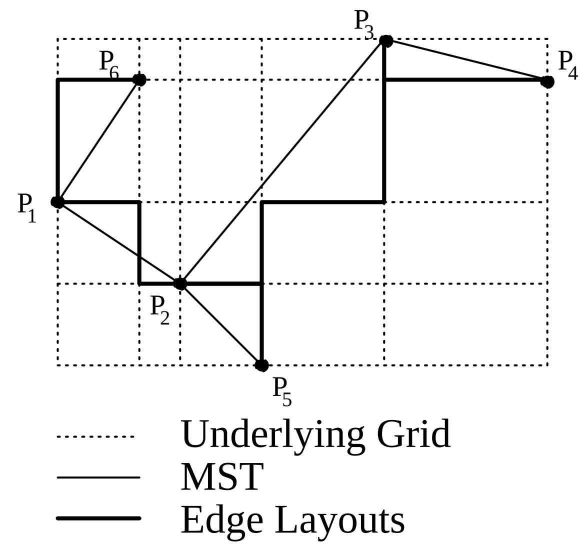
Comparison of Different Algorithms



Comparison of Different Algorithms

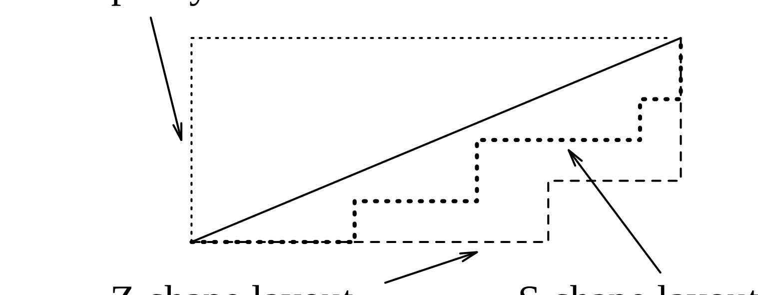
	Algorithms				
	Maze Routing			Line-Probe	
	Lee	Soukup	Hadlock	Mikami	Hightower
Time complexity	$h \times w$	$h \times w$	$h \times w$	L	L
Space complexity	$h \times w$	$h \times w$	$h \times w$	L	L
Finds path if one exists?	yes	yes	yes	yes	no
Is the path shortest?	yes	no	yes	no	no
Works on grids or lines?	grid	grid	grid	line	line

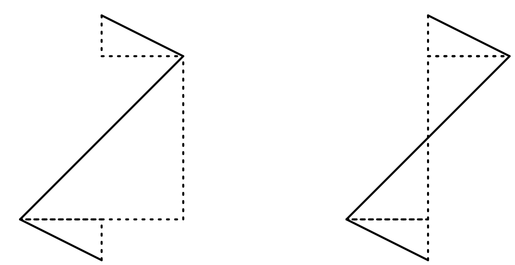
Concepts Used in Steiner Tree Algorithms

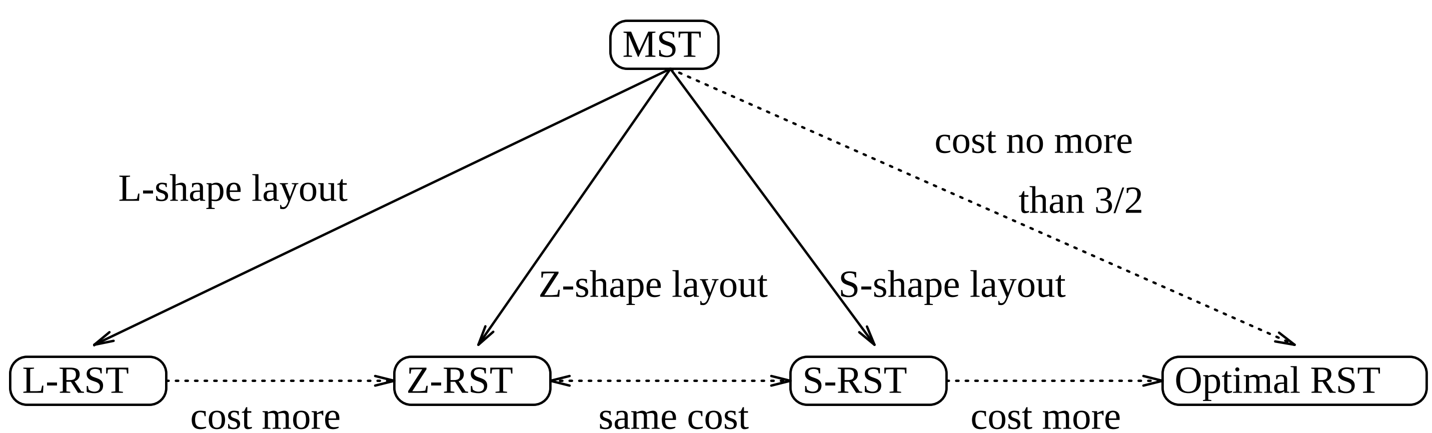


- Two-terminal net: path is used.
- Multi-terminal net: Steiner tree is used, which is a tree connecting all terminals of the net and some other points.
- RST is a Steiner tree with only rectilinear edges.
- An optimal RST problem is NP-complete problem.
- A RST obtained by rectilinearizing edges of MST is used.

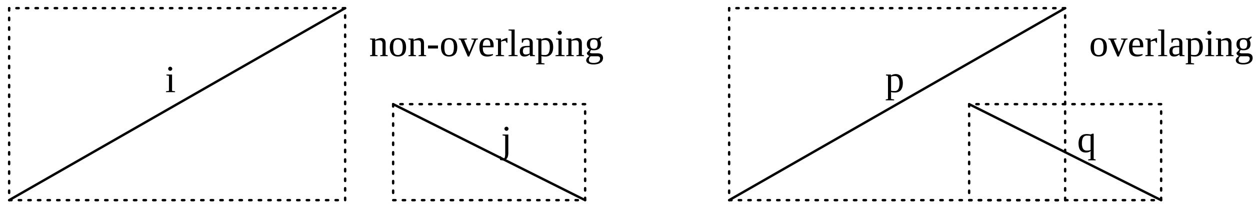
Different Layout Shapes





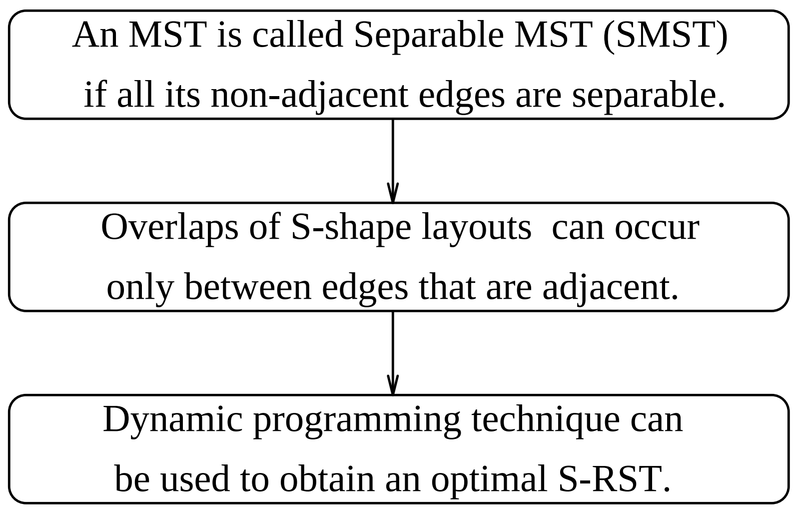


Separable MST

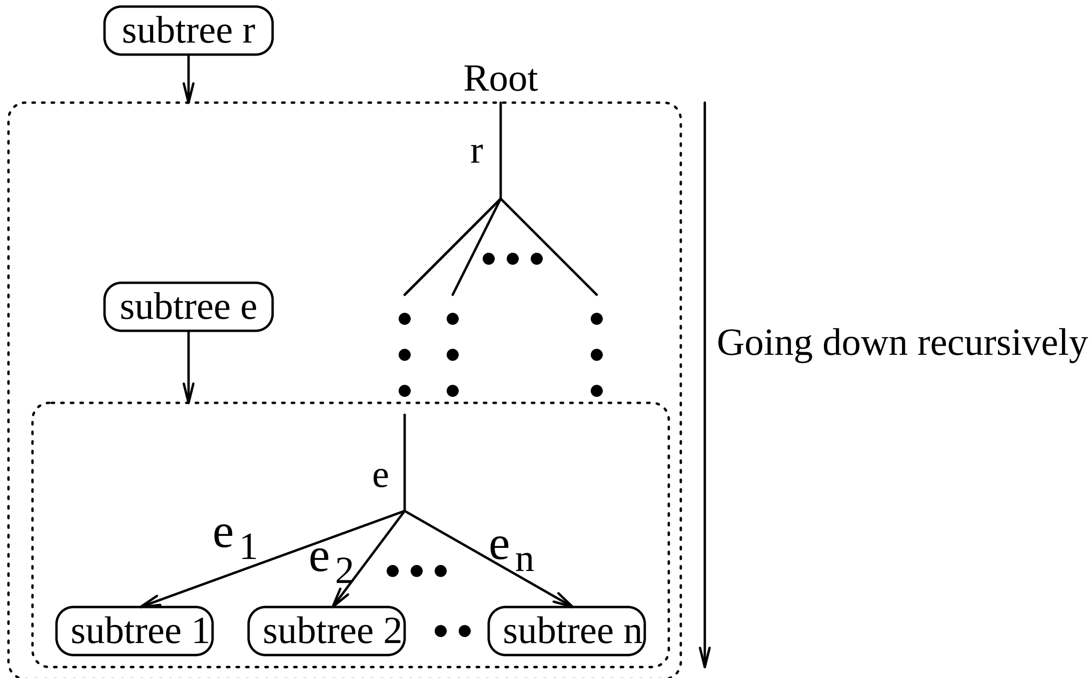


Edges i and j are called separable

p and q are not separable



Features of Algorithm Z-RST



- Subtree e is obtained by finding the minimum total weight among all combinations of z-shape layouts of e and $e_1, \ldots, e_n$ with subtree 1, ..., subtree n.
- Optimal Z-RST is obtained by finding the subtree r.
- It has the same cost as the optimal S-RST.

J. M. Ho and G. Vijayan and C. K. Wong *ICCAD* 1989

Algorithm Z-RST

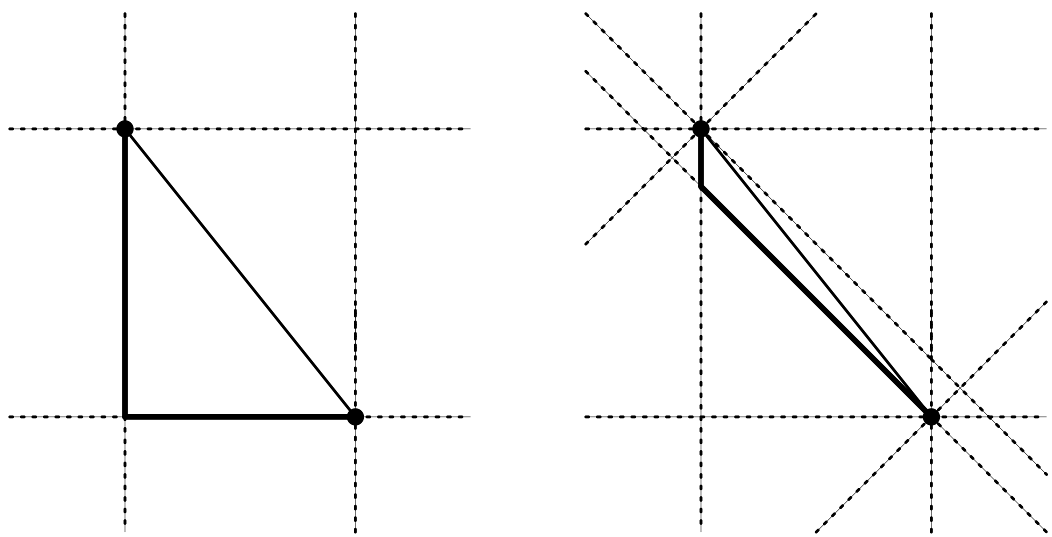
Algorithm Z-RST $(r, T_r, CostM, M)$
  input: r, T_r
  output: $CostM, M$
begin
  $CostM = \infty$;
  for each z-shaped layout z of r **do**
    LEAST-COST $(z, T_r, CostTempM, TempM)$;
    if $CostTempM < CostM$ **then**
      $M = TempM$;
      $CostM = CostTempM$;
end.

Function LEAST-COST

Function LEAST-COST($z, T_e, CostM_z[e], M_z[e]$)
    input: z, T_e
    output: $CostM_z[e], M_z[e]$
begin
    if CHILD-EDGES-NUM(e) $\neq 0$ **then**
        for each child edge e_i of e **do**
            for each layout z_{ij} of e_i **do**
                LEAST-COST($z_{ij}, T_{e_i}, CostM_{z_{ij}[e_i]}, M_{z_{ij}}[e_i]$);
        for (each combination of the layouts containing one
            optimal Z-RST layout for each T_{e_i}) **do**
            merge layouts in the combination with the layout z
            of edge e;
            calculate the resulting cost of merged layout;
        $CostM_z[e]$ = minimum cost among all merged layouts;
        $M_z[e]$ = the layout corresponding to the minimum cost;
    else (* The bottom edge is reached *)
        $M_z[e] = z$;
        $CostM_z[e]$ = cost of z;
end.

Rectilinearizations in Different Geometries

- In δ-geometry, edges with angles $i\pi/\delta$ for all i are allowed.
- When $\delta = 2$, δ-geometry is the Euclidean geometry.
- Any MST in a plane is δ-separable for any even $\delta \geq 4$.
- $10\% - 12\%$ reduction in tree length can be achieved using 4-geometry.
- Length reductin is marginal for higher geometries.



(a) 2-geometry (b) 4-geometry

Integer Programming Approach

- x_{ij} are the integer variables to be solved satisfying:

  $$\Sigma_{j=1}^{l_i} x_{ij} = n_i, i = 1, \ldots, n$$

  $$\Sigma_{(ij), e \in T_{ij}} x_{ij} + x_e = c(e), e \in E$$

  $$\Sigma_{i=1}^{n} \Sigma_{j=1}^{l_i} l(T_{ij}) \times x_{ij} \text{ should be minimized}$$

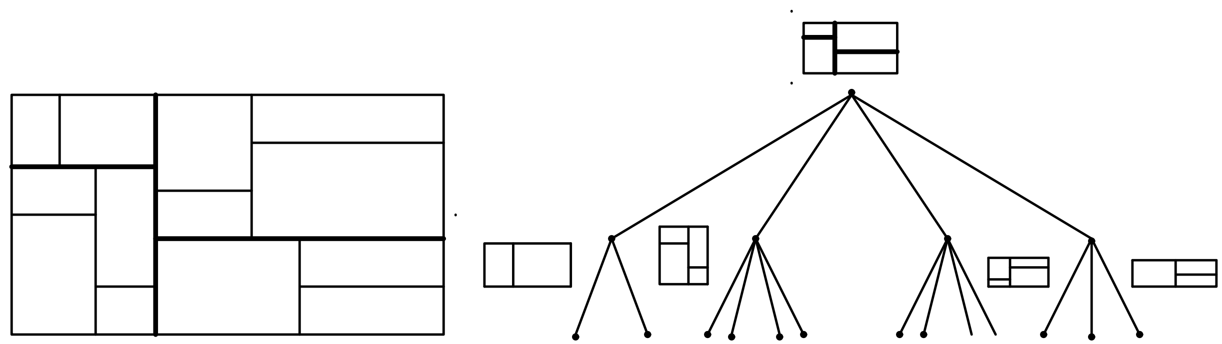
  where

  1. $S = \{S_i | 1 \leq i \leq n\}$ denotes a set of sets of vertices in the routing graph $G = (V, E)$.
  2. $T = \{T_{ij}\}, j = 1, 2, \ldots, l_i$, denotes a set of Steiner trees for $S_i, i = 1, 2, \ldots, n$.
  3. x_e is a slack variable for edge e which denotes the free capacity of e.
  4. $l(T_{ij})$ is the length of the Steiner tree T_{ij}.

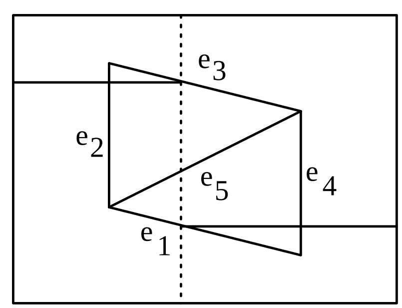
- The problem is usually too large to be solved efficiently. Hierarchical methods are used to break the problem into small problems.

Cut-tree and Routing Graph for Floorplan

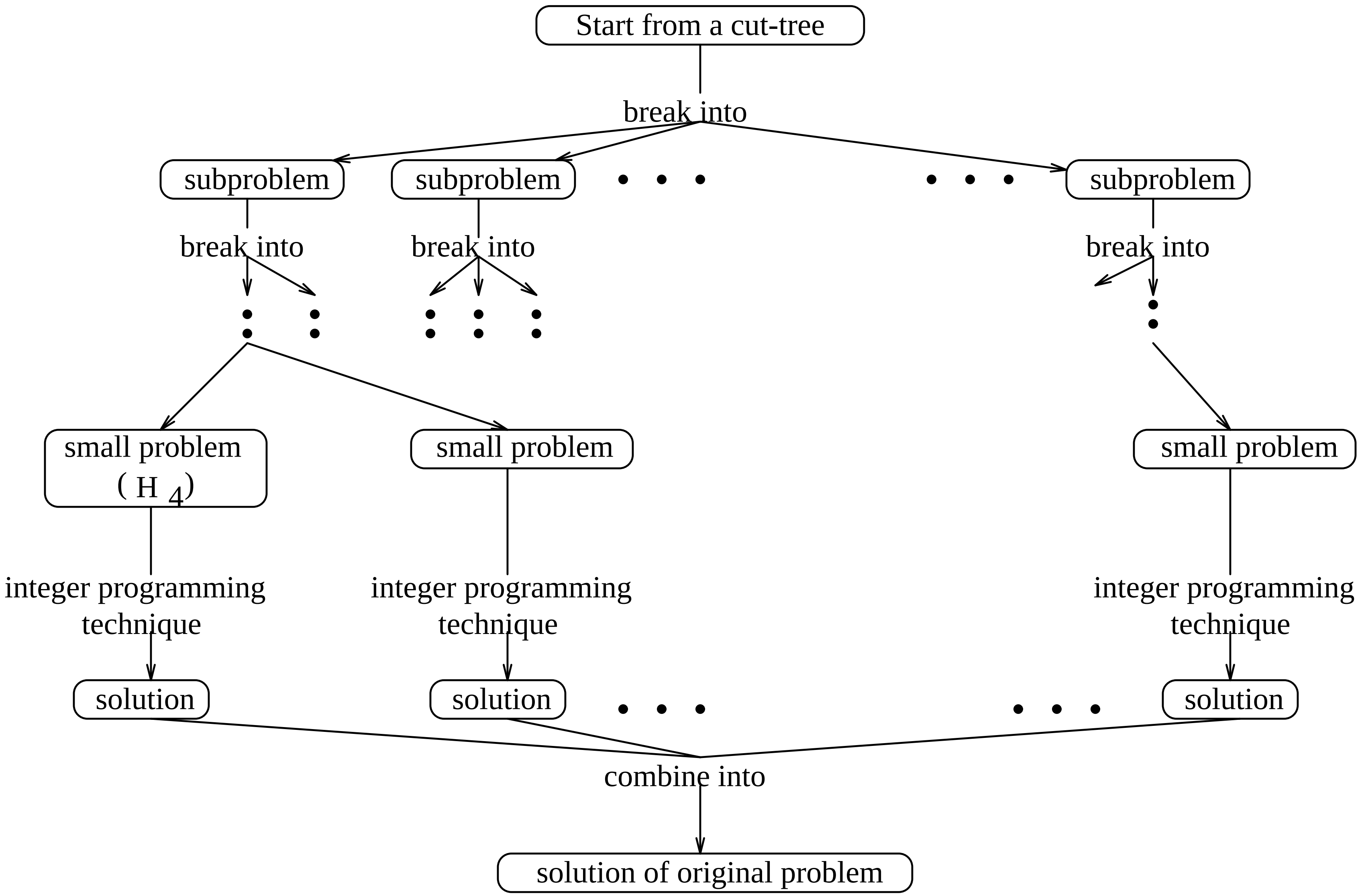
A floorplan and its preprocessed cut-tree



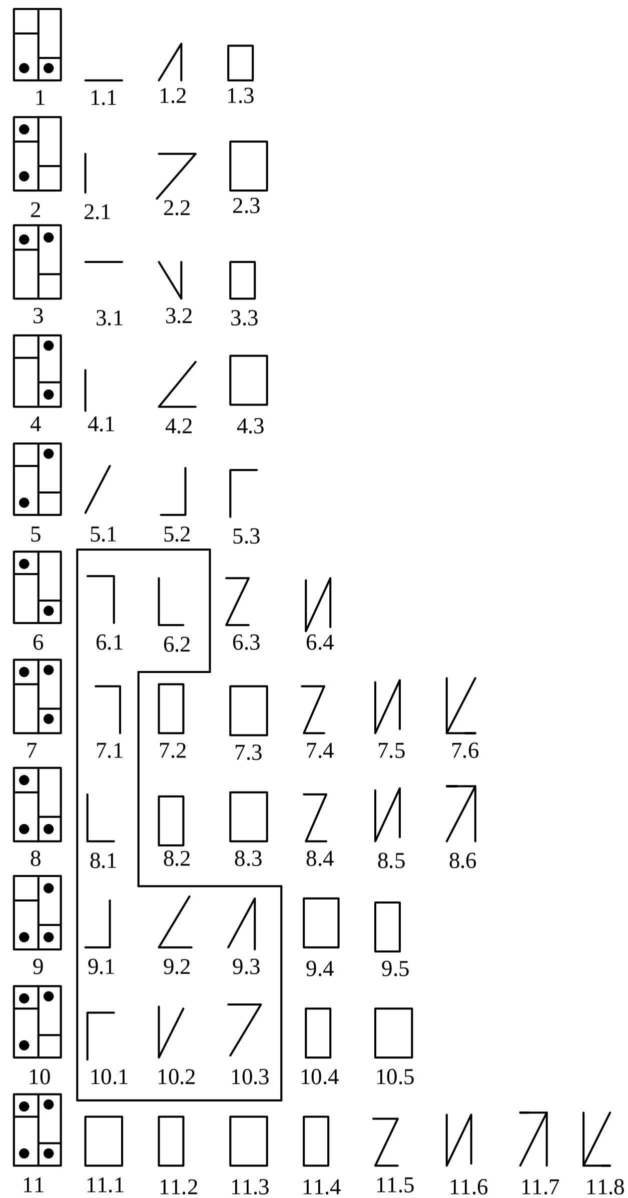
The routing graph H_4 and its floorplan



Hierarchical Integer Programming Approach



All Net Types and Routing Patterns for H_4

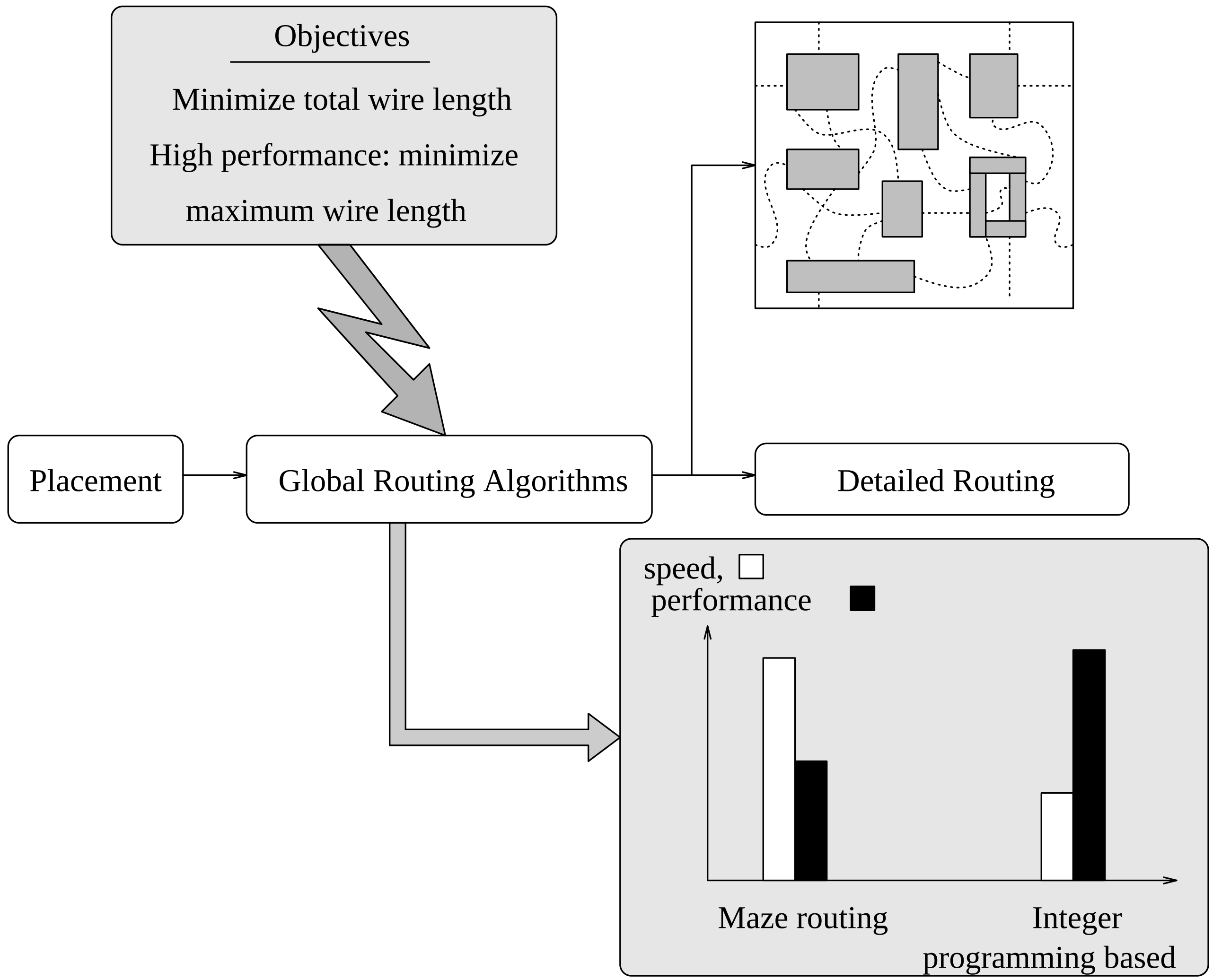


Reduce the Size of Integer Program in H_4

- Three steps:
  1. Routes $\min\{c(e_i), n_i\}$ nets by using T_{i1}.
     Results in a smaller problem $R - 4'$.
  2. Eliminate some redundant routing patterns and reduce the problem into R''_4.
  3. Solve integer program R''_4 which is much smaller than R_4.

J. Heisterman and T. Lengauer *TCAD* 1991

Summary



Objectives
Minimize total wire length
High performance: minimize maximum wire length
Placement
Global Routing Algorithms
Detailed Routing
speed,
performance
Maze routing
Integer programming based