King Fahd University of Petroleum & Minerals
College of Computer Sciences & Engineering
Department of Computer Engineering

GLOBAL ROUTING

Sadiq M. Sait & Habib Youssef

ICM'95 – December 1995

OUTLINE

1. Introduction

2. Cost Functions and Constraints

3. Routing Regions

4. The Steiner Tree Problem

5. Global Routing by Maze Running

6. Global Routing as an Integer Program

7. Global Routing in TimberWolf

8. Other Approaches

Global Routing

- The accepted practice to routing consists of adopting a two-step approach:

  1. Global routing.

  2. Detailed routing.

- The objective of the global routing is to elaborate a routing plan so that each net is assigned to particular routing regions, while attempting to optimize a given objective function.

- Then, detailed routing takes each routing region and, for each net, particular tracks within that region are assigned to that net.

- Global routing is also known as topological routing and loose routing.

- Global routing approaches belong to four general categories:

  (1) sequential approach,

  (2) mathematical programming approach,

  (3) stochastic iterative approach, and

  (4) hierarchical approach.

- **Sequential approach:** Nets are selected one at a time in a specific order and routed individually. If the routing space is updated after the routing of each net, then the approach is order dependent, otherwise it is order independent.

- **Mathematical programming approach:** Global routing is formulated as a 0-1-integer optimization program, where a 0-1 integer variable is assigned to each net and each possible routing tree of that net.

- **Stochastic iterative approach:** Iteratively update current solution by ripping up and rerouting selected nets, until an acceptable assignment of the nets is found.

- **Hierarchical approaches:**

  For the bottom-up approach, grid cells are clustered into bigger cells until the entire chip is seen as a supercell. At each level of the hierarchy, global routing is performed between the individual cells considered for grouping.

  For the top-down approach, the hierarchy proceeds from super cells to cells, until each cell is an individual grid cell or a small group of individual grid cells. The top-down approach is usually guided by the structure of the design floorplan.

Global routing is slightly different for different design styles.

- For the gate-array design style the routing regions consist of horizontal and vertical channels.

  Channels are rectangular regions with pins on the opposite sides of the region. The available routing capacities within the channels are fixed. A feasible global routing solution should not exceed the channel capacities. Among possible feasible solutions, the one that optimizes the given cost function is selected.

  The cost function is usually a function of the global routes of all nets, and/or function of overall performance (interconnect delays on the critical paths).

  Since the array has a fixed size and fixed routing space, the objective of global routing in this case is to check the feasibility of detailed routing.

- For the standard-cell design style the routing regions are horizontal channels with pins at their top and bottom boundaries.

  Global routing consists of assigning nets to these channels so as to minimize channel congestion and overall connection length. Inter-channel routing is provided by feedthrough cells inserted within the cell rows. Here, the channels do not have pre-fixed capacities. Channels can be made wider to achieve routability.

- In building-block design style the cells are of various shapes and sizes. This leads to irregular routing regions. These routing regions may be decomposed into horizontal and vertical channels, and sometimes switchboxes (rectangular regions with pins on all four sides). The identification of these routing regions is a crucial first step to global routing.

  Here again, the routing regions do not have pre-fixed capacities. For both the standard-cell and building-block layout styles the objective of global routing is to minimize the required routing space and overall interconnection length while ensuring the success of the following detailed routing step.

  Therefore the cost function is a measure of the overall routing and chip area. Constraints could be a limit on the maximum number of tracks per channel and/or constraints on performance.

- An important problem we are faced with in all design styles is the identification of the shortest set of routes to connect the pins of individual nets (Steiner tree).

- **Routing Regions:**

  Routing regions definition consists of partitioning the routing area into a set of non-intersecting rectangular regions called channels.

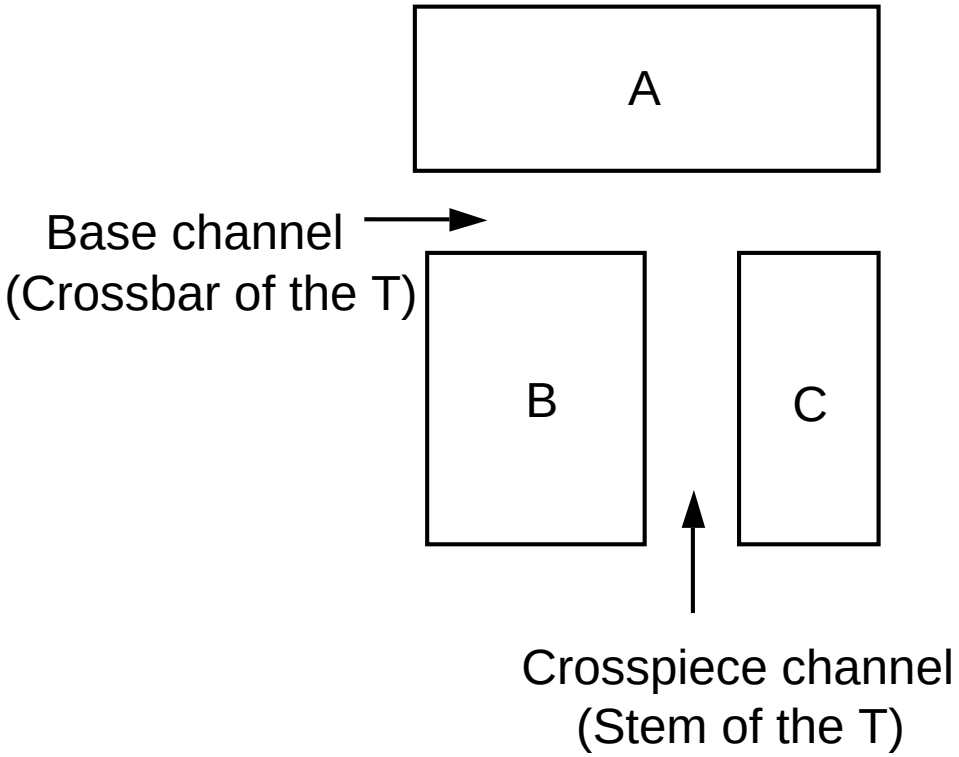
- Two types of channels: horizontal and vertical.

- A channel is horizontal (vertical) if and only if it is parallel to the x- (y-) axis.

- In most cases, horizontal and vertical channels can touch at T-intersections. The channel representing the stem of the T is called the $crosspiece$ and the other is called the $base$.

- Channel definition and ordering is an essential part of layout design. It is the knot that ties placement, global routing, and detailed routing together.

T-intersection



Routing Regions Definition

In a building-block layout style, three types of channel junctions may occur: an L-type, a T-type, and a +-type.

L-type junctions occur at the corners of the layout surface. For such junctions, the ordering does not have an impact on the final detailed routing.

For T-type junctions, the stem channel (crosspiece) must be routed before the base channel (the crossbar).

The +-type junctions are more complex and require the use of switchbox routers. On the other hand, L-type and T-type channels can be completely routed using channel routers.

This is of extreme importance since channel routers are the best and most widely investigated routing approaches.

Therefore it is advantageous to transform all +-type junctions into T-type junctions so that a channel router can be used for the following detailed routing step.

Coversion of +-type junctions
(Cai and Otten, 1989).

Layout is assumed to have a slicing structure (there are polynomial time algorithms to convert a general layout into a slicing structure).

Slicing structures are preferred topologies because they can be internally represented using a simple and flexible data structure (the slicing tree). Moreover, such structures lead to computationally efficient manipulation algorithms.

After converting all cross-junctions into T-channels, the channel ordering constraints are captured by a directed graph called the order constraint graph.

Another positive property of slicing structures is that, a slicing structure is guaranteed to have at least one conflict-free channel structure, i.e., a cycle-free order constraint graph.

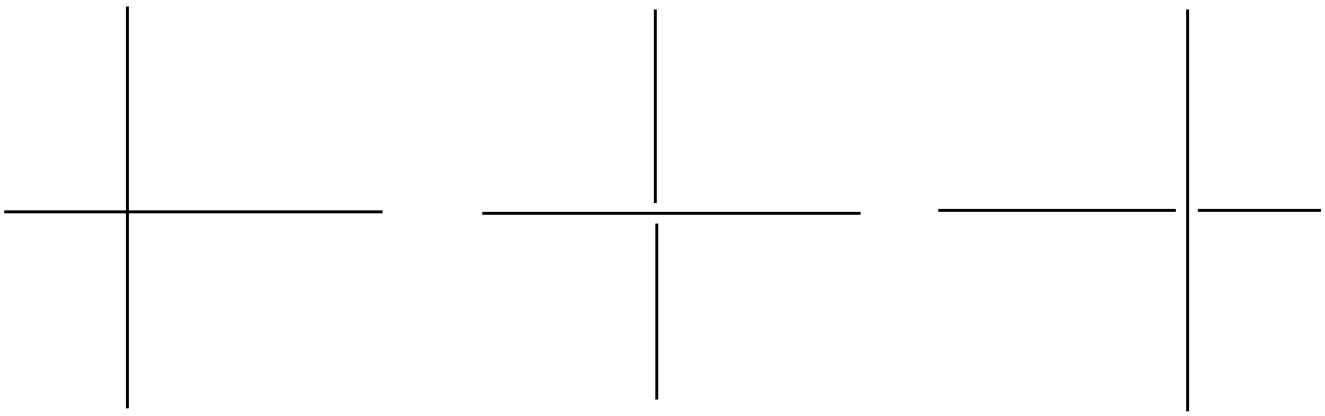
Criteria for Channel Crossing Conversion:
There are two ways of converting a +-junction into a T-junction:

(1) vertical conversion where the horizontal channel is split,

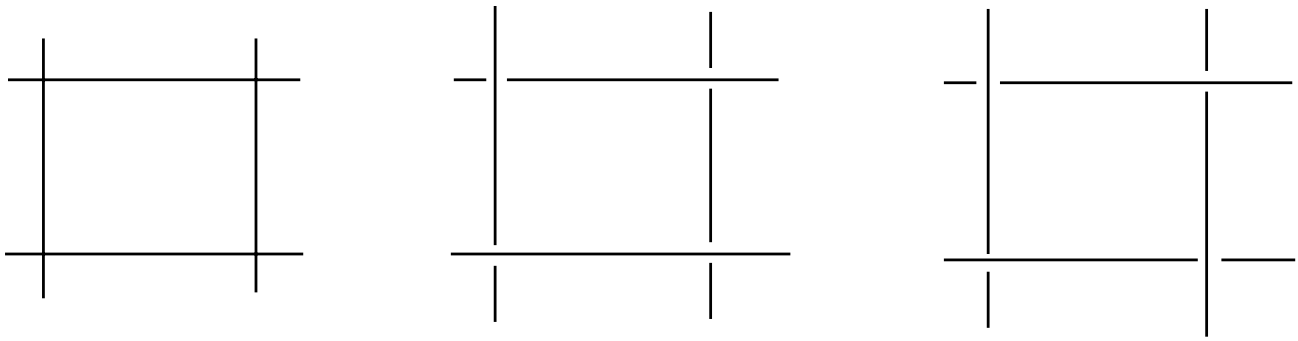
(2) horizontal conversion where the vertical channel is split.

This conversion must be carefully performed so as to avoid creating cycles in the order constraint graph.

Conversion of cross junctions:
(a) Cross junction;
(b) Horizontal conversion;
(c) vertical conversion.

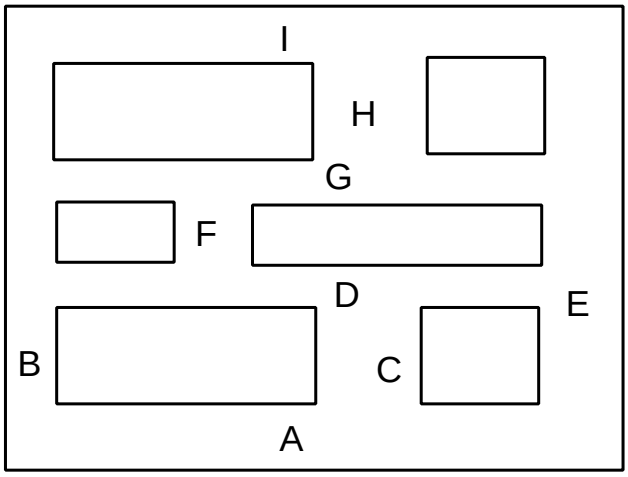


Conversion of cross-intersections:
(a) A channel configuration;
(b) A cycle-free conversion;
(c) A conversion introducing cycles.

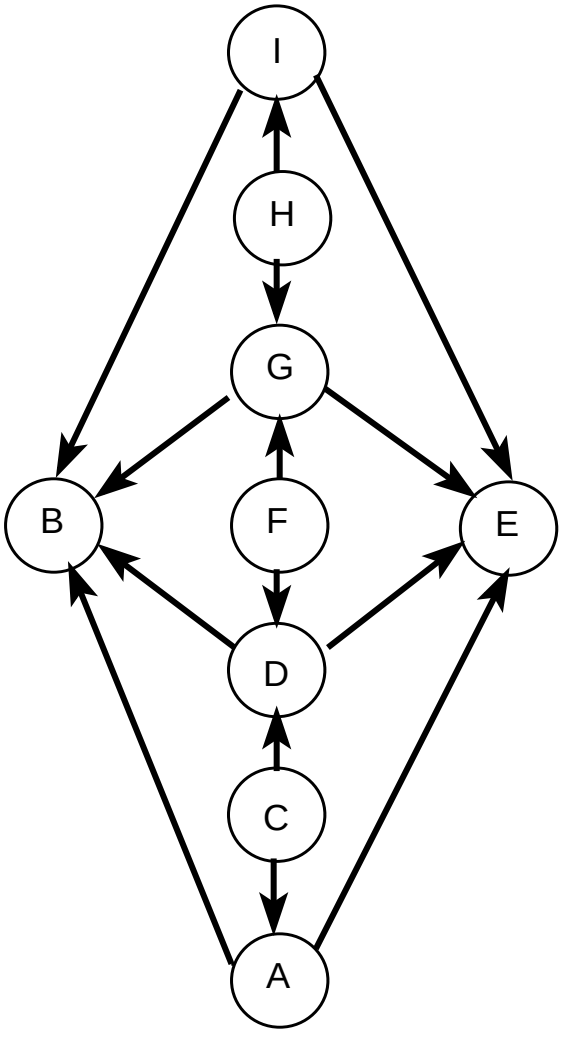


(a) (b) (c)

Order constraint graph:
(a) Channel structure;
(b) Its corresponding order constraint graph.



(a)



(b)

To minimize the negative side-effects that channel conversions might have on the wireability of the layout as a whole, two criteria are used:

(1) **Critical path isolation criterion:**
The objective of this criterion is to protect the critical paths of the channel position graphs from neighboring channels.

Each block-vertex in the horizontal (vertical) channel position graph is assigned a positive weight indicating the width (height) of the corresponding block.

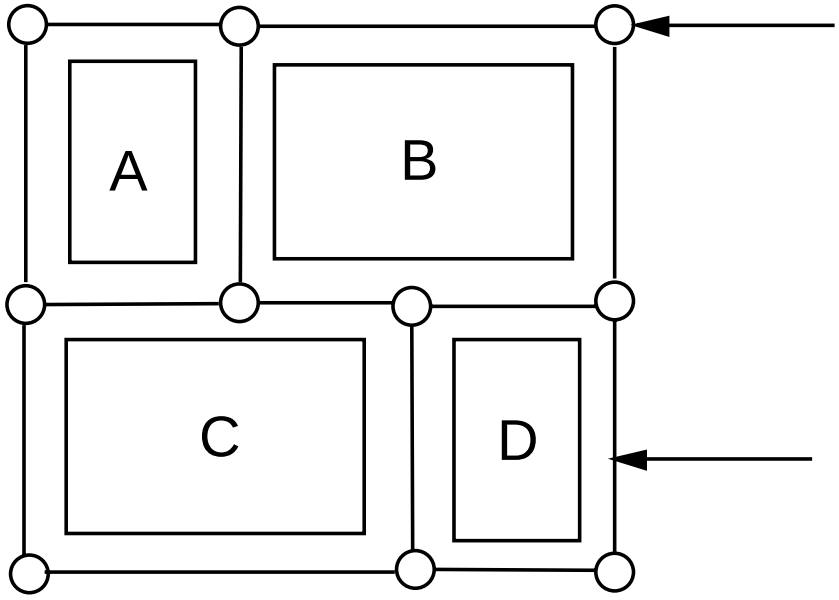
Also, each channel vertex is assigned a positive weight indicating the width of the corresponding channel.

The length of the critical path in the vertical (horizontal) position graph is equal to the height (width) of the design.

The critical path criterion attempts to perform the conversion in the direction of the critical path. This is in order to make neighboring channels perpendicular to the direction of the critical path as short as possible, thus splitting them.

Such criterion will also lead to a reduction in the widths of the channels along the critical paths, thus, reducing the overall layout size.

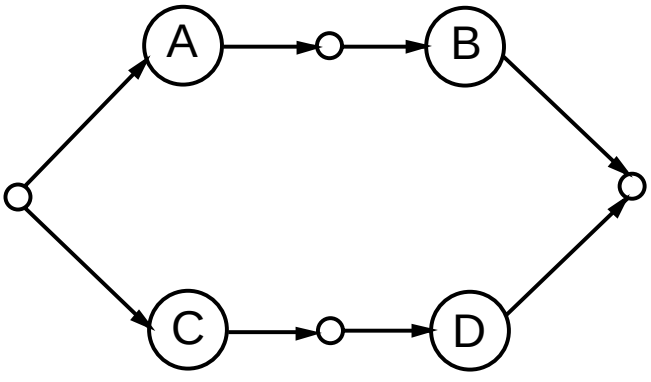
Channel Intersection/Floorplan Graph



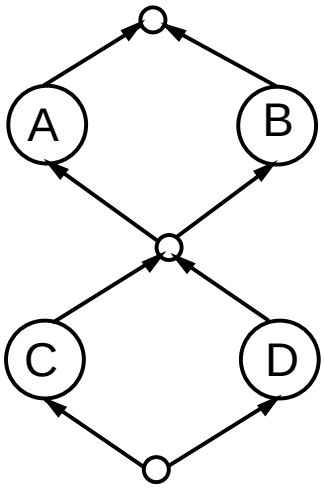
A vertex of the channel intersection graph

An edge of the channel intersection graph

Channel Position Graphs

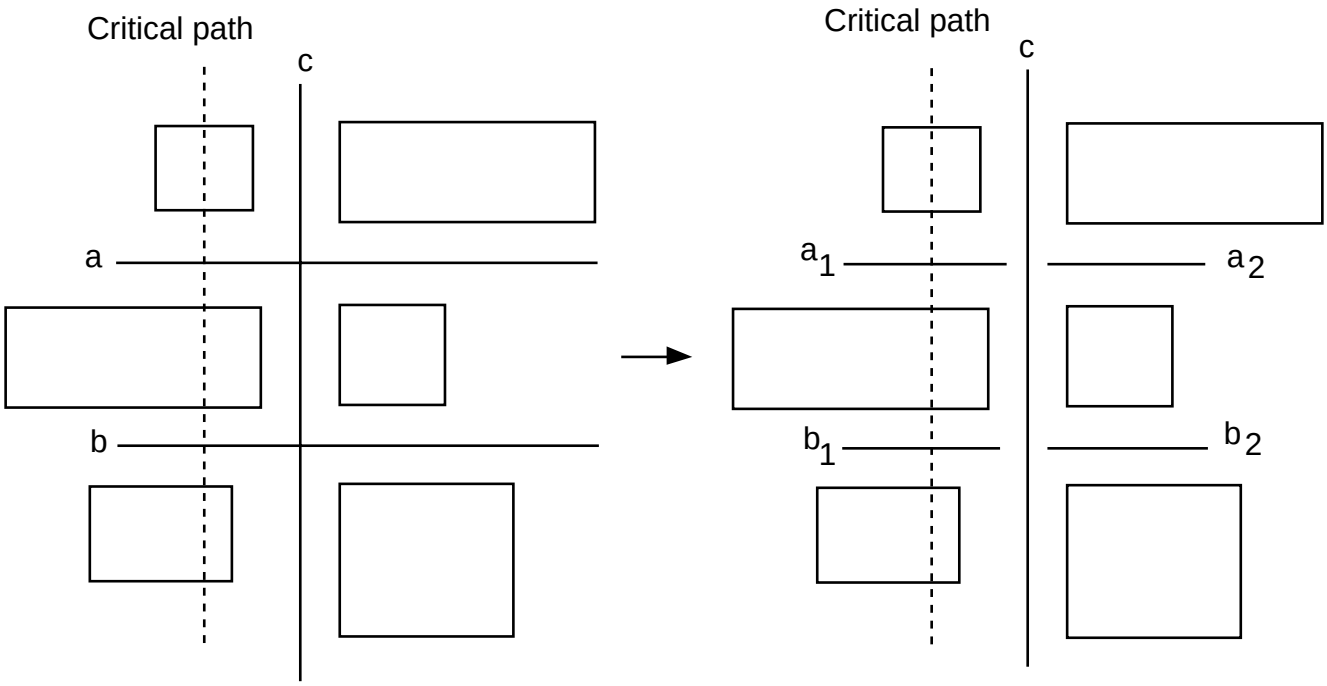


(a)



(b)

Illustration of the critical path criterion

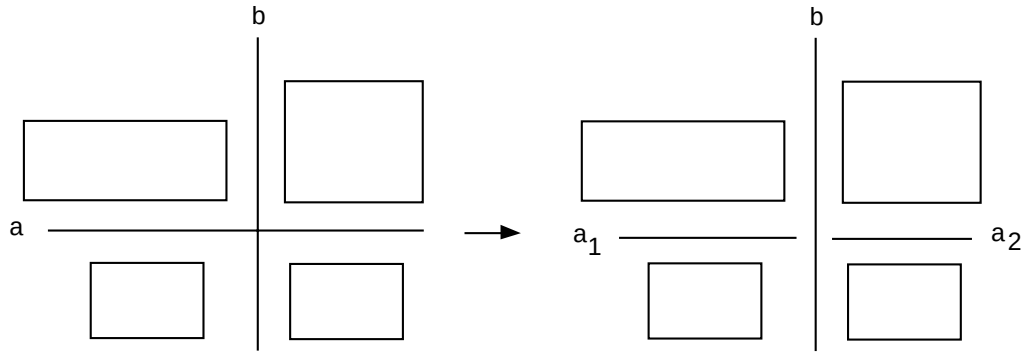


(2) **Major flow criterion:**
Channel conversion is carried out after global routing.

Therefore, the number of wires flowing across all channels is known before the conversion process starts.

In order to minimize the number of wire bends across channels, among the two channels of the cross-junction, we split the thinner of the two, i.e., the channel with the lesser number of flowing nets.

Illustration of the major flow criterion.
(Assume channel b has more nets than a)



For each cross-junction, the previous two criteria are used to compute a positive gain function.

This function is a bonus rewarding conversions of cross-junctions that favor the critical path isolation and major flow criteria.

Therefore, for each cross-junction adjacent to a channel segment that is on some critical path, a bonus is added to the conversion in the direction of the path.

Furthermore, for each cross-junction, a bonus is added to the direction of the channel with the largest wire flow.

For all other cases, a zero bonus is assigned to the crossing conversion in either direction.

The optimal channel structure is the one with the largest sum of crossing conversion bonuses.

Conversion Algorithm

The algorithm assumes that the layout has already been converted to a slicing structure.

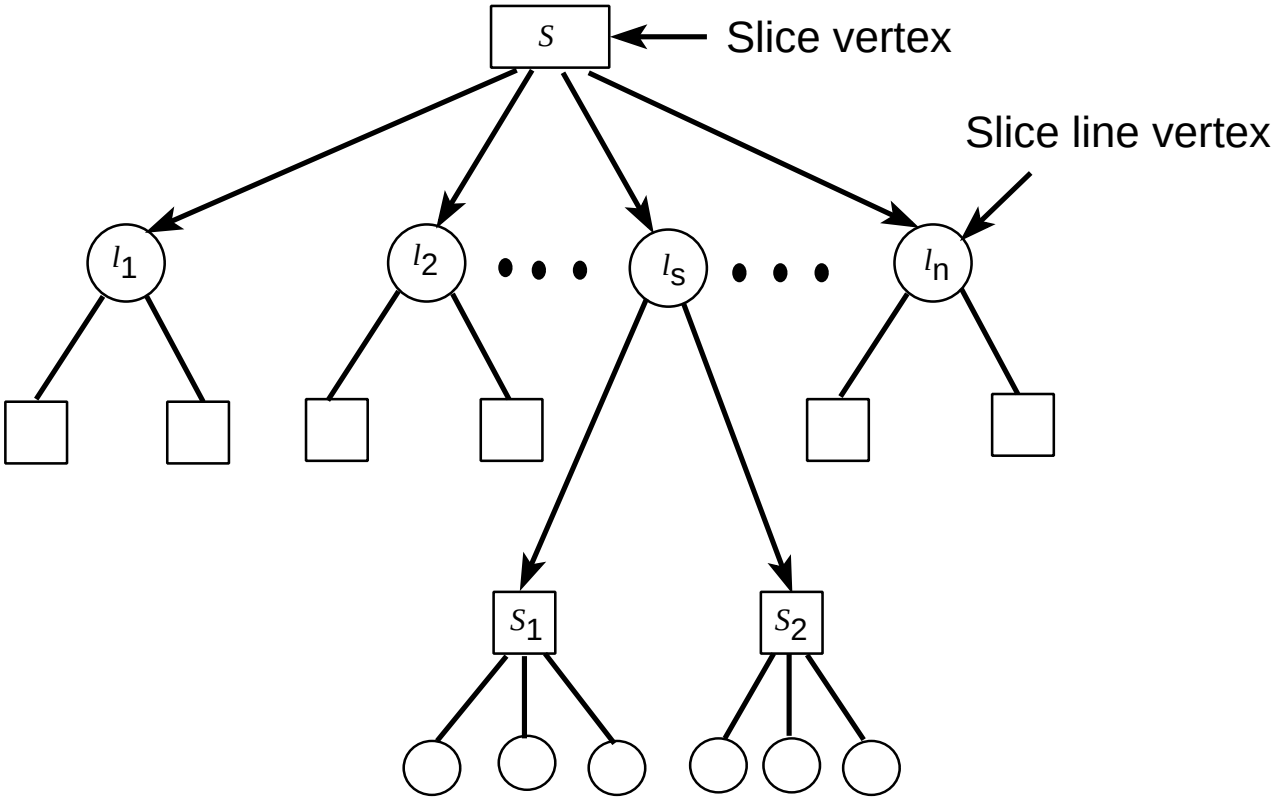
A bipartite directed graph called the slicing graph is constructed.

The slicing graph has a vertex for each candidate slice-line and a vertex for each slice.

There is a directed edge from a slice-vertex to each of its candidate slice-line vertices.

There is a directed edge from a slice-line vertex to the two resulting slice vertices.

A generic slice graph

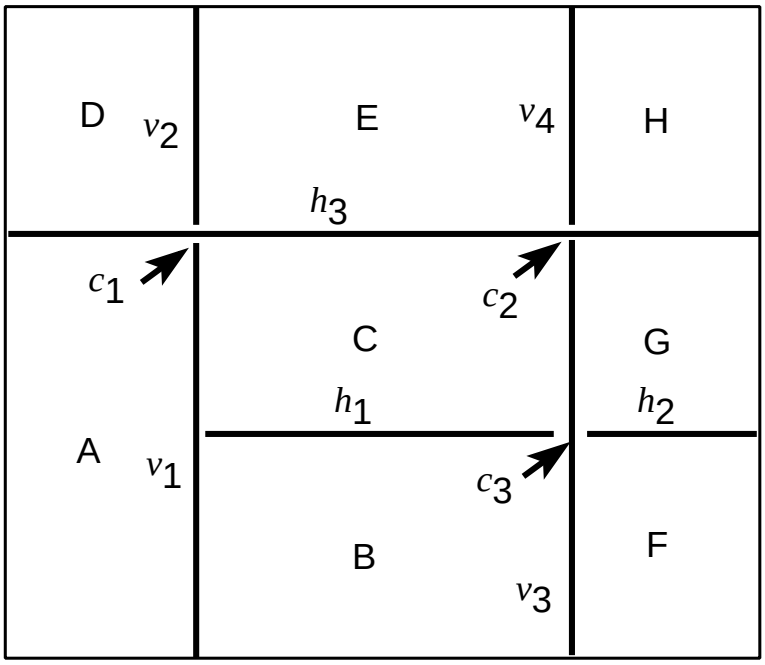


We **need not** have a vertex for each possible slice-line. This is due to what is referred to as the *locality property*.

The preferred conversion direction of a cross-junction is the direction with the maximum bonus.

When both directions have the same bonus, then both are equally good choices, and both should be considered.

Suppose that for a particular slice there is a slice-line which converts all crossings on that line in their preferred directions (for example slice-line h_3 in Figure below). Such a line is an *optimal slice-line*.



Moreover, if an optimal slice line is selected and the resulting two channel definition problems are optimally solved in the two subslices, then the combined solutions constitute an optimal solution to the original slice.

Therefore, to get an optimal solution to the channel definition problem (corresponding to the entire layout), we need to enumerate only trial slice-lines that have the potential of leading to optimal solutions.

This is similar to the dynamic programming algorithm strategy, where the solution to the problem is arrived at as a result of a sequence of decisions.

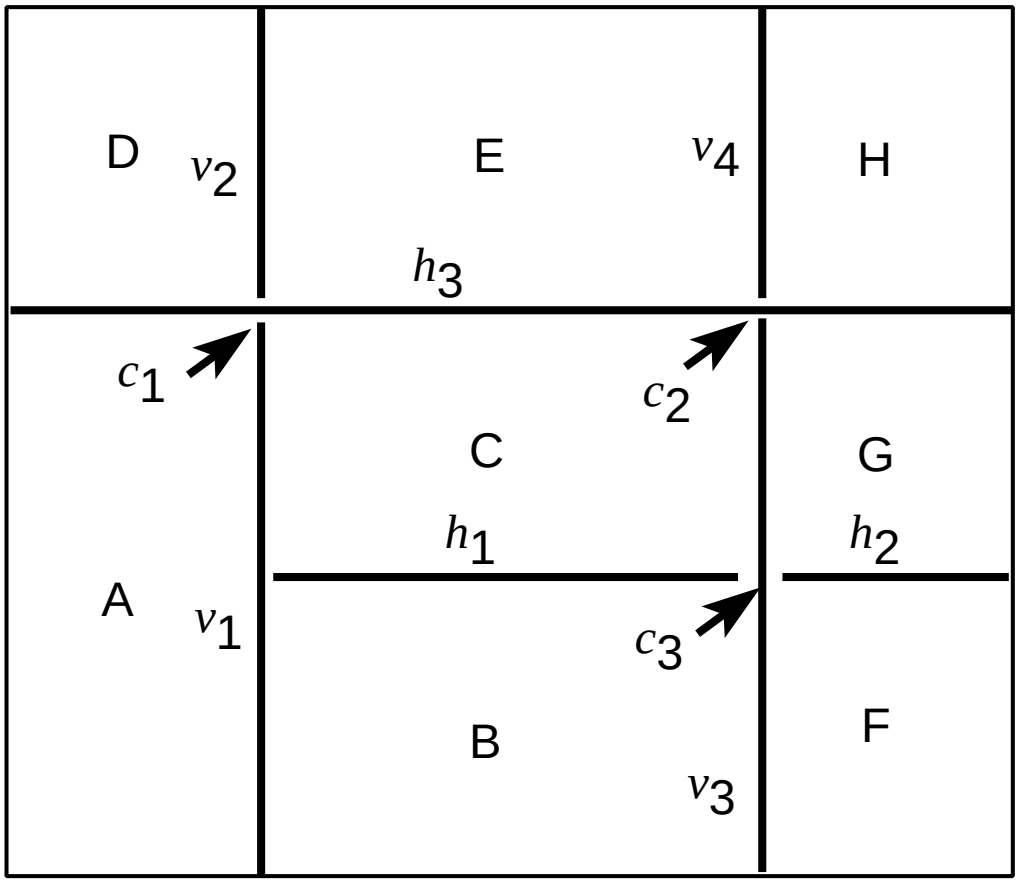
The dynamic programming algorithm is based on the *optimality principle*, which we state next, in the context of the channel conversion problem.

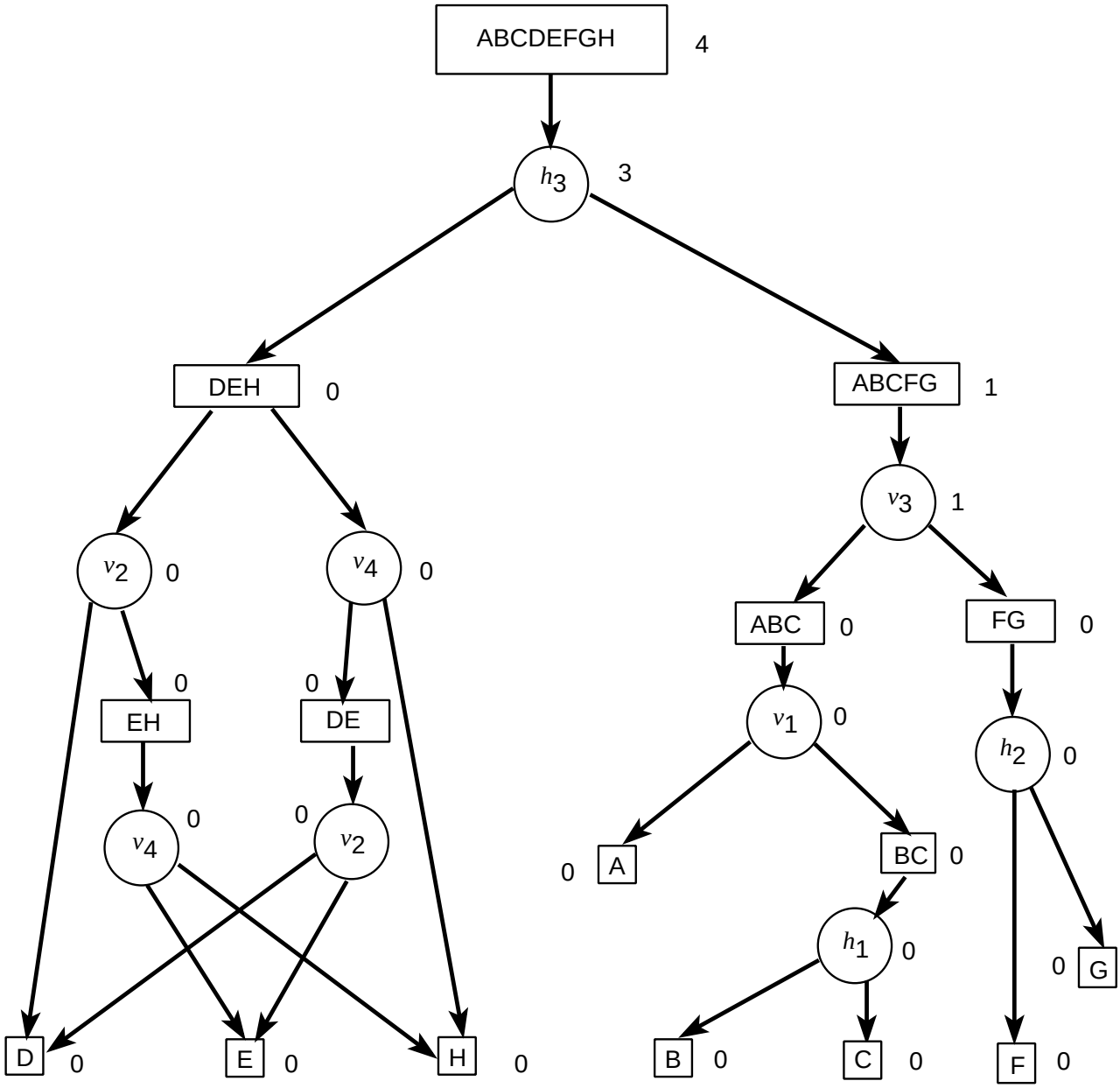
Optimality Principle:
Let $[d_1, d_2, ..., d_i, ..., d_k]$ *be a sequence of decisions with respect to the first slice-line, second slice-line,* $\cdots k^{th}$ *slice-line. If* $d_1 - to - d_k$ *is an optimal sequence of k consecutive decisions, then* $d_1 - to - d_i$ *is an optimal sequence of decisions and* $d_i - to - d_k$ *is also an optimal decision sequence with respect to the state (the subslices) resulting from the* $d_1 - to - d_i$ *decision sequence.*

Therefore, the slice graph should be constructed in a depth first manner, starting at the root (representing the entire layout), and making a decision at each slice vertex as to the slice-lines that should be tried with that particular slice.

The slice graph will have a single source vertex (with no incoming edges) corresponding to the entire layout. The graph will have exactly n sink vertices (with no outgoing edges), where n is equal to the number of layout blocks.

The construction of the slice graph will be illustrated later with the help of an example.


D
v_2
E
v_4
H
h_3
c_1
c_2
C
G
h_1
h_2
A
v_1
c_3
B
F
v_3


ABCDEFGH
4
h_3
3
DEH
0
ABCFG
1
v_3
1
v_2
0
v_4
0
ABC
0
FG
0
0
EH
0
DE
0
v_1
0
h_2
0
v_4
0
v_2
0
0
A
BC
0
h_1
0
0
G
D
0
E
0
H
0
B
0
C
0
F
0

Once the slice graph has been constructed, the algorithm proceeds with a breadth first traversal of the graph, from the sinks to the source. During the course of this traversal, the maximum bonus of each slice vertex is computed. The maximum bonus corresponding to a slice s is defined recursively as follows:

$$Bonus(s) = \max_{l_s}\{Bonus(l_s)+Bonus(s_1)+Bonus(s_2)\}$$

where s_1 and s_2 are the subslices resulting from cutting slice s with slice-line l_s.

Once we reach the source vertex, the channels that lead to the maximum bonuses are identified by tracing back the slice graph from the source to the sinks. The steps of the algorithm are summarized next.

Algorithm Channel_Conversion;
Begin

1. Determine the conversion bonuses of all channel crossings;
2. Determine the bonus of each slice-line;
3. Construct the slicing graph;
4. **For** each sink vertex v **Do**
   $Bonus(v) \leftarrow 0$
   EndFor;
5. Traverse the graph sinks-to-source, computing along the way the maximum bonus of each slice vertex;
6. Traverse the graph source-to-sinks, selecting along the way the slice-lines that incurred maximum bonuses;
7. Output the selected slice-lines;

End.

Example:

Suppose we are given the slicing floorplan of previous Figure.

The floorplan has eight blocks identified with the letters A to H, and three cross-junctions c_1, c_2, and c_3.

Let ${c_i}^h$ and ${c_i}^v$ denote the horizontal and vertical conversions of cross-junction c_i, $i = 1, 2, 3$. c_i is said to be vertically (horizontally) converted if the horizontal (vertical) channel is split into left and right (top and bottom) sub-channels.

Assume that the three cross-junctions have the following bonuses:

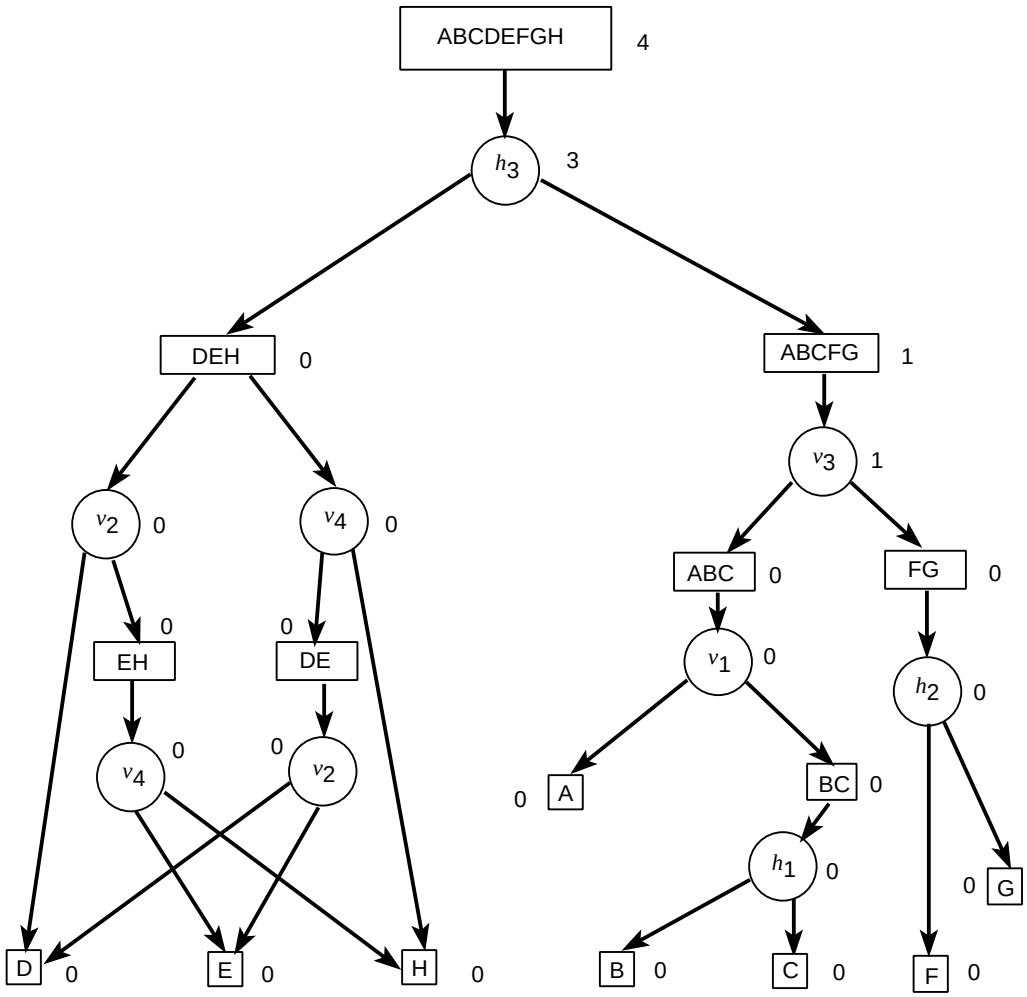
$$Bonus(c_1^v) = 0; \qquad Bonus(c_1^h) = 2;$$
$$Bonus(c_2^v) = 0; \qquad Bonus(c_2^h) = 1;$$
$$Bonus(c_3^v) = 1; \qquad Bonus(c_3^h) = 0;$$

We would like to identify the channel structure corresponding to a maximum bonus conversion of all three cross junctions.

Solution:

The slicing structure of previous Figure assumes that the crossings are initially converted in their preferred directions.

The slicing graph corresponding to this floorplan is given in Figure below, where the slice vertices are represented by boxes and the slice-lines by circles.



The number next to each vertex is the maximum bonus corresponding to the slice vertex/line. The sinks (vertices A to H) are assigned zero bonuses.

The graph is constructed as follows.

Starting at the source vertex $ABCDEFGH$, we find that there is only one optimal slice-line h_3. Line h_3 converts the cross junctions c_1 and c_2 in their preferred directions.

On the other hand, the line $v_{1,2} = v_1 \cup v_2$ is not optimal since it does not convert the cross-junction c_1 into its preferred direction.

The subslices resulting from cutting the original floorplan with slice-line h_3 are [DEH] and [ABCFG].

For slice [DEH], there are two slice-lines of equal merit (both have a zero bonus).

Therefore, both should be included in the slice graph.

However, the other subslice [ABCFG] has only one optimal slice-line v_3, with a bonus equal to 1.

The other slice-line, v_1, is not optimal, and therefore is not included in the graph.

This process is continued until the entire graph is constructed.

Now that the slice graph has been constructed, we first determine the bonuses of all slice-line vertices.

The bonus of a horizontal (vertical) slice-line is equal to the sum of all horizontal (vertical) conversion bonuses of the cross-junctions traversed by that line.

Therefore, for our example, the slice-line bonuses will be,

$$
\begin{aligned}
Bonus(h_1) &= Bonus(h_2) = 0; \\
Bonus(h_3) &= Bonus(c_1^h) + Bonus(c_2^h) = 3; \\
Bonus(v_1) &= Bonus(v_2) = Bonus(v_4) = 0; \\
Bonus(v_3) &= Bonus(c_3^v) = 1.
\end{aligned}
$$

The next step is to traverse the slice graph from the sinks to the source, and compute the maximum bonuses of the intermediate slice vertices. Proceeding this way, we get,

$$Bonus(EH) = Bonus(v_2) + Bonus(E) + Bonus(H) = 0;$$

$$Bonus(DE) = Bonus(v_4) + Bonus(D) + Bonus(E) = 0;$$

$$Bonus(DEH) = \max\{ Bonus(v_2) + Bonus(D) + Bonus(EH); Bonus(v_4) + Bonus(DE) + Bonus(H)\} = 0.$$

The maximum bonuses of the remaining slice vertices are computed in a similar way and are as follows,

$Bonus(BC) = 0$, $Bonus(FG) = 0$,
$Bonus(ABC) = 0$, $Bonus(ABCDFG) = 1$,
and $Bonus(ABCDEFGH) = 4$;

The last step of the algorithm is to proceed from the source to the sinks in order to determine the slice-lines that contributed to the maximum bonuses.

For this example, h_3 is selected first.

Then proceeding down to subslice DEH, we have two equally good choices, either line v_3 or line v_4. Suppose we randomly select slice-line v_4.

Then, the remaining slice lines are v_2, v_3, v_1, h_1, and finally h_2. Hence, the initial slicing structure is the optimal one.

Channel Ordering:

Once all nets have been assigned to individual channels, the final step is to assign the nets to individual tracks within every channel, i.e., to perform detailed routing.

The channels are usually routed one at a time in a specific order.

Channel ordering is an important intermediate step executed prior to detailed routing and after global routing.

This step is needed to specify to the detailed router which channel to route first, which second, and which last.

Obviously, it is assumed that all routing regions are channels.

Channel ordering is an important final step of global routing. The order in which channels should be routed is dictated by the fact that pin locations must be fixed before performing detailed routing of that channel.

Of the two channels of a T-intersection, the crosspiece channel must be routed before the base channel.

This is for the following two reasons:

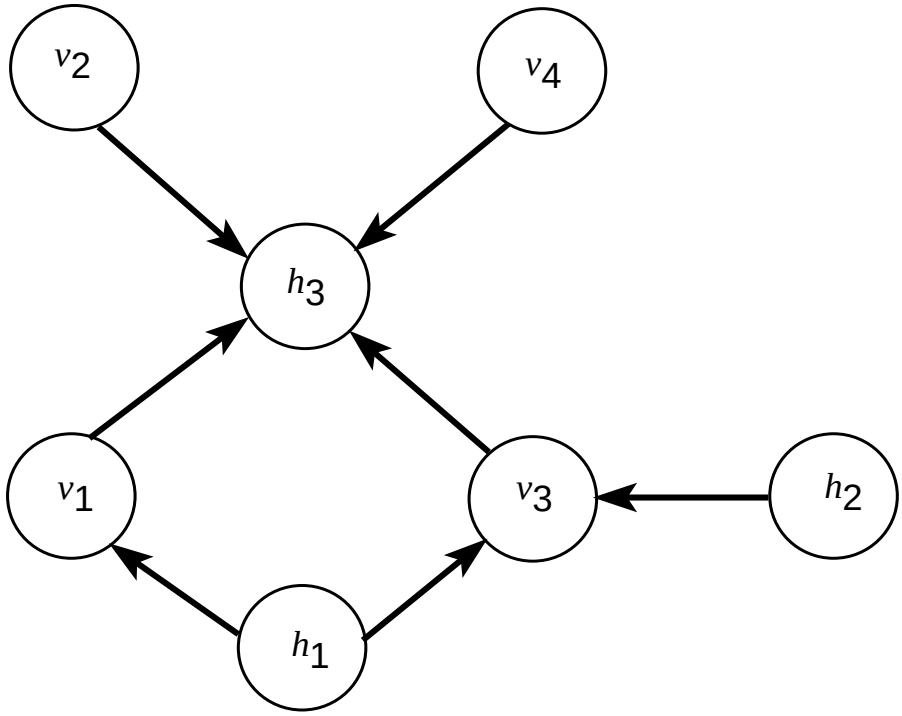
(1) To route the base channel, we need the pin information at the T-junction, i.e., the nets going through the junction. This necessitates that the crosspiece be routed before the base.

(2) When routing the crosspiece channel, we may realize that we need to move blocks at the left (top) and/or right (bottom) of that channel to provide for extra tracks. This will change the pin positions within the base channel. This is another compelling reason to route the crosspiece channel before the base.

To order the channels, an order constraint graph (OCG) is built as follows.

Each channel is represented by a vertex. There is an arc (i, j) in the OCG if and only if channels i and j touch at a T-junction of which i is the crosspiece and j is the base.

The corresponding order constraint graph is the following.



Notice that this constraint graph is cycle free.

Routing Regions Representation

Once the routing regions have been defined, a routing graph is constructed. There are three general approaches to construct this graph.

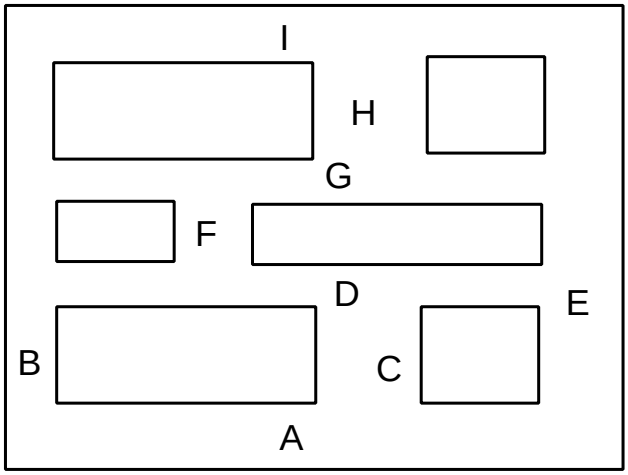
(1) Use a channel connectivity graph $G = (V, E)$ where each channel is represented by a vertex.

Each edge models the adjacency between the corresponding channels.

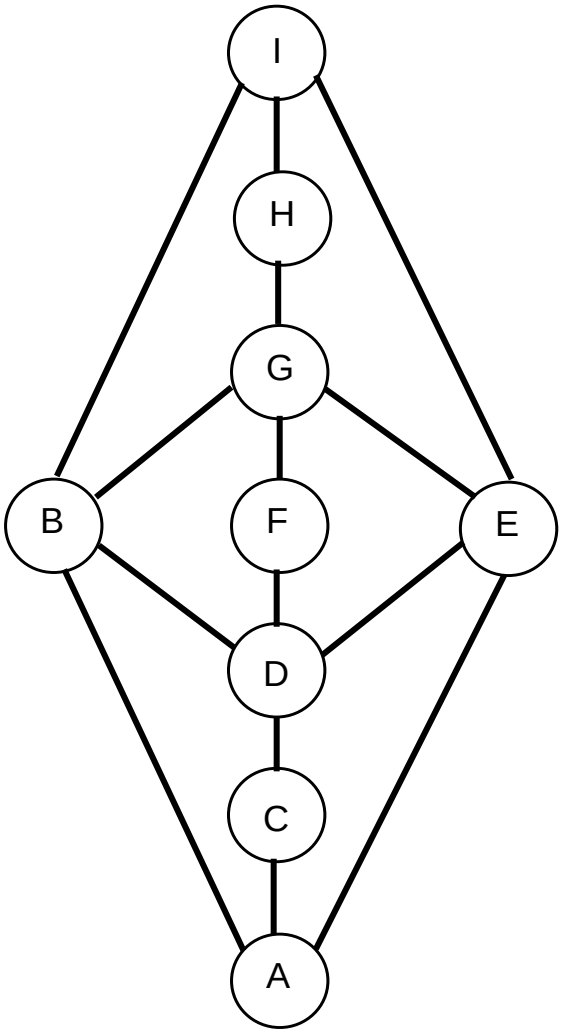
Vertices can be assigned weights to indicate the number of nets passing through the channel and/or the number of available tracks in that channel.

Notice that the channel connectivity graph is the order constraint graph when arc directions are removed.

(a) A building-block layout;
(b) Corresponding channel connectivity graph.



(a)

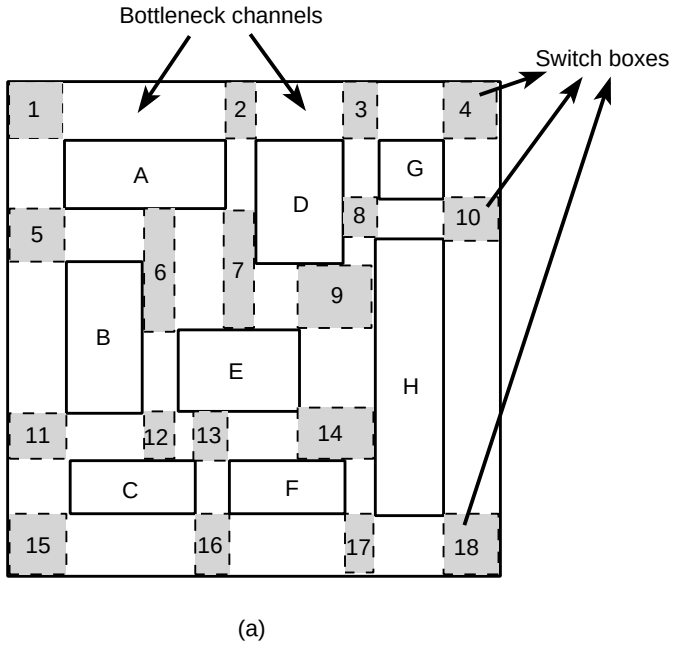


(b)

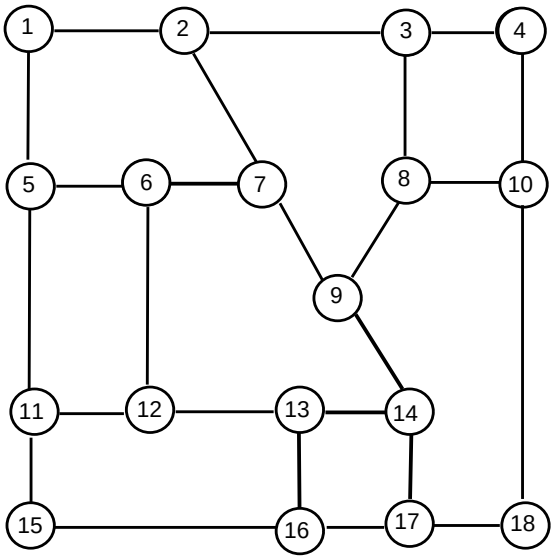
(2) Use a bottleneck graph $G = (V, E)$, where only switchboxes are modeled by vertices.

There is an edge $(u, v) \in E$ if and only if the corresponding switchboxes are on opposite sides of the same vertical or horizontal channel. These routing channels are called bottllenecks, hence the name of the graph.

(a) A building-block layout;
(b) Corresponding bottleneck graph.



(a)

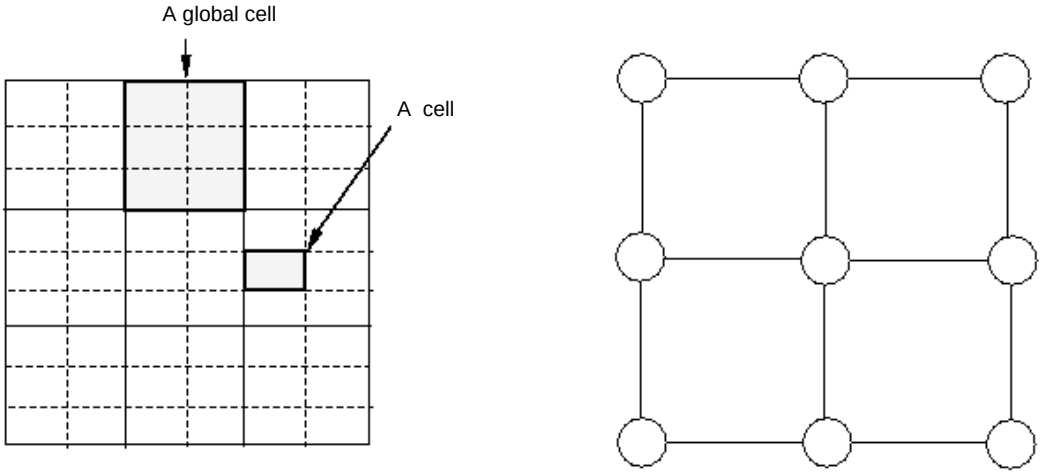


(b)

(3) Use a grid graph $G = (V, E)$ where vertices model global cells and edges adjacencies between these cells.

For two layer-routing, each vertex is assigned two numbers indicating the number of available horizontal and vertical tracks.

(a) A two-dimensional grid; (b) Corresponding grid graph where each global cell (3×2 grid cells) is modeled by a vertex.



Sequential Global Routing.

Sequential global routing is the most widely used approach. It is graph based.

Once the routing channels have been identified and the corresponding routing graph constructed, global routing proceeds as follows.

For each net, we mark the vertices of the channel connectivity graph in which the net has pins.

Hence, routing the net amounts to identifying a tree (preferably the shortest) covering those marked vertices.

If the net has pins in only two vertices, the problem reduces to finding the shortest path between the marked vertices.

If the graph is a grid-graph, we can use Lee algorithm,

For all three graph models, we can use Dijkstra shortest path algorithm.

Algorithm Shortest_Path(s, G)
(* s: a source vertex, and G is a weighted graph *)
(* D_i: shortest distance from s to node i; *)
(* d_{ij}: weight of edge (i, j); *)
(* M: set of permanently marked nodes. *)
Begin
1. (* Initialization *)
$M \leftarrow s$;
$D_s \leftarrow \emptyset$;
ForEach $j \in V(G)$ **Do** $D_j \leftarrow d_{sj}$;
2. (* Find the next closest node *)
Find a node $i \notin M$ such that $D_i = \min_{j \notin M} D_j$;
$M \leftarrow M \cup \{i\}$;
If M contains all nodes then $STOP$;
3. (* Update markings *)
ForEach $j \notin M$ **Do** $D_j \leftarrow \min_i(D_j; D_i + d_{ij})$;
Goto 2;
End.

However, in general, nets have three or more pins. Finding the shortest paths covering three or more nodes is known as the steiner tree problem.

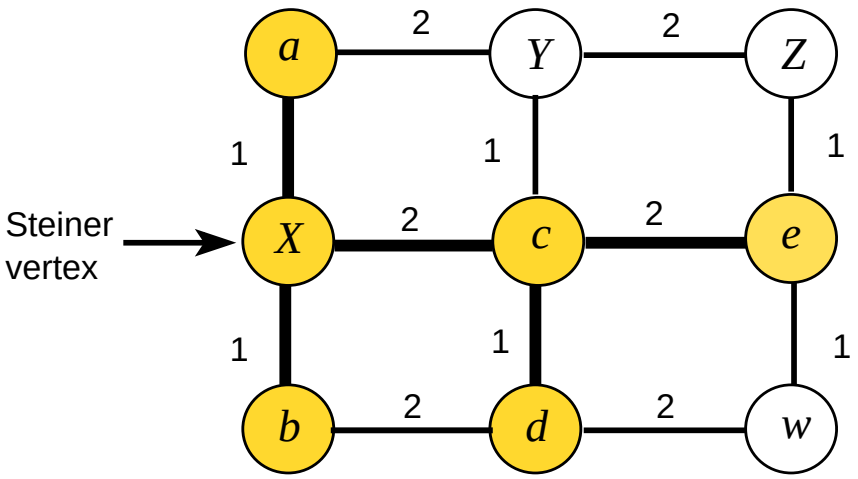
This problem is of crucial importance to global routing and is the subject of the following subsection.

The Steiner Tree Problem:

Let M be the set of marked vertices. A tree connecting all vertices of M as well as other vertices of G that are not in M is called a steiner tree.

A minimum steiner tree is a steiner tree with minimum length.

Steiner tree corresponding to the net $M = \{a, b, c, d, e\}$



The steiner tree problem is NP-hard. Therefore, instead of finding a minimum steiner tree, heuristics are used to identify as quickly as possible a tree of reasonable length not necessarily of minimum length.

Most steiner tree heuristics use a modification of minimum shortest path algorithm of Dijkstra or a variation of Lee's maze routing algorithm.

Usually the heuristic proceeds in a greedy fashion as follows.

First, one of the marked vertices is selected.

Then the shortest path to any one of the remaining marked vertices is identified.

Then, one of the remaining marked vertices is picked and a shortest path from that node to any of the nodes of the partial tree is identified.

This process continues until all marked vertices have been processed.

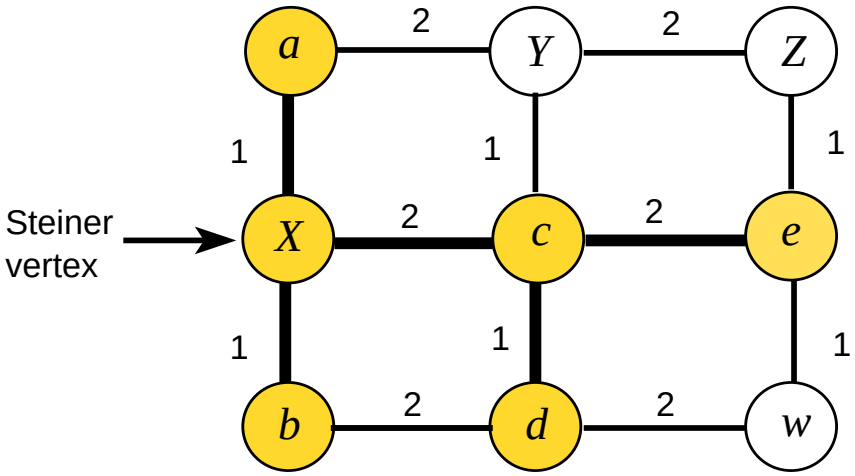
Algorithm Steiner_Tree;
Begin
$M \leftarrow$ set of marked nodes;
(* nodes where the net has pins *)
$s \leftarrow$ select a node from M;
$M \leftarrow M - \{s\}$;
Apply Dijkstra_algorithm to find $\pi_{s,e}$,
the shortest path from s to some node e of M;
$M \leftarrow M - \{e\}$;
$V \leftarrow \{s, e\}$; (* nodes of Steiner tree *)

While $M \neq \phi$ **Do**
Begin
$e \leftarrow next(M)$; (* get next node from M *)
Apply Dijkstra_algorithm to find $\pi_{e,x}$,
the shortest path from e to some node $x \in V$;
$V(\pi_{e,x}) \leftarrow$ nodes covered by $\pi_{e,x}$;
$V \leftarrow V \cup V(\pi_{e,x})$;
(* remove marked nodes covered by $\pi_{e,x}$).
$M \leftarrow M - M \cap V(\pi_{e,x})$;
EndWhile
End.

Example:

Apply this Steiner tree heuristic on the following graph.

Steiner tree corresponding to the net $M = \{a, b, c, d, e\}$



Solution:

The vertices of the graph are,
$V = \{a, b, c, d, e, W, X, Y, Z\}$.

Initially $V = \phi$ and the set of marked vertices
$M = \{a, b, c, d, e\}$.

Suppose vertex a is selected first. The shortest path to any of the remaining marked vertices is $\pi_{a,b} = [a, X, b]$.

Therefore, the sets V and M become, $V = \{a, X, b\}$, and $M = \{c, d, e\}$.

Assume that vertex c is selected next. The shortest path from c to any of the vertices of V is $\pi_{c,X} = [c, X]$.

Therefore, the sets V and M become, $V = \{a, X, b, c\}$, and $M = \{d, e\}$.

Assume that vertex d is selected next. The shortest path from d to any of the vertices of V is $\pi_{d,c} = [d, c]$.

Therefore, the sets V and M become,
$V = \{a, X, b, c, d\}$, and $M = \{e\}$.

Finally vertex e is selected last. The shortest path from e to any of the vertices of V is $\pi_{e,c} = [e, c]$. Therefore, the sets V and M become,
$V = \{a, X, b, c, d, e\}$ and $M = \phi$.

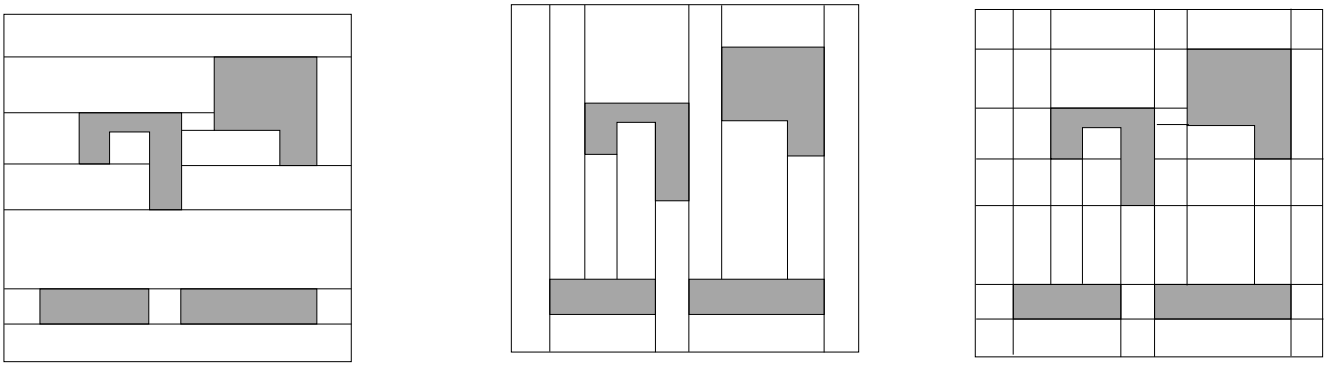
Hence, the steiner tree identified by the algorithm has the following edges,
(a, X), (X, b), (X, c), (c, d), and (c, e).

The weight of the tree is equal to 7.

Such a tree can be improved using rip-up and rerouting.

Global Routing by Maze running

First identify the routing regions.



(a) (b) (c)

Two dimensional routing model. Shaded spaces indicate cells. The unshaded spaces are routing areas. (a) Horizontal routing areas. (b) Vertical routing areas. (c) Routing regions model.

Once the routing channels (regions) have been identified, the task now is to assign nets to them.

To accomplish this, the channels are modeled by a weighted undirected graph called *channel connectivity graph*.

Nodes of the graph correspond to channels and edges between nodes indicate that the corresponding channels are adjacent.

For two layer routing, each node is assigned two weights giving the horizontal capacity (width) and the vertical capacity (length) of the corresponding channel.

The sequential approach is the simplest and most widely used approach to global routing.

This approach consists of picking one net at a time and finding an optimal (or sub-optimal) steiner tree which covers all the pins of the net.

Two general approaches are possible in this case:

(1) Order dependent approach.
For order independent global routing, each net is routed independently of all other nets.

Then congested passages are identified and the affected nets are rerouted, while penalizing paths going through such passages.

This approach avoids net ordering and considerably reduces the complexity of the search space since the only obstacles are the cells.

However, this might require a large number of iterations before a feasible global routing solution is found.

(2) Order independent approach.
For order dependent global routing, first the nets are ordered according to some criteria.

Then the nets are routed in the resulting order, while updating the available routing space after each net.

The search is slightly more complex, since the number of obstacles has increased (cells and already routed nets).

Furthermore, net ordering is crucial to the final outcome of the global routing step. Both approaches are somewhat similar, in that, they try to identify a steiner tree for one net at a time.

In the remainder of this section, we shall be describing the order dependent approach only.

Each side of a block is unambiguously attached to a unique channel. Hence, each pin is uniquely associated with a channel.

Therefore, the nodes of the channel connectivity graph in which a net has pins are unambiguously determined.

Then globally routing a net amounts to determining a tree that covers all those nodes in which the net has pins.

To illustrate the search process, we shall focus on the easy case when the net has pins in only two nodes.

The search procedure is similar to the one used in Lee algorithm.

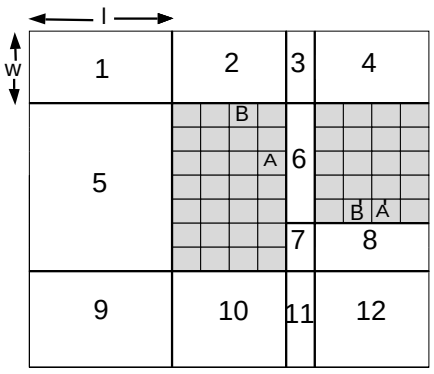
For simplicity, we assume that the length of all edges in the channel-graph is one unit.

As in Lee algorithm, starting from a node labeled k, all adjacent nodes are labeled $k + 1$.

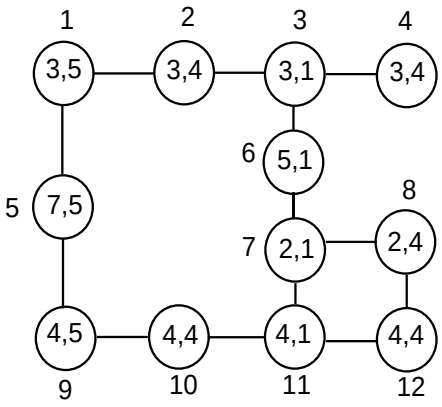
The labeling procedure continues until the target node is reached.

The shortest path in the channel graph is found by a sequence of nodes with decreasing labels.

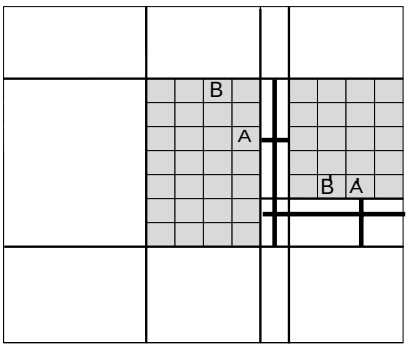
Once the path is found, the net is assigned to the channels and for all nodes (channels) in the path the capacity weights w and l are decreased according to the width and length of the net to be routed.


l
w
1
2
3
4
5
6
7
8
9
10
11
12
B
A
B
A

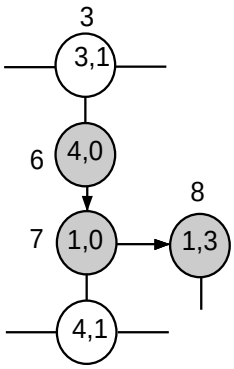
(a)


1
2
3
4
3,5
3,4
3,1
3,4
6
5,1
5
7,5
8
7
2,1
2,4
4,5
4,4
4,1
4,4
9
10
11
12

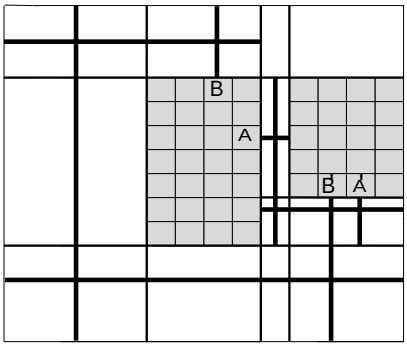
(b)


B
A
B
A

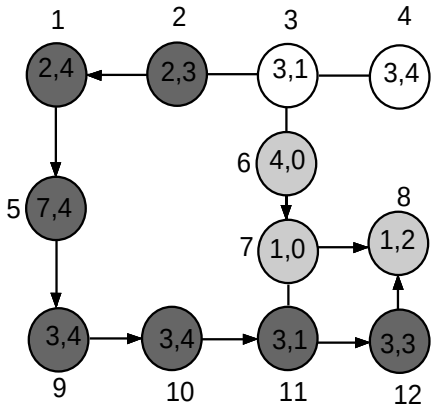
(c)


3
3,1
6
4,0
8
7
1,0
1,3
4,1

(d)


B
A
B
A

(e)


1
2
3
4
2,4
2,3
3,1
3,4
6
4,0
5
7,4
8
7
1,0
1,2
3,4
3,4
3,1
3,3
9
10
11
12

(f)

Observe that both the vertical as well as the horizontal weights of a node are updated when the path bends at the node.

The width and length of the channel are reduced by an amount equal to the space used by the wiring segment.

If the net has a third pin located in another vertex of the graph, then the expansion will continue at that third pin and will terminate when any of the three nodes 6, 7, or 8 is reached.

Another application of the above technique is to determine the required separation between cells in order to ensure routability of the chip.

Therefore, the global router is used only to determine the required separation between cells.

The detailed routing will not necessarily follows exactly the channels assigned to nets by the global router.

The modification to the above method to determine separation between cells would be to start with a zero separation, and this will represent the initial weights of the nodes.

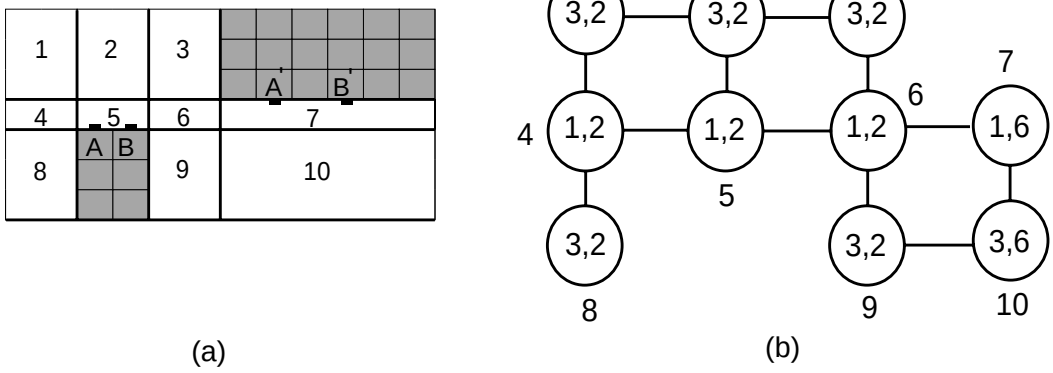
Next, every time a path is found, the weights of the corresponding weights are increased.

At the end of this procedure, the relative placement of the cells is maintained, but the minimum separation between the cells will be as given by the horizontal and vertical weights of the nodes of the channel-graph.

Notice here that the estimations are somewhat pessimistic since they assume that nets do not share routing tracks.

In order to avoid congestion of nets between channels, an upper limit for each node of the channel graph can be set. The global router then will look for alternate paths in the channel-graph.

Example:
A placement containing two cells is shown in Figure below, together with its channel graph. The cells contain two nets $A-A'$ and $B-B'$. Determine if the circuit is routable.



The weights in the nodes indicate the width (w) and length (l) of the channel.

Mathematical Programming Approach

- Global routing can be formulated as a 0-1 integer program as follows.

- The layout is modeled as a grid graph, where each node represents a grid cell (super cell).

- The boundary between any two adjacent grid cells l and k is supposed to have a capacity of $c_{l,k}$ tracks.

- This corresponds to a positive weight $c_{l,k}$ on the arc linking nodes l and k in the grid graph.

- For each net i, we need to identify the different ways of routing the net.

- Suppose that for each net i, there are n_i possible trees t_1^i, t_2^i,..., $t_{n_i}^i$, to route the net. Then, for each tree t_j^i, we associate a variable $x_{i,j}$ with the following meaning:

$$x_{i,j} = \begin{cases} 1 & \text{if net } i \text{ is routed according to } t_j^i \\ 0 & otherwise \end{cases} \tag{1}$$

- For each net i, we associate one equation to enforce that only one tree will be selected for that net,

$$\sum_{j=1}^{n_i} x_{i,j} \;=\; 1 \tag{2}$$

- Therefore, for a grid graph with M edges and T trees, we can represent the routing trees of all nets as a 0-1 matrix $A_{M\times T} = [a_{i,p}]$ where,

$$T \;=\; \sum_{i=1}^{N} n_i \tag{3}$$

where N is the number of nets, and,

$$a_{i,p} = \begin{cases} 1 & \text{if edge } i \text{ belongs to tree } t_k^l \\ & \quad \text{and } p \text{ as defined in equation (5).} \\ 0 & otherwise \end{cases} \tag{4}$$

$$p = \sum_{m=1}^{l-1} n_m + k \tag{5}$$

- A second set of equations is required to ensure that the capacity of each arc (boundary) i, $1 \leq i \leq M$, is not exceeded , i.e.,

$$\sum_{k=1}^{N} \sum_{l=1}^{n_k} a_{i,p} \times x_{l,k} \leq c_i \tag{6}$$

Finally, if each tree t_i^j is assigned a cost $g_{i,j}$, then a possible objective function to minimize is,

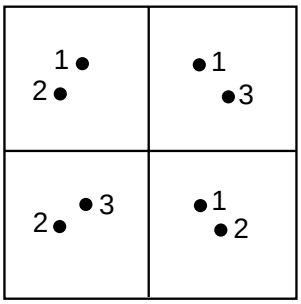
$$F = \sum_{i=1}^{N} \sum_{j=1}^{n_i} g_{i,j} \times x_{i,j} \tag{7}$$

- Therefore, a possible 0-1 integer programming formulation of global routing is,

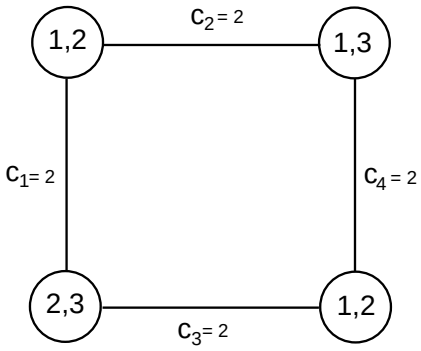
$$\begin{cases} \sum_{i=1}^{N} \sum_{j=1}^{n_i} g_{i,j} x_{i,j} & \leftarrow minimize \\ \text{subject to :} & \\ \sum_{j=1}^{n_i} x_{i,j} = 1 & 1 \leq i \leq N \\ \sum_{k=1}^{N} \sum_{l=1}^{n_k} a_{i,p} x_{l,k} \leq c_i & 1 \leq i \leq M, \\ & \text{and } p \text{ as in (5)} \\ x_{k,j} = 0, 1 & 1 \leq k \leq N \\ & and \;\; 1 \leq j \leq n_k \end{cases} \tag{8}$$

- If $g_{i,j} = g$ for all i and j and if we change the objective to the maximization of F, then this is equivalent to seeking a solution that achieves the maximum number of connections.

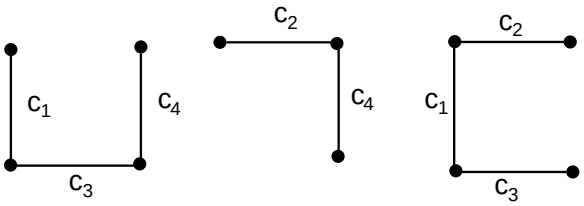
- If $g_{i,j} = \delta_{ij}$, where $\delta_{ij} = u_{ij} - d_{ij}$ then this is equivalent to seeking a solution that achieves correct timing behavior.


1
2
1
3
2
3
1
2

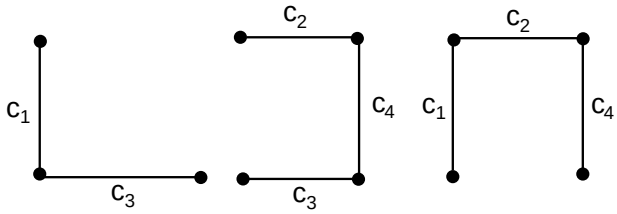
(a)


1,2
$c_2 = 2$
1,3
$c_1 = 2$
$c_4 = 2$
2,3
$c_3 = 2$
1,2

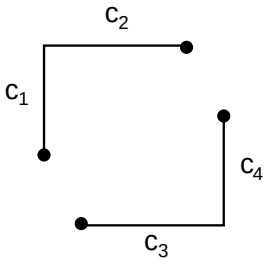
(b)


c_1
c_4
c_3
c_2
c_4
c_2
c_1
c_3

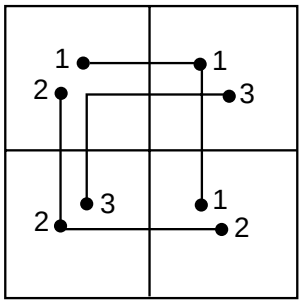
(c)


c_1
c_2
c_4
c_3
c_2
c_1
c_4

(d)


c_2
c_1
c_4
c_3

(e)


1
1
2
3
3
1
2
2

(f)

- The gate-array layout of previous figure has four cells and three nets.

- Assume that the capacity of each boundary is equal to two tracks.

- First, a grid graph is built.

- Each node of the graph is marked with the label of those nets which have pins in that node.

- There are three trees for net 1, three trees for net 2, and two trees for net 3.

- Assume that the cross capacity of each boundary is equal to two tracks. There are four boundaries corresponding to four edges in the grid graph. The matrix A will be as follows,

	t_1^1	t_2^1	t_3^1	t_1^2	t_2^2	t_3^2	t_1^3	t_2^3
1	0	1	1	1	0	1	1	0
2	1	0	1	0	1	1	1	0
3	0	1	1	1	1	0	0	1
4	1	1	0	0	1	1	0	1

- Assume that the cost of tree t_i^j is equal to the length of that tree. In that case,
  $g_{1,1} = 2$, $g_{1,2} = 3$, $g_{1,3} = 3$,
  $g_{2,1} = 2$, $g_{2,2} = 3$, $g_{2,3} = 3$, and
  $g_{3,1} = 2$, $g_{3,2} = 2$.

- Therefore, the resulting 0-1 integer program is

$$\begin{cases} F = 2x_{1,1} + 3x_{1,2} + 3x_{1,3} + 2x_{2,1} + 3x_{2,2} \\ \qquad + 3x_{2,3} + 2x_{3,1} + 2x_{3,2} \leftarrow \text{minimize} \\ Subject\ to: \\ x_{1,1} + x_{1,2} + x_{1,3} = 1 \\ x_{2,1} + x_{2,2} + x_{2,3} = 1 \\ x_{3,1} + x_{3,2} = 1 \\ x_{1,2} + x_{1,3} + x_{2,1} + x_{2,3} + x_{3,1} \leq 2 \\ x_{1,1} + x_{1,3} + x_{2,2} + x_{2,3} + x_{3,1} \leq 2 \\ x_{1,2} + x_{1,3} + x_{2,1} + x_{2,2} + x_{3,2} \leq 2 \\ x_{1,1} + x_{1,2} + x_{2,2} + x_{2,3} + x_{3,2} \leq 2 \\ x_{i,j} = 0, 1 \quad 1 \leq i \leq 3,\ 1 \leq j \leq 3 \end{cases}$$

- The integer programming formulation is elegant and finds a globally optimum assignment of the nets to routing regions.

- However, this approach suffers from the following problems:

  (a) We need to identify several Steiner trees for each net.

  (b) The trees should be selected so as to guarantee the feasibility of the problem.

  (c) There are too many $a_{i,j}$'s, leading to too many constraints.

  (d) There may be too many arcs, i.e., too many c_i's, leading to too many constraints.

  (e) All constraints are integer constraints.

- Hierarchical approaches come to the rescue, and can be used to solve some of the above problems. This problem decomposition will of course be at the price of not achieving global optimality!

Simulated Annealing

Simulated annealing is used in the the TimberWolf package to improve an initially constructed global routing solution.

In TimberWolf, the layout style assumed is Standard Cells style, and global routing is solved in two stages.

The objective of the first stage is to assign the nets to the horizontal routing channels so as to minimize the overall channel densities.

At the end of the first stage, all nets that are switchable (assignable to an adjacent channel) are identified.

The goal of the second stage is to attempt to reduce the overall channel densities by changing the channel assignment of the switchable nets.

After global routing, TimberWolf proceeds with a refinement of the placement by randomly interchanging neighboring cells. After each interchange, both stages of the global router are invoked to reroute the nets affected by the interchange.

It is only during the second stage of global routing (as well as the placement refinement phase) that simulated annealing is used.

Terminology:

A group of pins of a given cell that are internally connected are called a *pin cluster*. The pins of the same cluster are all equivalent.

The x-coordinate of a pin cluster P is equal to the average of the x-coordinates of its constituent pins, i.e.,

$$x(P) = \frac{1}{|P|} \times \sum_{i \in P} x(i) \tag{9}$$

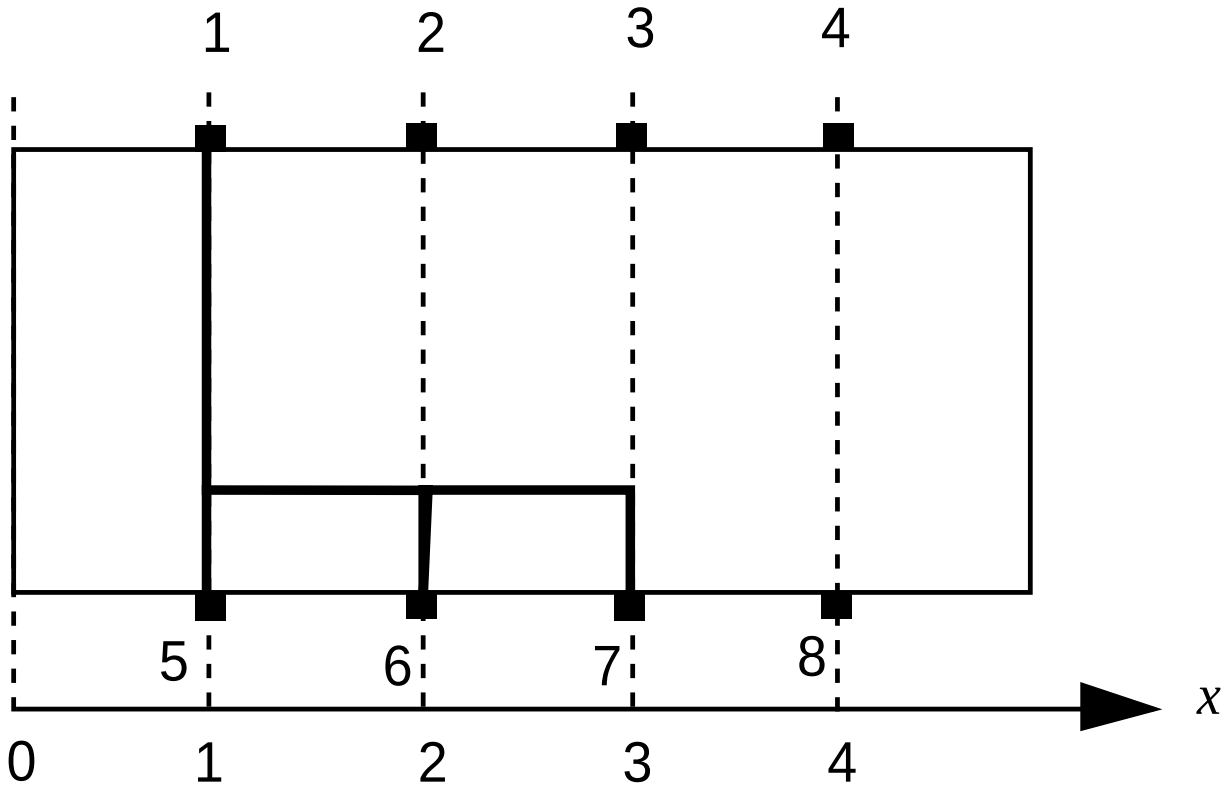


Illustration of the pin cluster concept;
$P = \{1, 5, 6, 7\}$ and $x(P) = \frac{1+1+2+3}{4} = \frac{7}{4}$.

A portion of a net connecting two pin clusters, say P_1 and P_2, is called a *net segment*.

If P_1 and P_2 belong to two different cells placed on the same row and both have pins on the top and bottom sides of the cells, then the net segment connecting both clusters is called a *switchable segment*.

The cost function used is equal to the sum of the total channel densities, i.e.,

$$D = \sum_{\forall c} d(c) \tag{10}$$

where $d(c)$ is the density of channel c, which is equal to the number of nets assigned to the channel.

The First Stage

The global routing algorithm of the first stage has four distinct steps which are executed for each net.

(1) **Initialization:** All pin clusters of the net are identified, together with their x-coordinates. Then the pin clusters are sorted on their x-coordinates, from smallest to largest.

(2) **Construction of a cluster graph:** In this step, a cluster graph is constructed, where nodes model the pin clusters and edges possible connections (net segments) between the corresponding pin clusters.

(3) **Construction of a minimum spanning tree:** In this step, Kruskal algorithm is used to construct a minimum spanning tree of the cluster graph.

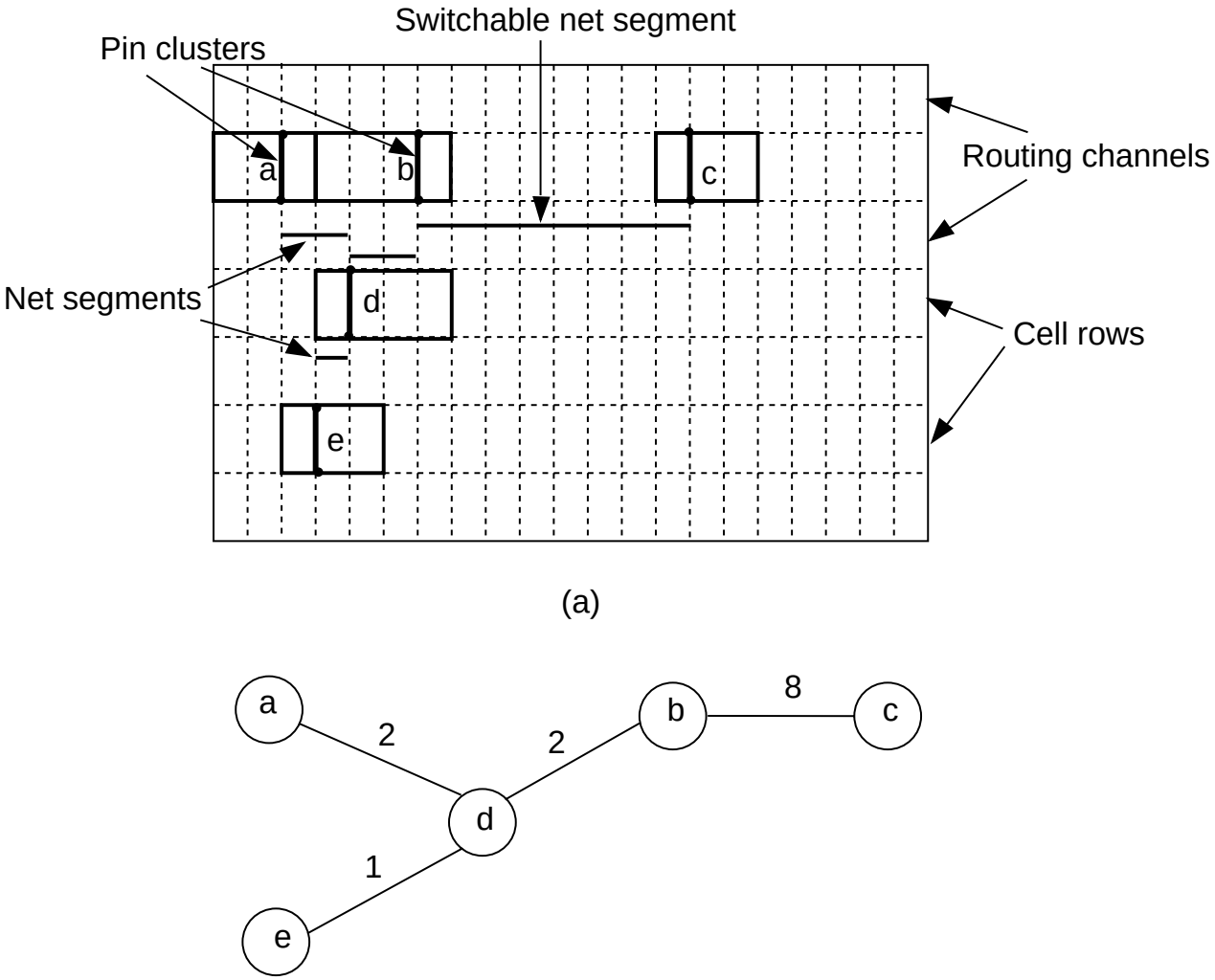
(4) **Identification of all net segments:** In this final step of the first stage, individual pins within the pin clusters are selected to form the actual net segments. Also, if the net segment is switchable, then two pairs of pins are selected, one pair for the upper row, and one for the lower row.

Example:

Assume we are given the following partial standard-cell design.

Suppose that each pin cluster has exactly two pins available on opposite sides of the cell. Assume further that one of the nets is connecting exactly five pin clusters labeled a, b, c, d, and e.

We would like to illustrate how the cluster graph is constructed for this net.



Construction of the cluster graph:
(a) A partial standard-cell design.
(b) Cluster graph constructed during stage 1 of TimberWolf.

solution: The sorted pin clusters are $[a, e, d, b, c]$.

The sorted sequence will be processed one element at a time until the cluster graph is constructed (one edge at a time).

At the first execution of the outer repeat loop, $P_1 = a$, a is removed from the sequence $clusters$, and the variable $TorB = 0$.

$TorB$ is equal to 0, $+1$, -1, to indicate that the next pin is at the same row, the top row, or the bottom row respectively.

The closest pin to the right of a that is located at the top, bottom, or same row is pin cluster d. Therefore, $P_2 = d$, $row(d) = 2$, and $TorB$ is set equal to 1 (P_2 is on top of P_1). Hence the edge (a, d) is added to the cluster graph.

The next closest pin cluster to the right of a is b. However because $row(b) = 1 = row(a)$, the edge (a, b) is not added.

Then, the next closest pin cluster to the right of a is c. Since $row(c) = row(a) = 1$, edge (a, c) is not considered.

At this time, we exit from the inner repeat loop, $TorB$ is reset to 0, and the $head(clusters)$ returns pin cluster e, which is removed from $clusters$.

The closest pin cluster to the right of e and located in the same row or adjacent row is pin cluster d. Therefore, $P_2 = d$, $TorB = -1$, and the edge (e, d) is added to the graph.

No other pin in the same or adjacent row is to the right of e. Therefore, we exit from the inner repeat loop.

Continuing in this way, edge (d, b), then edge (b, c) will be added to the cluster graph.

At this moment, the sequence $clusters$ becomes empty.

The cluster graph thus constructed is given in previous Figure. The positive numbers indicated on the edges of the graph represent the lengths of the corresponding net segments. Notice that the graph is already in the form of a minimum spanning tree.

As observed by the author of this approach, the cluster graph will in most cases turn out to be a minimum spanning tree.

Notice that, among the net segments selected, only segment $b - c$ is switchable. Therefore for this segment, two pairs of pins are selected, one if the segment is assigned to the channel above, and one if it is assigned to the channel below.

The Second Stage:

In stage 2, the simulated annealing search technique is used to refine the global routing solution produced by the stage 1 sequential algorithm. Only switchable net segments are considered for re-routing.

The objective of this stage is to minimize the total channel density.

The $generate$ function used to obtain new solutions is as follows. First, a switchable segment is randomly selected from the pool of switchable net segments.

Then, the channel assignment of the selected segment is switched from its current channel to the opposite one.

If the switch reduces the value of the cost function, then the switch is accepted. If the new solution has the same cost as the previous one, then the switchable segment is assigned to the channel with the smaller density over the span of the net segment. The purpose of this decision is to facilitate the following step of detailed routing.

New solutions with higher cost functions are not accepted.

As for the cooling schedule, the temperature is maintained equal to zero throughout the search. Hence only downhill moves are accepted. Furthermore, since $T = 0$, there is no need for the inner loop, nor a need to update the schedule parameter.

The stopping criterion used is a function of the number of switchable segments. The search is stopped after the generation of $S = 30 \times N$ new states, where N is the number of switchable net segments.

Other Approaches

The KFUPM Approach