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

Channel Routing

Sadiq M. Sait & Habib Youssef

December 1995

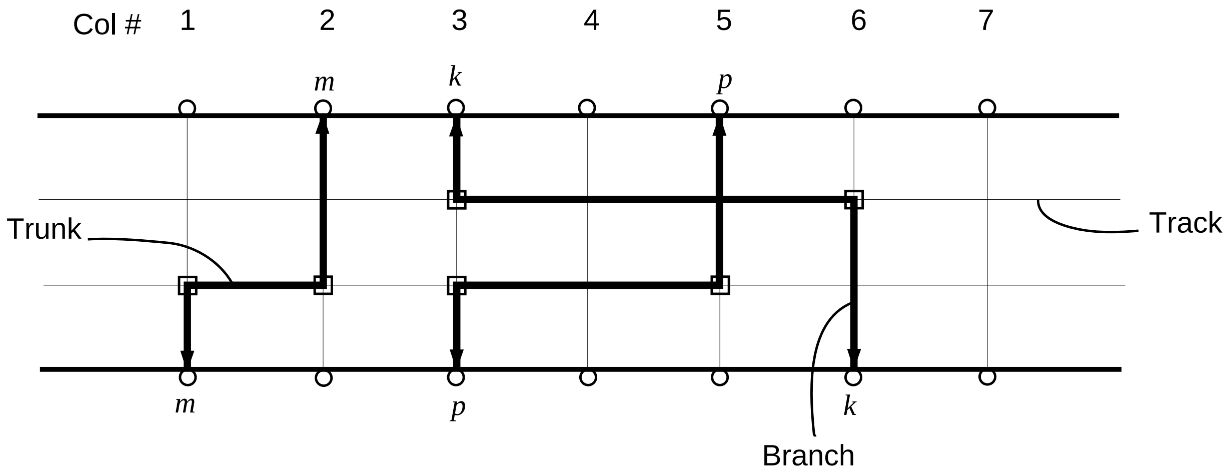
Introduction to Channel Routing

- *Channel routing* is a special case of the routing problem.

- Apply a 'divide-and-conquer' strategy.

- Used in the design of custom chips as well as uniform structures such as gate-arrays and standard-cells.

- Very popular because it is efficient and simple, and guarantees 100% completion.



Problem Definition

- The channel is defined by a rectangular region with 2 rows of terminals along its top/bottom sides.
- A number between 0 and N is assigned to each terminal.
- These numbers are labels of grid points and represent the netlist (zeros indicate that no connection).
- In case of a standard-cell design methodology, the objective is minimize number of tracks for routing.
- For gate-array design methodology the objective is to finish routing.
- Horizontal segments belonging to different nets that do not overlap may be assigned to same track (see Figure).
- Thus, there are horizontal constraints on nets.
- Also, any two nets must not overlap at a vertical column. (see Figure, column 3).

Constraint Graphs

- For any instance of the channel routing problem we associate two constraint graphs; to model the horizontal and vertical constraints.

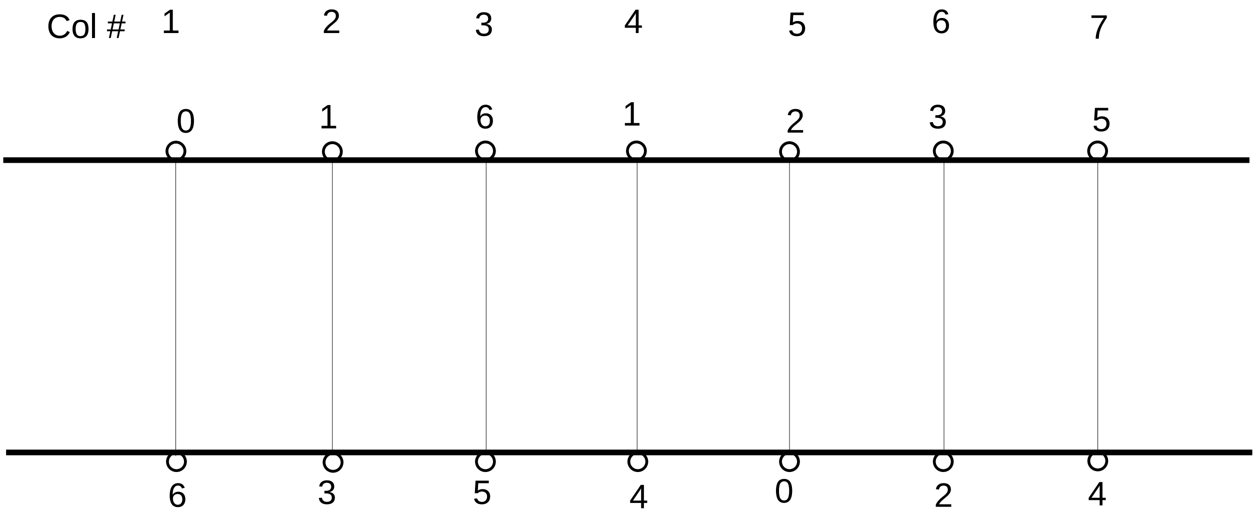
- The $HCG(V, E)$ is an undirected graph where a vertex represents a net.

- An edge $(i, j) \in E$ represents that the horizontal segments of net i and net j overlap.

- The $VCG(V, E)$ is a directed graph where a vertex represents a net.

- Each vertical column introduces an edge $(i, j) \in E$ if net i has a pin on the top and net j on the bottom of the channel.

Example



- The netlist can be represented by two vectors TOP and BOT given by TOP=[0,1,6,1,2,3,5] and BOT=[6,3,5,4,0,2,4].

- To determine if two horizontal segments of nets overlap we define a set $S(i)$, where $S(i)$ is the set of nets whose horizontal segments intersect column i.

- The number of elements in each set is called the local density.

- Clearly, the horizontal segments of two nets in any set $S(i)$ must *not* be placed in the same horizontal track.

- For the above channel routing problem the values of $S(i)$ are:

$S(1) = \{6\}$ $S(2) = \{1, 3, 6\}$
$S(3) = \{1, 3, 5, 6\}$ $S(4) = \{1, 3, 4, 5\}$
$S(5) = \{2, 3, 4, 5\}$ $S(6) = \{2, 3, 4, 5\}$
$S(7) = \{4, 5\}$

- Those sets which are already subsets of other sets can be eliminated.

- For example, $S(1) = \{6\}$, and $S(2) = \{1, 3, 6\}$ are subsets of $S(3) = \{1, 3, 5, 6\}$. Therefore they need not be considered. The remaining sets $S(i)$ after elimination are called **maximal sets**.

- For this example, the maximal sets are:

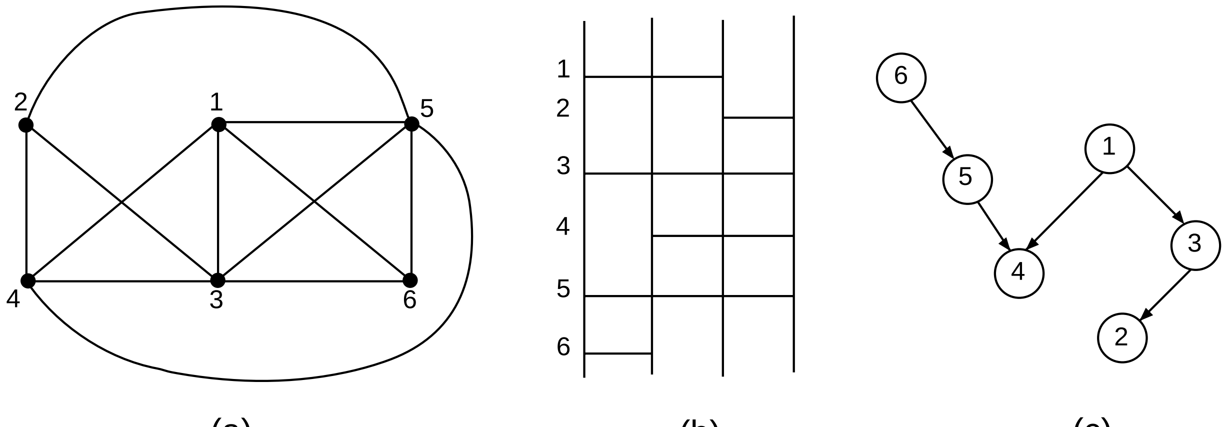
$S(3) = \{1, 3, 5, 6\}$ $S(4) = \{1, 3, 4, 5\}$
$S(5) = \{2, 3, 4, 5\}$

Construction of Horizontal Constraint Graph

- The HCG (also known as the **interval graph**) is constructed by placing an edge between vertices i and j if both i and j belong to a set $S(k)$,

- For example, $S(3) = \{1, 3, 5, 6\}$. Therefore edges are placed between vertices (1,3), (1,5), (1,6), (3,5), (3,6), and (5,6).

- The complete HCG is shown in Figure (a) below.



(a) (b) (c)

Zone Representation

- Is an alternate representation of the HCG.

- It is a graphical representation of the maximal sets $S(i)$.

- Each set $S(i)$ is represented by a column and the elements of the maximal sets $S(i)$ are represented by line segments.

- In terms of the interval graph a zone is defined by a maximal clique, and the clique number is the density.

Zone Table

The zone-table for channel routing problem of the previous Example is given in Table below.

$column$	$S(i)$	$zone$
1	{6}	1
2	{1,3,6}	
3	{1,3,5,6}	
4	{1,3,4,5}	2
5	{2,3,4,5}	3
6	{2,3,4,5}	
7	{4,5}	

Construction of VCG

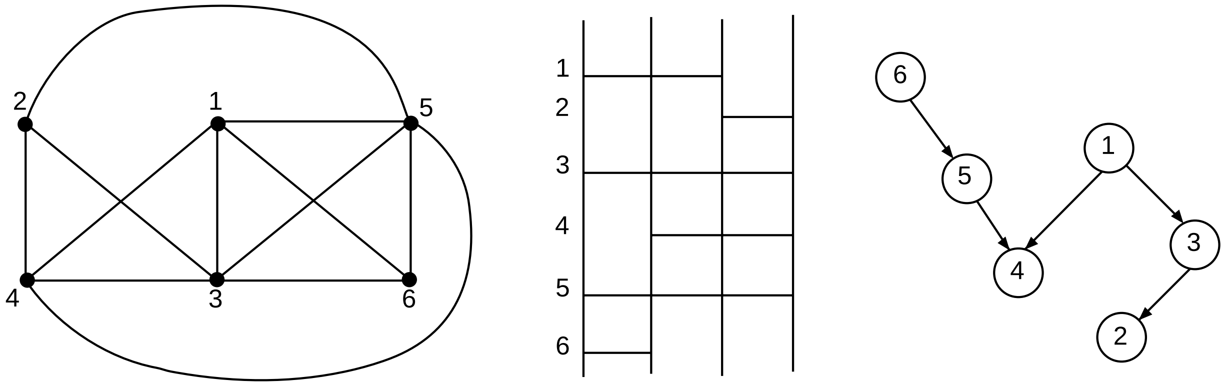
- The VCG is simpler to construct.

- For every column k of the channel not containing a zero in either TOP(k) or BOT(k) a directed edge is drawn from vertex TOP(k) to vertex BOT(k).

- For example, in the given netlist, TOP(2)=1 and BOT(2)=3. Therefore the VCG will have an edge from vertex 1 to vertex 3.

- Similarly there is an edge from vertex 6 to vertex 5, and so on.

- The complete VCG is shown in Figure (c) below.



(a) (b) (c)

Cost function and Constraints

- In the channel routing problem the length of the channel is fixed.
- The objective then is to *minimize* number of tracks.
- Unnecessary contact cuts and vias are also highly undesirable (decrease in yield and reliability).
- The number of layers available for routing is constant. (two or three).
- Several routing models exist for three layer channel routing (VHV and HVH routing).
- In such a model there are obviously no vertical constraints.
- We present heuristics to solve the two layer H-V routing problem.

Approaches to Channel Routing

- Most techniques are based on the left-edge algorithm with several extensions and variations of this.

- In this algorithm tracks are processed one at a time.

- In this lecture we will present the basic Left Edge Algorithm.

- Then, the dogleg algorithm proposed by Deutch which performs splitting of nets is described.

- Finally another technique that uses merging of nets proposed by Yoshimura and Kuh is explained.

- All the above techniques aim at reducing the total number of horizontal tracks required to perform channel routing.

The Basic Left-Edge Algorithm

- The original left-edge channel routing algorithm was proposed by Hashimoto and Stevens.

- It attempts to maximize the placement of horizontal segments in each track.

- Segments of nets are sorted in the increasing order of their *left* end points from the left-edge of the channel, hence the name.

- The basic algorithm imposes the restriction that each net consists of a single trunk.

- Trunks (horizontal segments) are routed on one layer and branches (vertical segments) on the other.

- In no vertical constraints the algorithm produces a solution with minimum number of tracks given by $\max_i \mid S(i) \mid$.

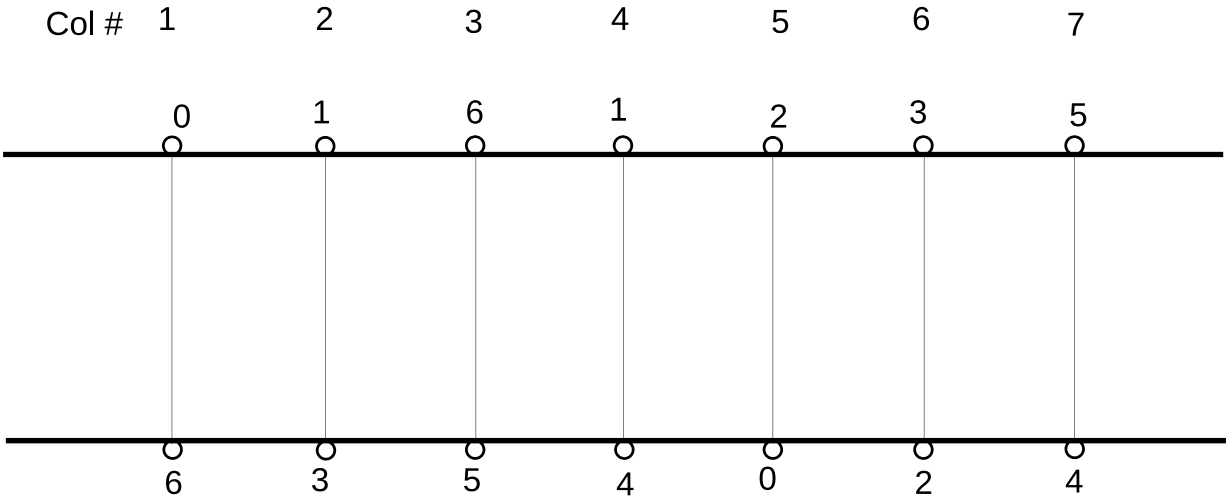
- This is also the lower bound on the number of tracks.

Unconstrained left-edge algorithm

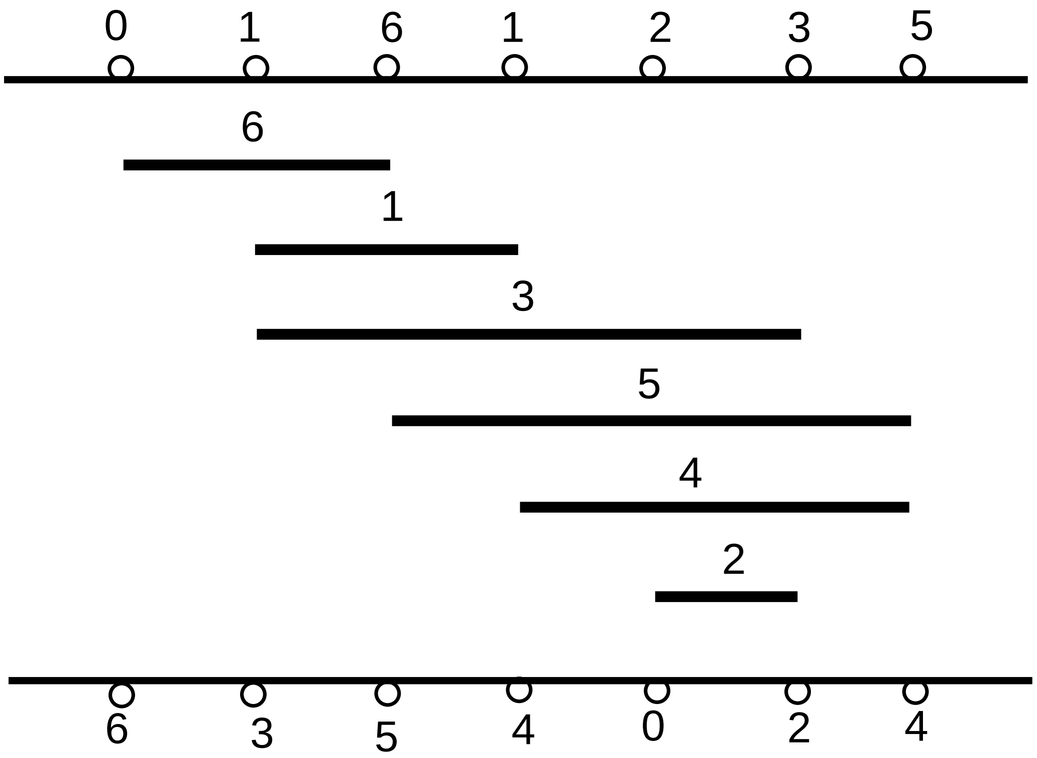
```
ALGORITHM  Unconstrained_ChannelRouter
Begin
1.  Sort all nets on their leftmost
    end positions;
2.  Select the net with the lowest
    left position;
    Place it on the lower most available
    track;
    Delete net from list;
3.  Continue scanning the list and select
    from it nets that do not overlap with
    the nets assigned to this track;
    Assign the nets to the current track
    and delete from list;
4.  If list $\neq \phi$ Then Goto  2;
5.  Exit
End.
```

Example

- For the netlist shown in Figure below:



- Use the left-edge algorithm to assign nets to tracks:
- The trunks, sorted in the order of their left end points, are 6, 1, 3, 5, 4 and 2. This is illustrated in Figure below.



- Ignore the vertical constraints for the moment. Using the left-edge algorithm we try to assign the sorted segments to tracks.

- The first segment chosen from the above sorted list is 6 and is placed in track 1.

- The next segment in sequence is 1. But since we have an edge (1,6) in HCG (see Figure (a)) it cannot be placed in the same track as 6.

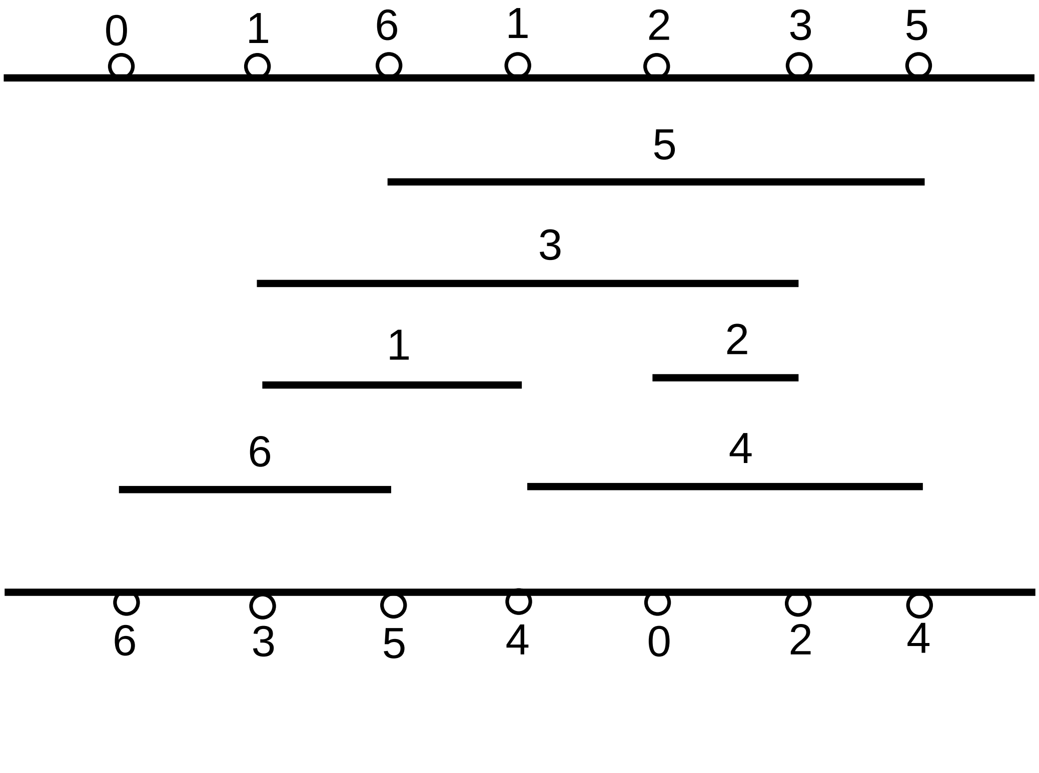
- So also is the case with the trunks of nets 3 and 5 which are after net 1.

- The next net in sequence is 4 and since there is no edge (6,4) in HCG, 4 is assigned to the same track.

- The last element in the sorted list is 2, and although there is no edge (6,2) in HCG, we do have (4,2), therefore 2 is not assigned to track 1.

- The set of remaining sorted nets contains 1, 3, 5 and 2. Now the same procedure is repeated to place the remaining segments in track 2, and then in track 3 and so on.

- The final solution is shown in Figure below.



Example (contd)

- In the absence of vertical constraints the above solution is acceptable.
- But we do have vertical constraints, and ignoring them will create short-circuit between nets.
- A more elaborate algorithm which takes into account the vertical constraint is the *constrained left-edge algorithm* reported by Perskey et al.
- As in the previous case, horizontal segments are placed on tracks from the lower left corner of the routing region.
- The algorithm will place a horizontal segment of a net only if it does not have any descendants in the vertical constraint graph.
- The algorithm is commonly known as the *constrained left-edge algorithm*.

Constrained left-edge algorithm

```
ALGORITHM Constrained_ChannelRouter
Begin
1. Sort all nets on their leftmost end positions;
2. Select the next net $n$ with the lowest
   left-end position;
     If $n$ has no descendants in VCG
        Then Begin
               Place $n$ on the lowermost
               available track;
               Delete $n$ from the sorted list;
               Delete $n$ from VCG
                End
        Else Goto 2
     EndIf
3. Continue scanning the sorted list and from it
   select those nets which do not overlap with
   nets assigned to this track and have no
   descendents in VCG;
   Remove all selected nets from the list
4. If list $\neq \phi$ Then Goto 2
5. Exit
End.
```

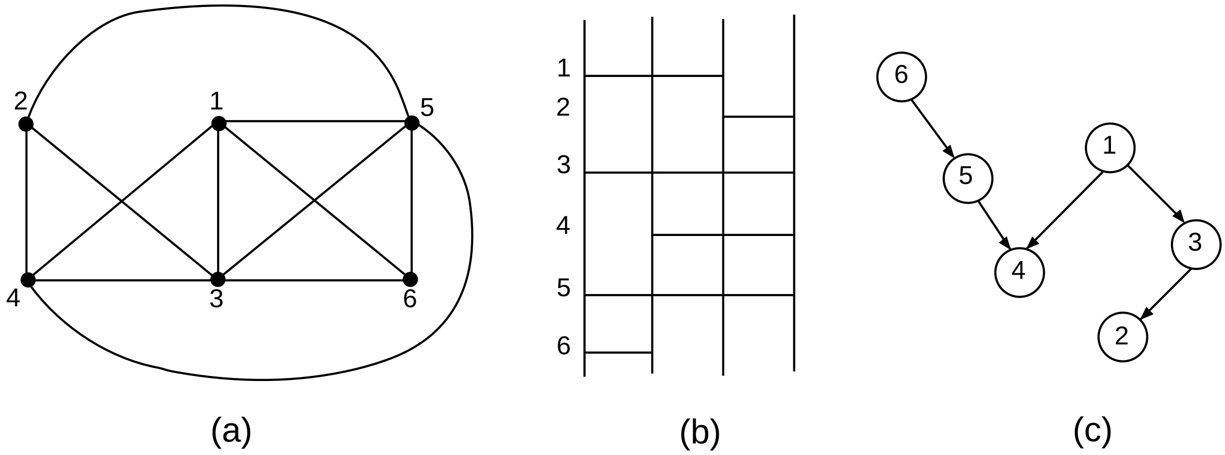
Example (contd)

Problem: Obtain a solution to the previous channel routing problem of taking both the horizontal and vertical constraints into account.

- The same procedure as above is now repeated but taking into consideration the vertical constraints.

- In this case, a segment corresponding to a net can be placed in a track only if the nets corresponding to its descendants have already been assigned.

- Referring to the vertical constraint graph shown below we see that only nodes 4 and 2 have no descendants.



- Now scanning the sorted list we ignore nets 6, 1, 3, and 5 because they all have descendants and the corresponding nets have not been assigned.

- The first candidate is net 4. Therefore net 4 is assigned to track 1 and is deleted from the sorted list as well as from the VCG.

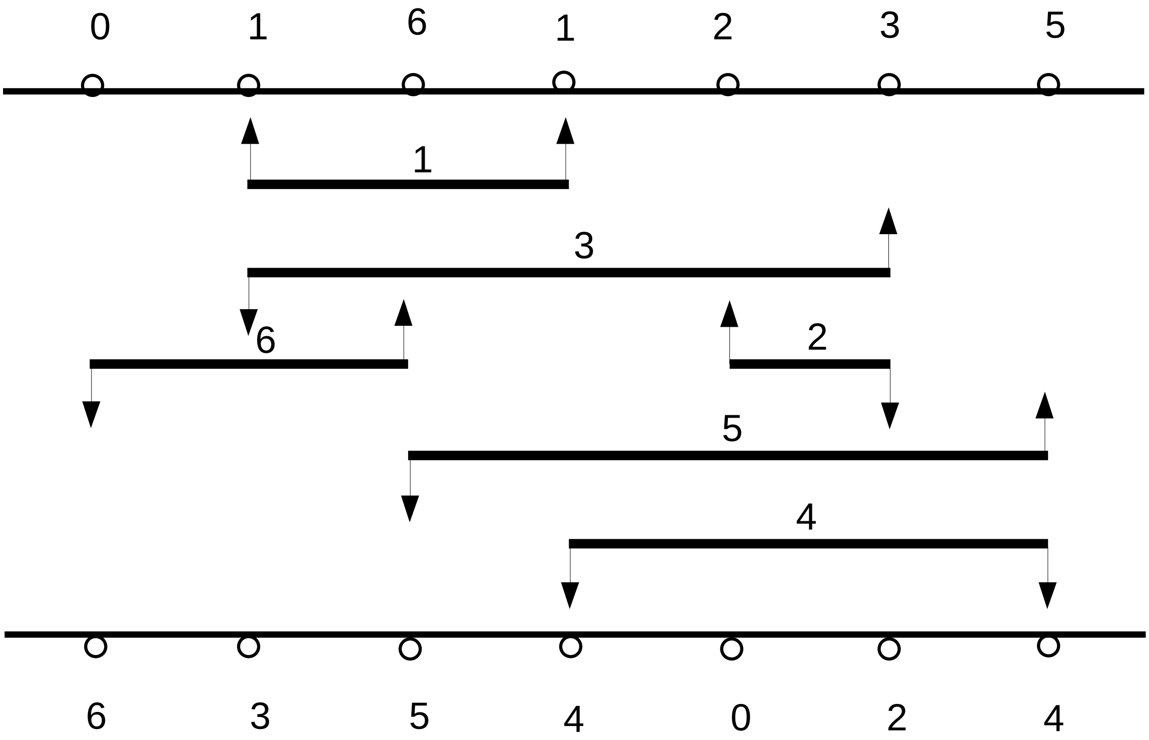
- Continuing the scanning of the sorted list, we reach net 2, which cannot be assigned to track 1 because of horizontal constraint (Figure (a)).

- The nets remaining in the list are 6, 1, 3, 5, and 2.

- We now search for candidates that can go into track 2. Scanning the sorted list, we ignore nets 6, 1, and 3 since these have descendants in the VCG.

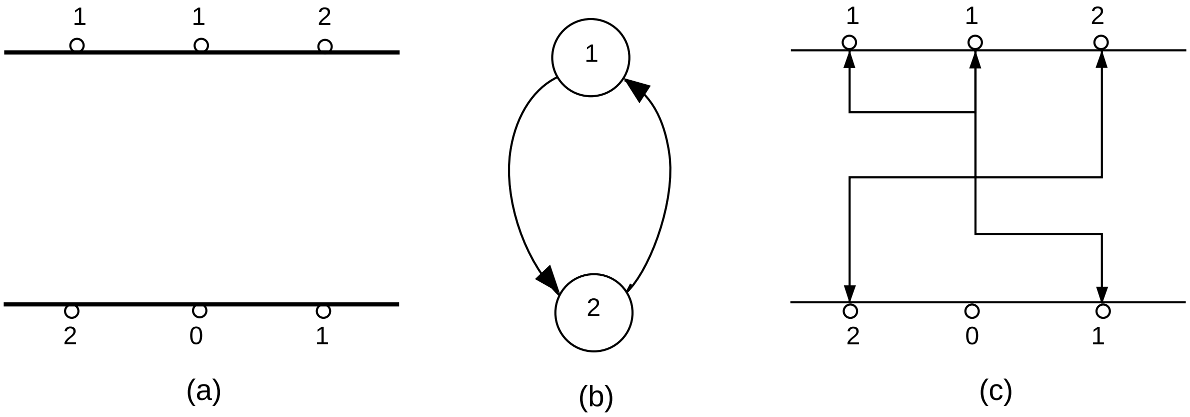
- The next net, which is 5, is chosen and assigned to track 2. Net 2, the next in sequence cannot be assigned to the same track as net 5 because of horizontal constraint.

- The above procedure is continued, and the final solution is shown in Figure below.


0
1
6
1
2
3
5
1
3
6
2
5
4
6
3
5
4
0
2
4

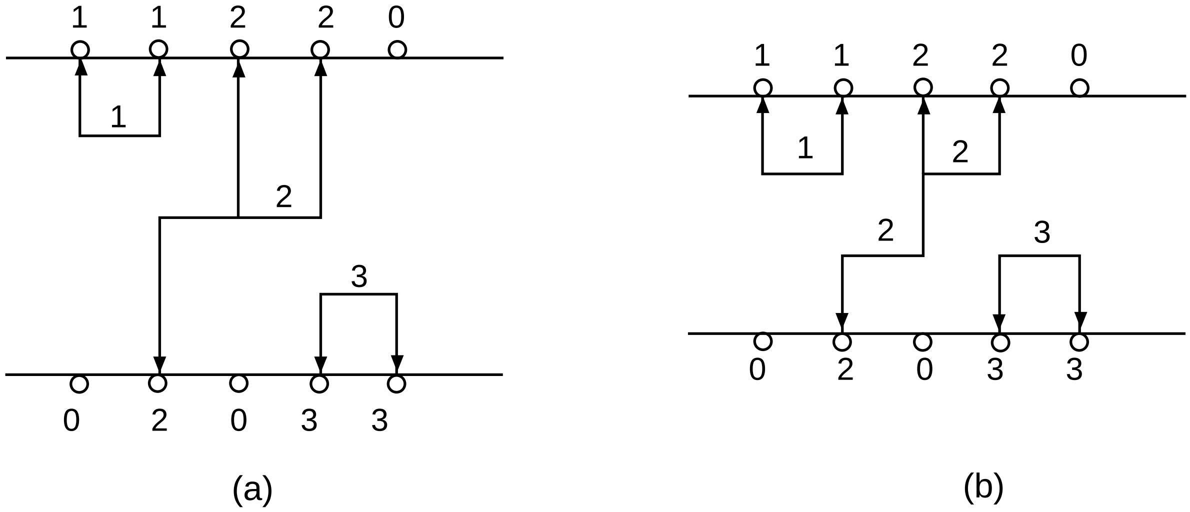
Dogleg Algorithm

- The algorithm mentioned in the previous section will fail if there are cycles in the VCG.



(a) Routing problem. (b) Vertical constraint graph. (c) Solution using doglegging.

- The constrained left-edge algorithm fails, since each net (1 and 2) is the descendant of the other.
- Figure (c) shows that a solution to this problem is possible only if we allow splitting of net segments.
- Even if the VCG contains no cycles, it is desirable to allow splitting of horizontal tracks (reduces the channel density).



(a)

(b)

Example illustrating doglegging. VCG has no cycles. (a) Problem and solution without doglegging uses 3 tracks. (b) Solution with doglegging uses 2 tracks.

- The splitting of horizontal segments of a net is called *doglegging*.
- In doglegging we assume that the horizontal splitting of a net is allowed at terminal positions only and no additional vertical tracks are allowed.

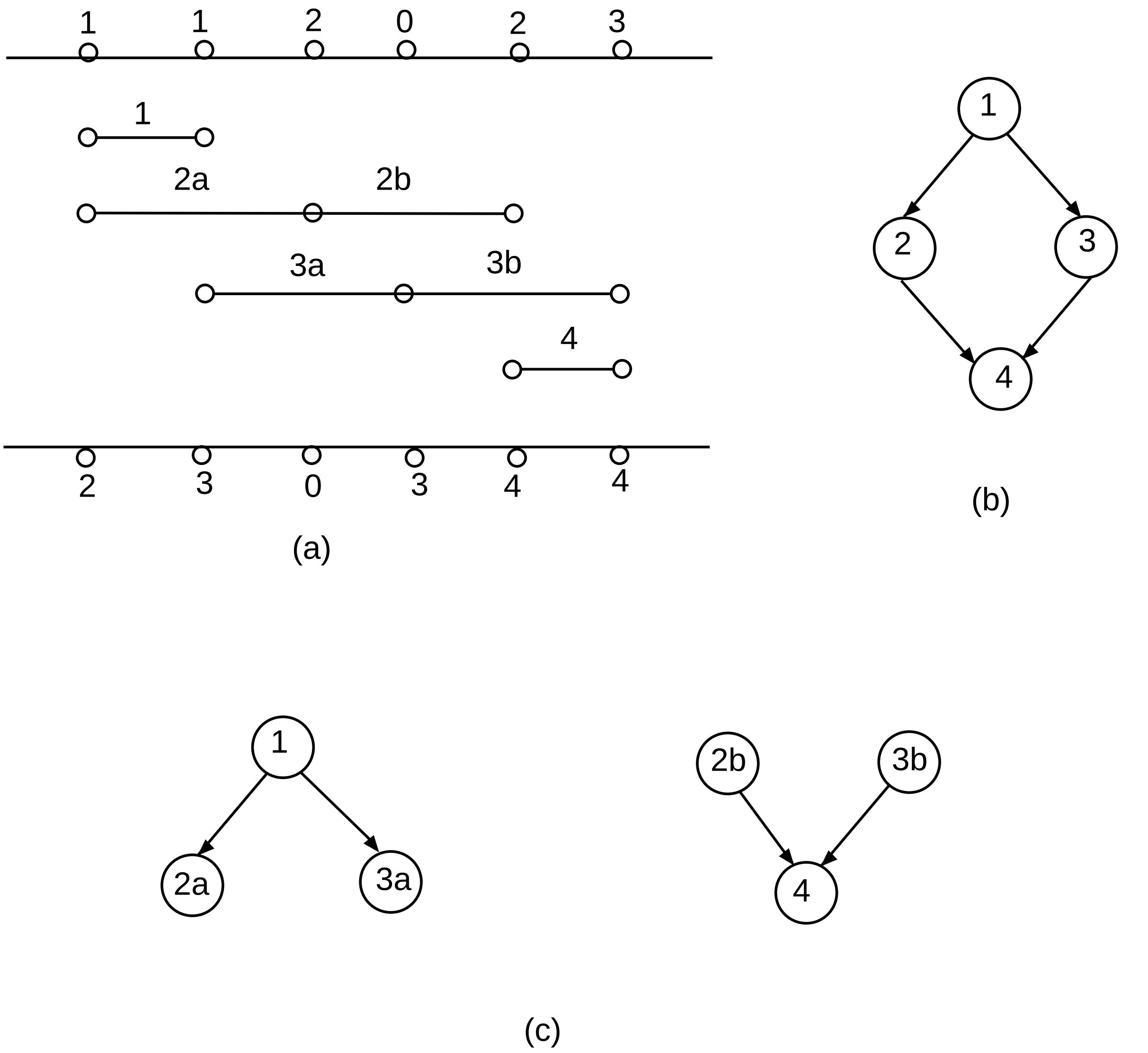
Deutch's Dogleg Algorithm

- This algorithm was proposed by Deutch.

- It helps in reducing the number of horizontal tracks, particularly for channels with multi-pin signal nets.

- The algorithm takes each multi-pin net and breaks it into horizontal segments.

- A break occurs only in columns that contain a pin for that net.



(a) Channel routing problem. (b) VCG without splitting of nets. (c) VCG with nets split.

- Using the new VCG the dogleg algorithm is similar to the constrained left-edge algorithm.

Example

- **Problem:** For the channel routing problem shown in Figure (a) find a solution using the dogleg algorithm.

- **Solution:** The sorted list of net segments is [1,2a,3a,2b,3b,4]. The set of nets $S(i)$ whose horizontal segments cross column i are given by:

$$S(1) = \{1, 2a\} \qquad S(2) = \{1, 2a, 3a\}$$
$$S(3) = \{2a, 2b, 3a\} \qquad S(4) = \{2b, 3a, 3b\}$$
$$S(5) = \{2b, 3b, 4\} \qquad S(6) = \{3b, 4\}$$

- Referring to Figure (c) the net segments that do not have descendants in the VCG are $2a$, $3a$ and 4.

- Scanning the sorted list of net segments we find that the first net segment that does not have descendants is $2a$.

- This segment is placed in the lowermost track.

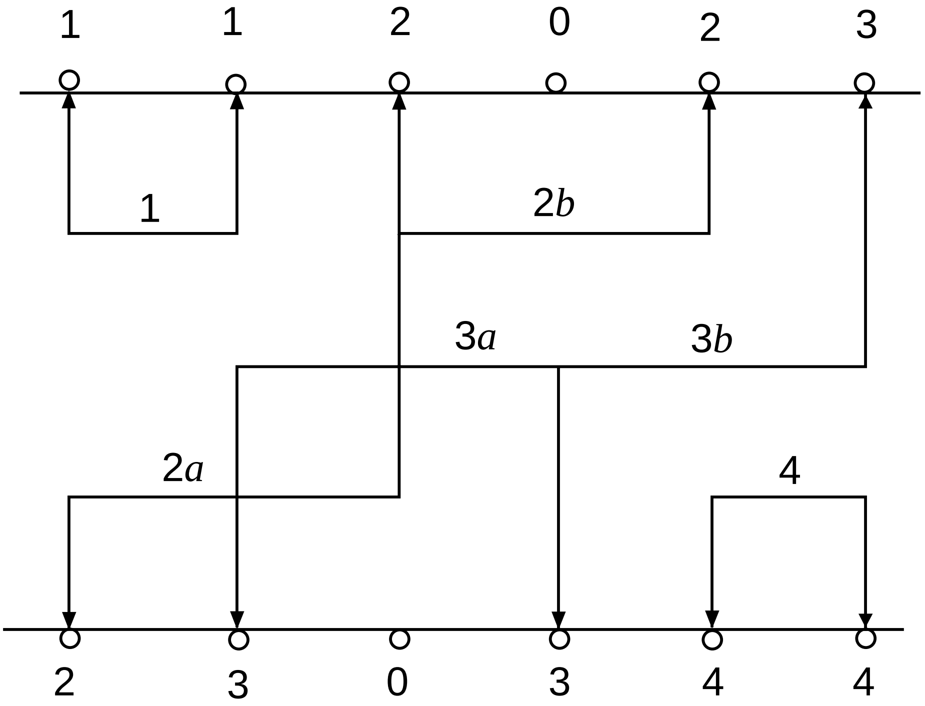
- Continuing the scanning of the sorted list, the next segment in sequence is $3a$, but due to horizontal constraint (see $S(2)$ above) it cannot be placed in the same track as segment $2a$.

- Net 4, the next candidate in sequence, does not have any horizontal constraint with segment $2a$, and therefore is placed in the same horizontal track.

- The placed nets are deleted from the sorted list and the corresponding nodes are deleted from the VCG.

- The above procedure is repeated by placing the remaining segments in the next track, and so on.

- The final solution is shown in Figure below.



Yoshimura and Kuh Algorithm

- **Motivation:** If there is a path n_1-n_2-n_3-$\cdots$-n_k in the vertical constraint graph, then obviously no two nets among $\{n_1, n_2, n_3, \cdots, n_k\}$ can be placed on the same track.

- Therefore, if the longest path in terms of the # of nodes is k, at least k tracks are necessary.

- We now present two algorithms proposed by Yoshimura and Kuh.

  - The first algorithm uses the VCG and the zone representation of HCG and attempts to minimize the longest path in the VCG.

    This is done by merging nodes of VCG so that the longest path length after merging is minimized.

    Obviously, this merging is performed for the purpose of minimizing the channel density.

  - The second algorithm proposed by Yoshimura and Kuh achieves longest path minimization through matching techniques on a bipartite graph.

- Both techniques report better results than the dogleg algorithm.

- Before describing these algorithms we first introduce the required terminology.

Definitions

- Let i and j be the nets for which,

  (a) there exists no horizontal overlap in the zone representation, and

  (b) there is no directed path between node i and node j in the vertical constraint graph,

  (i.e., net i and net j can be placed on the same horizontal track). Then, the operation "merging of net i and net j", results in the following.

  (a) It modifies the vertical constraint graph by shrinking node i and node j into a single node $i \cdot j$; and,

  (b) It updates the zone representation by replacing net i and net j by net $i{\cdot}j$ which occupies the consecutive zones including those of net i and net j.

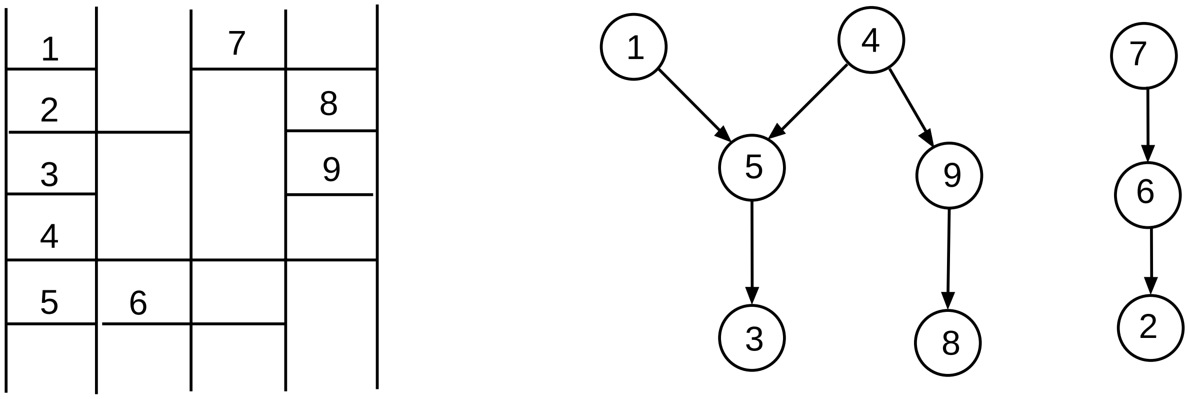
Example

- Consider the netlist given below.

  0 1 4 5 1 6 7 0 4 9 0 0
2 3 5 3 5 2 6 8 9 8 7 9

  The zone table for this example is given in Table below:

column	*S(i)*	*zone*
1	{2}	1
2	{1,2,3}	
3	{1,2,3,4,5}	
4	{1,2,3,4,5}	
5	{1,2,4,5}	
6	{2,4,6}	2
7	{4,6,7}	3
8	{4,7,8}	4
9	{4,7,8,9}	
10	{7,8,9}	
11	{7,9}	
12	{9}	



(a) (b)

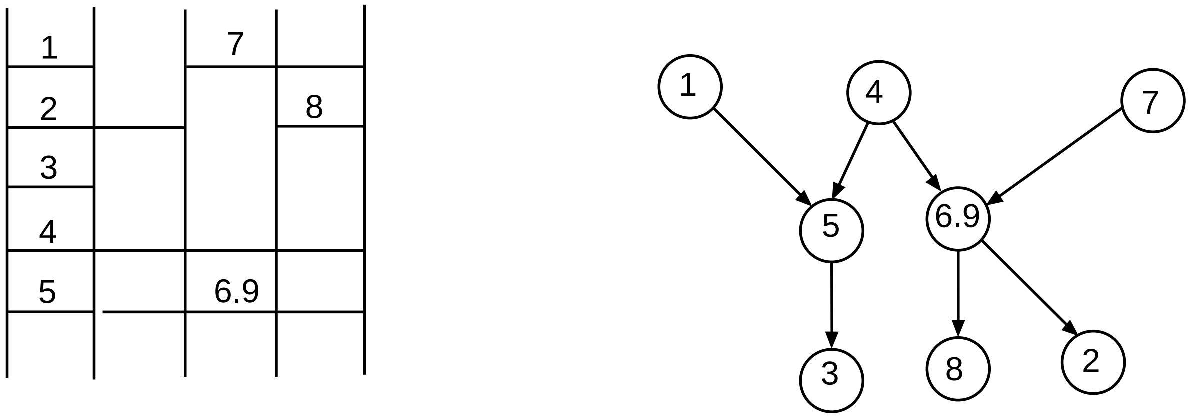
- Consider nets 6 and 9. No horizontal overlap and no vertical conflict (on separate vertical paths), and therefore are candidates for merging.

- The merge operation explained above can then be applied.

- The updated VCG and the zone representation are shown in Figure.

- Due to merging, both nets 6 and 9 will be placed in the *same* horizontal track. However the position of the track is not yet decided.

- Modified zone representation and VCG with merged nodes for the previous Example.



(a) (b)

The Algorithm

```
Algorithm   merge1$\{z_s, z_t\}$
Begin
1. L=$\{\}$; $z_s$ =leftmost zone; $z_t$ =rightmost zone.
2. For $z = z_s$ to $z_t$ Do
   Begin
3.     $L = L + \{$nets which terminate at zone $z\}$;
4.     $R = \{$nets which begin at $z + 1\}$;
5.     merge $L$ and $R$ so as to minimize
       the increase of the longest path
       in the vertical constraint graph;
6.     $L = L - \{n_1, n_2, \cdots, n_j\}$, where
       $\{n_1, n_2, \cdots, n_j\}$, are nets merged in Step 5;
   End;
   EndFor;
End.
```

Example of Yoshimura-Kuh Algorithm

- **Problem:** Apply merging of nets to the channel routing problem of previous Example and obtain the routed solution.

- **Solution:**

  zone 1: Refer to Figure. The set of nets that terminate at zone 1 is $L = \{1, 3, 5\}$ and the set of nets which begin at zone 2 is $R = \{6\}$.

  The merge operation can merge nets (1,6), (3,6) or (5,6).

  Verify that only the merging of nets 5 and 6 causes minimum increase in the length of the longest path.

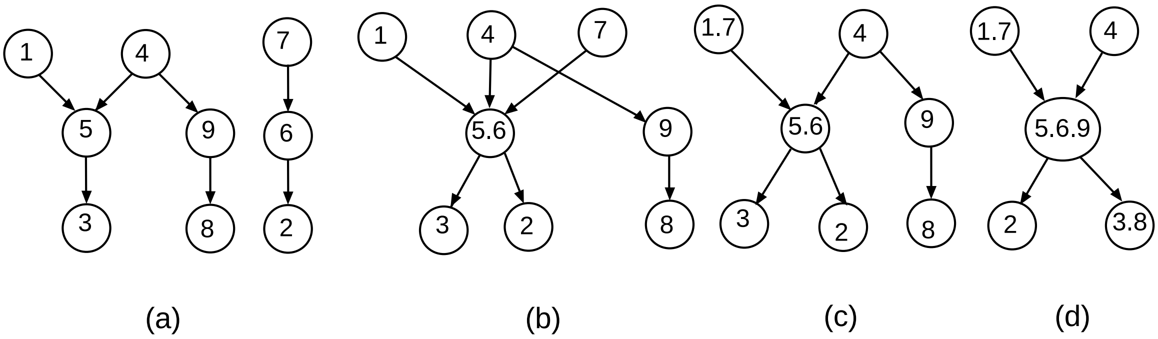
  Therefore nets 5 and 6 are merged to form a merged net $5 \cdot 6$ which ends at zone 3. The updated set L is $L = \{1, 3, 5\} - \{5\} = \{1, 3\}$. (see Figures (a) and (b)).

- **zone 2:** In the next iteration, nets that terminate at zone 2 are added to L. Note that net 2 ends at zone 2. The updated set L is $\{1, 2, 3\}$.

  Only one net begins at zone 3, that is net 7, therefore $R = \{7\}$. In the VCG, (see Figure (b)) since nets 2 and 3 are along the same path as net 7, the only candidate that can be merged with net 7 is net 1.

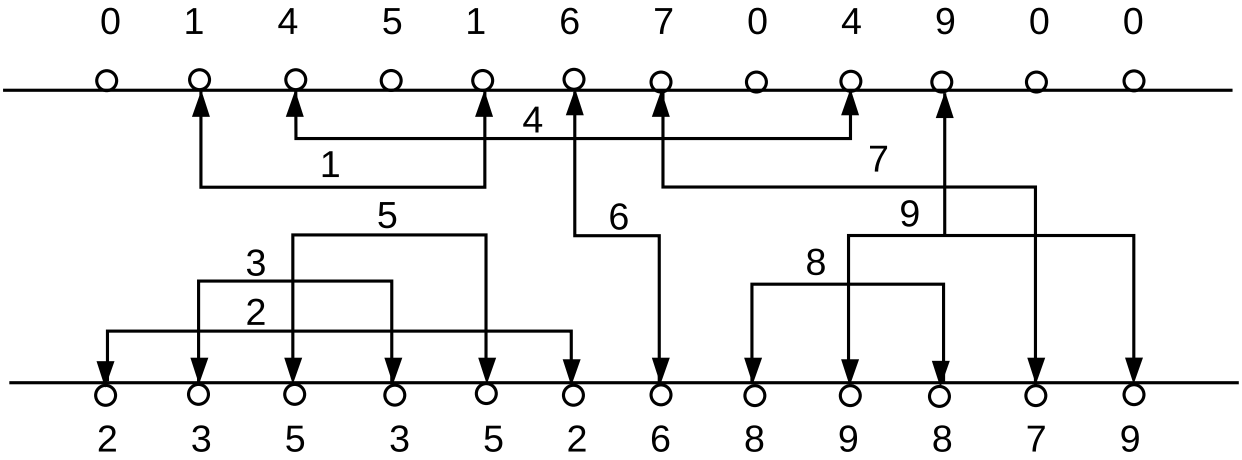
The new merged net is $1 \cdot 7$. Next (see Figure (c)), L is updated by adding nets that end at zone 3 (net $5 \cdot 6$) and removing the nets that are merged (net 1), $L = \{1,2,3\} - \{1\} + 5 \cdot 6 = \{2,3,5 \cdot 6\}$. The set of nets which begin at zone 4 is $R = \{8,9\}$.

- **zone 3:** Observe that the merged net $5 \cdot 6$ ends at zone 3. In this step the merged net $5 \cdot 6$ is merged with net 9, and net 3 is merged with net 8 to form the merged nets $5 \cdot 6 \cdot 9$ and $3 \cdot 8$.
- The above procedure continues until the last zone is reached. Figure below illustrates how the vertical constraint graph is updated by the algorithm.



- Thus, applying the algorithm first, net 5 and net 6 are merged. Then net 1 and net 7, and finally nets $5 \cdot 6$ with net 9 and net 3 with net 8 are merged. The final graph is shown in Figure (d) above.

- In the next step we apply the left-edge algorithm and assign horizontal tracks to the nodes of the graph. The list of nets sorted on their left-edges is $[2, 3 \cdot 8, 1 \cdot 7, 5 \cdot 6 \cdot 9, 4]$.
- The final routed channel is shown in Figure below.



An Improved Algorithm Based on Matching

- In the algorithm of the previous section it is possible that a merging of nets may block subsequent mergings.

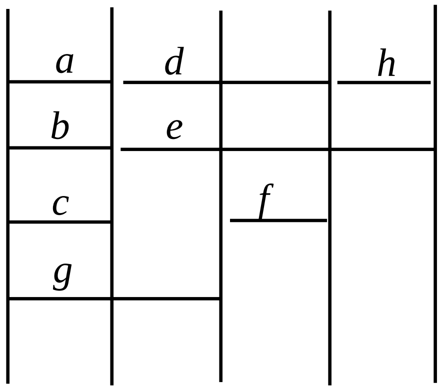
- To avoid this type of situation as much as possible Yoshimura and Kuh introduced another algorithm.

- In this algorithm a bipartite graph G_h is constructed where a node represents a net and an edge between net a and net b indicates that nets a and b can be merged.

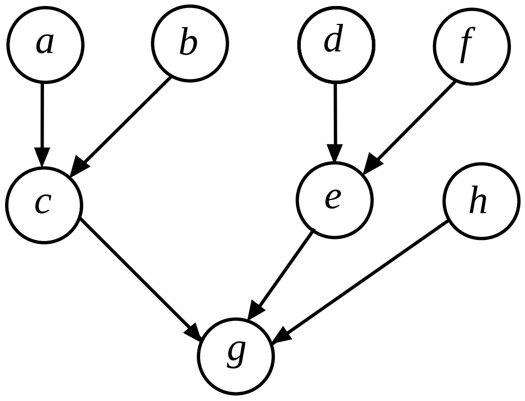
- A merging is expressed by a matching on the graph which is updated dynamically.

Example

- Given the problem instance of Figure below.



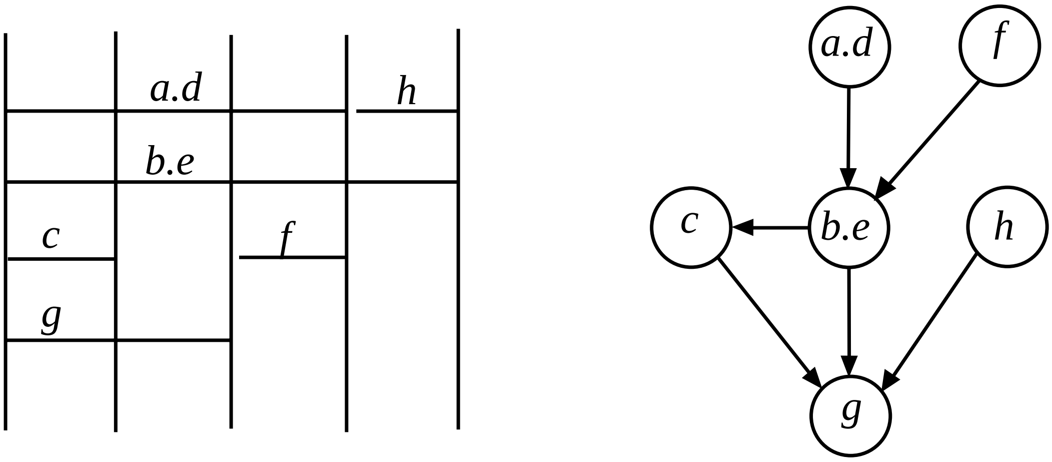
(a)



(b)

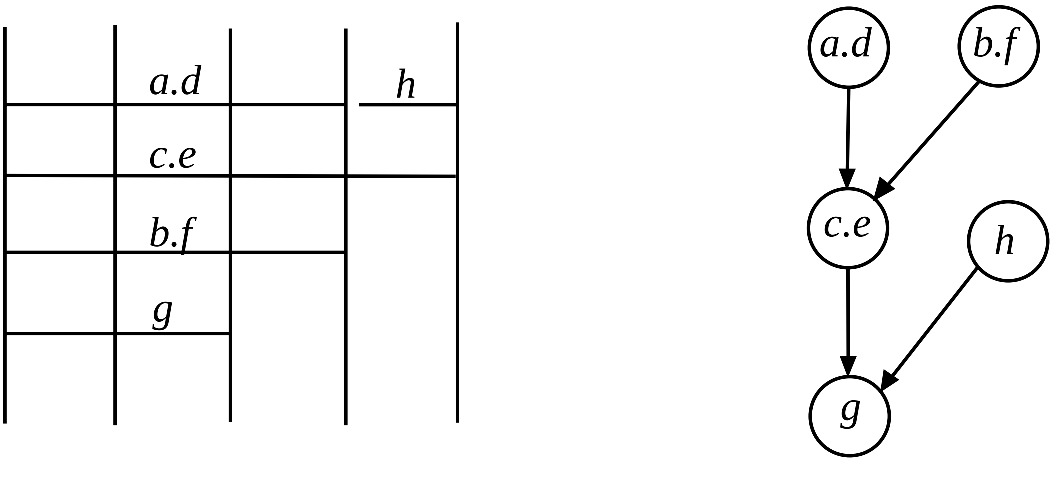
- Let us assume that at zone 1 the algorithm merges net a with net d, and net b with net e respectively (if we follow the merging algorithm of the last section these mergings will not occur, but they are assumed only for illustration).

- The VCG and zone representations are modified as shown in Figure below.



(a) (b)

- The merged VCG indicates that net f cannot be merged with either net c or net g because a cycle would be created.
- However if we merge net a with d and net c with e, then net f can be merged with net b as illustrated in Figure below.
- Merging of nets a and d, c and e, and b and f.



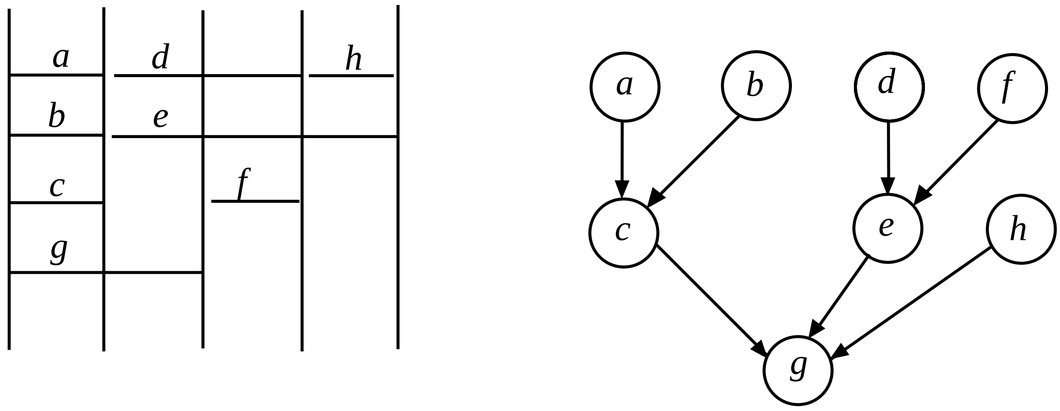
(a) (b)

- Therefore the final solution is order dependent.

Example

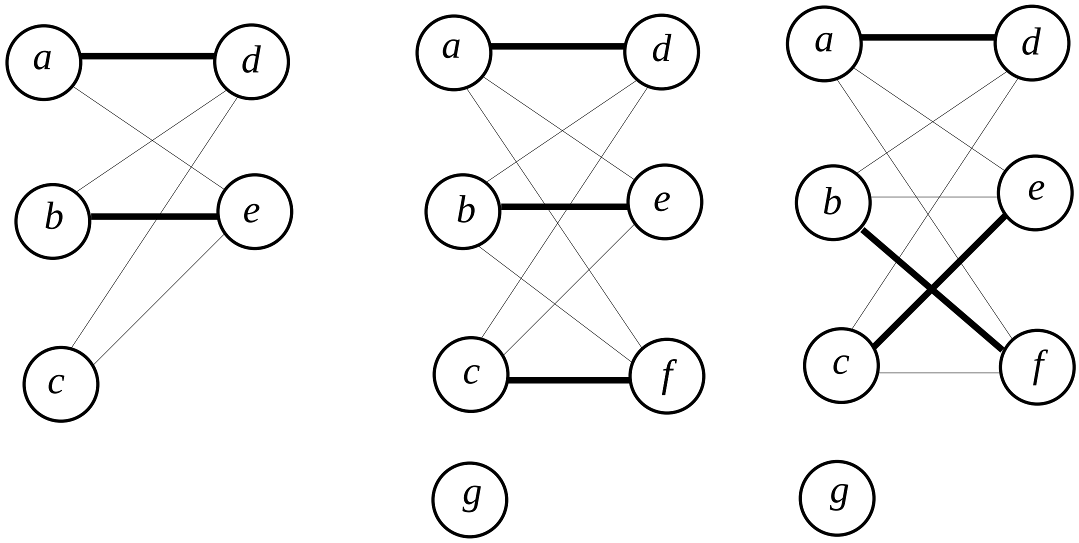
- In this example we introduce a bipartite graph where a node represents a net and an edge between net a and net b signifies that net a and net b can be merged.

- A merging is expressed by a matching on the graph. The idea is explained using the example of Figure below.



(a) (b)

- We see that net d as well as net e can be merged with any of the three nets a, b or c in zone 1. The algorithm constructs a bipartite graph G_h as illustrated in Figure (a) below and a temporary merging is feasible but neither the vertical constraint graph nor the zone representation are updated.



(a) (b) (c)

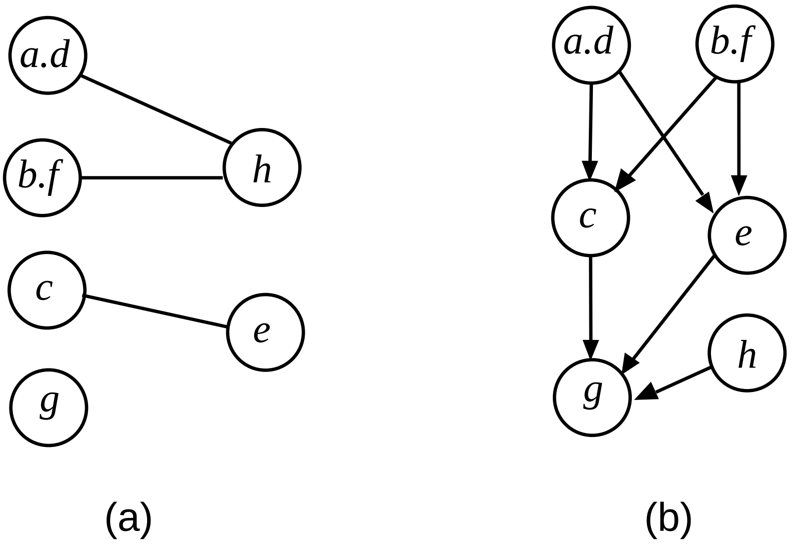
- Next we move to zone 2, where net g terminates and net f begins. So we add g to left and f to the right of G_h as shown in Figure (b).

- Since the VCG in Figure (b) indicates that net f can be merged with either net a, net b or net c, three edges are added and the matching is also updated as shown by the heavier lines in Figure (b).

- Of course there is no guarantee that the merging which corresponds to the updated matching satisfies the vertical constraints (horizontal constraints are satisfied automatically), so the algorithm checks the constraints and modifies the matching as shown in Figure (c).

- At zone 3, net d and net f terminate. This means that, in processing zone 3, node d and node f should be moved to the left side in graph G_h and merged with their partner nets a and b, respectively, as shown in Figure (a) below.

(a) Updated graph G_h. (b) Updated VCG after processing zone 3.



- Net c and net e have not been merged yet, since e has not terminated.

- The vertical constraint graph is also updated as shown in Figure (b) above.

- A matching is next sought for the updated G_h. The above procedure continues until all zones have been processed (see Exercise at the end of the chapter).

Greedy Channel Router

- The channel routers discussed in the earlier sections route the channel one track at a time.
- A heuristic for channel routing known as the '*greedy channel router*' routes the channel column by column.
- In each column the router applies a sequence of greedy but intelligent heuristics to maximize the number of tracks available in the next column.
- It does not use horizontal or vertical constraints. All decisions are made locally at the column.
- The algorithm handles routing problems even with cycles in the VCG.
- Routing is always completed, sometimes with additional columns at the end of the channel called *spillover*.
- Unlike other channel routers, in this technique a net can occupy two different tracks until the heuristic decides to merge them.
- Also, instead of doglegging only at terminal locations, the greedy router allows any horizontal connection to change tracks.

Definitions

- Initial-channel-width
- Minimum-jog-length
- Steady-net-constant
- Rising, falling, and steady nets
- Split net and collapsible net
- Spillover area

Greedy Channel Router .....

- The greedy router takes as input the specific channel routing problem and three non-negative integers, *initial-channel-width*, *minimum-jog-length*, and *steady-net constant*.

- The router scans the channel column-by-column from left to right.

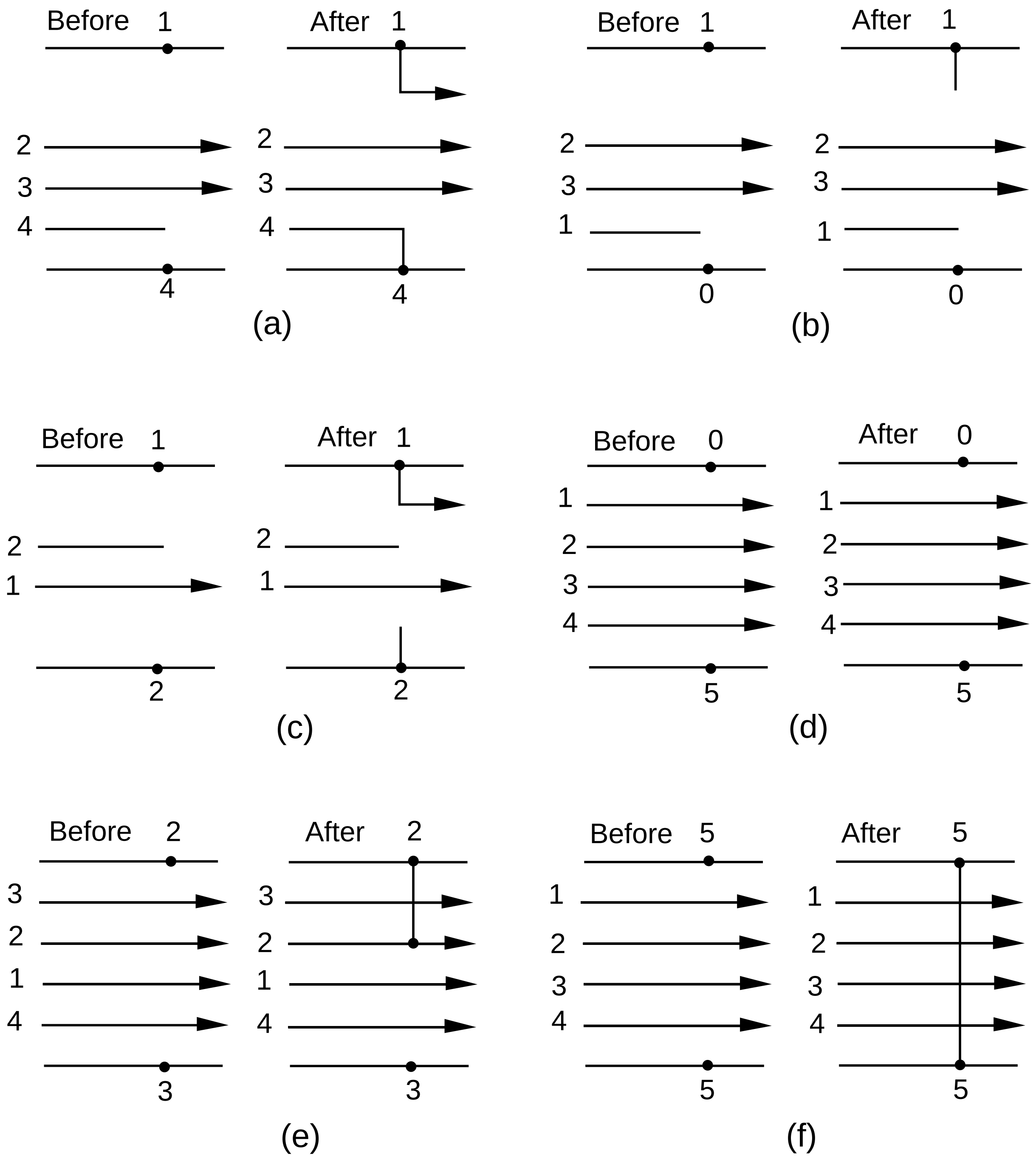
- It uses *six* steps in routing each column.

- The wiring at a given column is completed before proceeding to the next. We will now explain the general idea and the various steps involved.

Steps of Greedy Channel Router

- Step 1: Make feasible top and bottom connections in minimal manner.
- Step 2: Free up as many tracks as possible by collapsing split nets.
- Step 3: Add jogs to reduce the range of split nets.
- Step 4: Add jogs to raise rising nets and lower falling nets.
- Step 5: Widen channel if needed to make previously infeasible top or bottom connections.
- Step 6: Extend to next column.

Step 1 of Greedy Channel Router



(a)

(b)

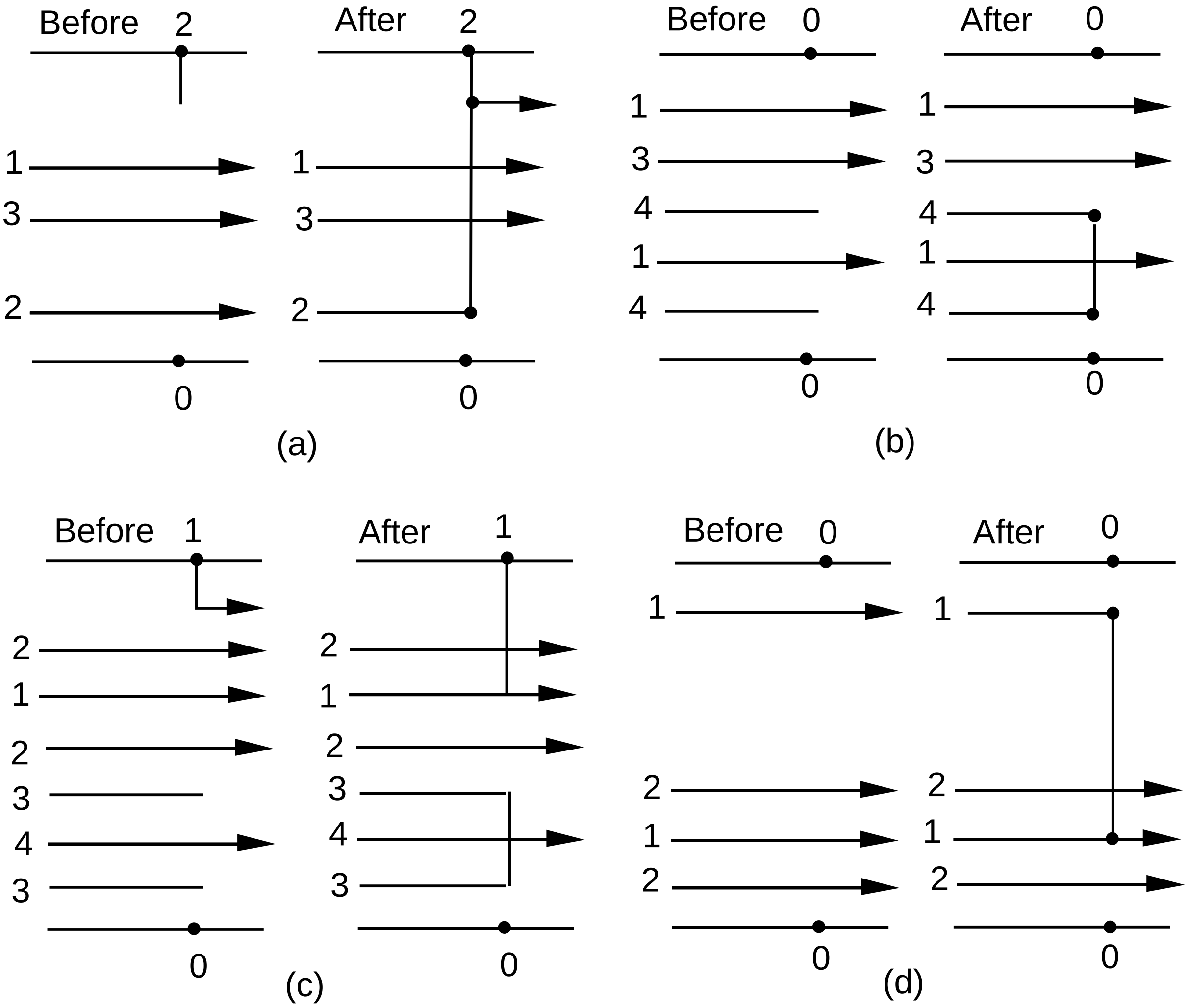
(c)

(d)

(e)

(f)

Step 2 of Greedy Channel Router



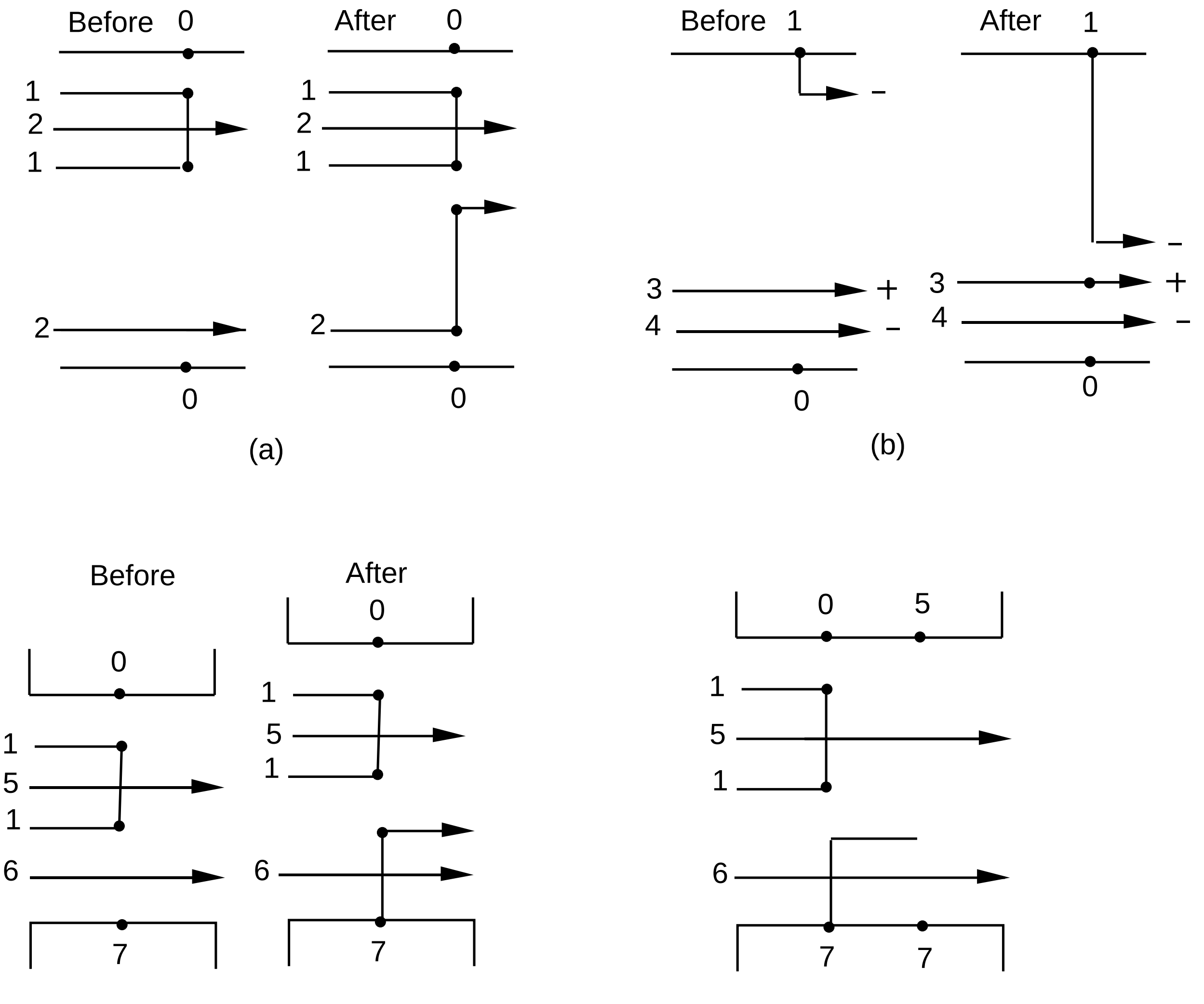
(a)

(b)

(c)

(d)

Steps 3 to 6 of Greedy Channel router. (a)-(d).



(a)

(b)

(c)

(d)

The router will always complete routing successfully, although, to do so, sometimes it may use a few additional columns beyond the right end of the channel.

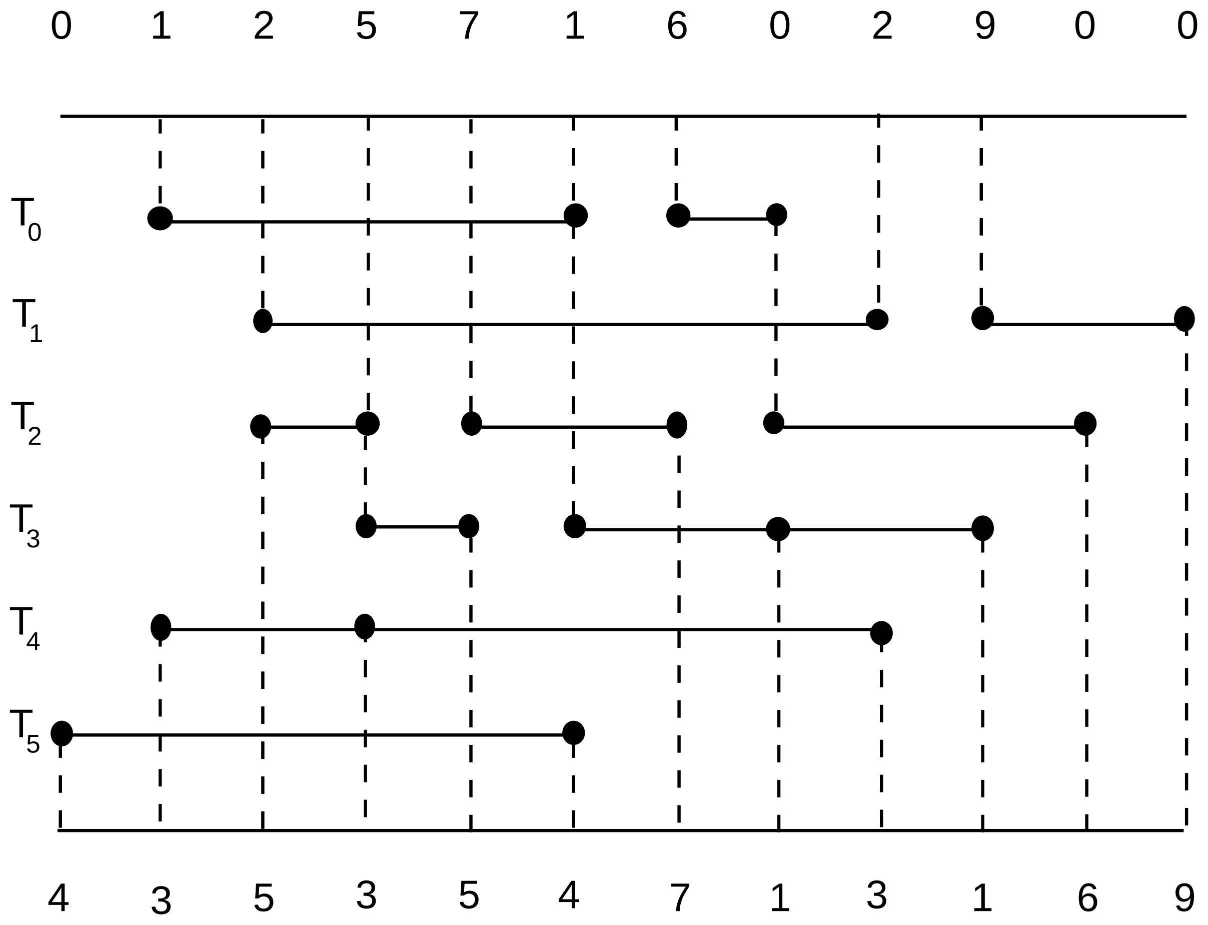
Example of Greedy Channel Router

Apply the greedy algorithm of Rivest and Fiduccia to route the netlist given below.

0 1 2 5 7 1 6 0 2 9 0 0
4 3 5 3 5 4 7 1 3 1 6 9

Let the *initial-channel-width* be 6 tracks, and the *minimum-jog-length* allowed be equal to 1.

Solution to greedy router problem of above Example is shown in Figure below.



- The solution is obtained as follows:

  Let the tracks be numbered from T_0 to T_5 as shown in Figure above. We shall explain the application of the above heuristic at each column starting from the left-most.

  Column 1: Connect pin 4 to T_5 and extend it to the next column.

  Column 2: Connect pin 1 to T_0 and pin 3 to T_4 and extend tracks T_0, T_4, and T_5 to the next column.

  Column 3: Connect pin 2 to T_1 and pin 5 to T_3. Jog net 5 from T_3 to T_2 since net 5 is a rising net. Extend tracks T_0, T_1, T_2, T_4, and T_5 to the next column.

  Column 4: Connect pin 5 to T_2 and pin 3 to T_4. Jog net 5 from T_2 to T_3 since net 5 now is a falling net. Extend tracks T_0, T_1, T_3, T_4, and T_5.

  Column 5: Connect pin 7 to T_2 and pin 5 to T_3. Extend tracks T_0, T_1, T_2, T_4, and T_5 to the next column.

  Column 6: Connect pin 1 to T_0 and pin 4 to T_5. Jog net 1 from T_0 to T_3 since net 1 is a falling net. Extend tracks T_1, T_2, T_3, and T_4.

Column 7: Connect pin 6 to T_0 and pin 7 to T_5. Merge tracks T_2 and T_5. Extend tracks T_0, T_1, T_3, and T_4.

Column 8: Connect pin 1 to T_5. Jog net 5 from T_0 to T_2 and jog net 1 from T_5 to T_3. Extend tracks T_1, T_2, T_3, and T_4 to the next column.

Column 9: Connect pin 2 to T_1. and pin 3 to T_5. Merge tracks T_4 and T_5 and extend tracks T_2 and T_3.

Column 10: Connect pin 9 to T_0 and pin 1 to T_5. Jog net 9 from T_0 to T_1, merge Tracks T_3 and T_5 and then extend tracks T_1 and T_2.

Column 11: Connect pin 6 to T_5, merge Tracks T_2 and T_5, and extend tracks T_0 and T_1.

Column 12: Connect pin 9 to T_5 and merge tracks T_1 and T_5.

- The complete solution obtained by the application of the greedy router is shown in Figure above.

Switchbox Routing

- When rectangular cells are placed on the layout floor, normally two kinds of routing regions are created: *channels* and *switchboxes*.

- Switchboxes are generalizations of channels and allow terminals on all four sides of the region.

- The switchbox routing problem, as will be seen, is more difficult than the channel routing problem.

Switchbox Routing Problem Definition

- A switchbox is a rectangular region with no inside obstructions, and with terminals lying on all four sides.

- The terminals are grouped into a collection S of disjoint sets called *nets*.

- To identify which terminals are to be connected, each terminal is labeled with a net identification number k, $1 \leq k \leq |S|$.

- Formally a switchbox is defined as a region R=$\{0, 1, \cdots, m\} \times \{0, 1, \cdots, n\}$ where m and n are positive integers.

- Each pair (i, j) in R is a grid point. The ith column is a set of grid points $COL(i) = \{(i, j) \mid j \in \{0, 1, \cdots n\}\}$, $1 \leq i \leq m$.

- The jth row or track is a set of points $ROW(j) = \{(i, j) \mid j \in \{0, 1, \cdots m\}\}$, $1 \leq j \leq n$.

- The zeroth and m^{th} columns are the left and right boundaries of the switchbox respectively.

- Similarly, the zeroth and n^{th} rows are the top and bottom boundaries of the switchbox.

- The connectivity and location of each terminal is represented as LEFT$(i) = k$, RIGHT$(i) = k$, TOP$(i) = k$, and BOT$(i) = k$, depending on which side of the switchbox it lies on, where i stands for the coordinate of the terminal along the edge and k for its identification number.

- Lee-type algorithms are not suitable for solving this problem. Lee-type routers do not check ahead to avoid unnecessary blocking of other terminals.

- The goal of the switchbox router is to electrically connect the terminals in each individual net.

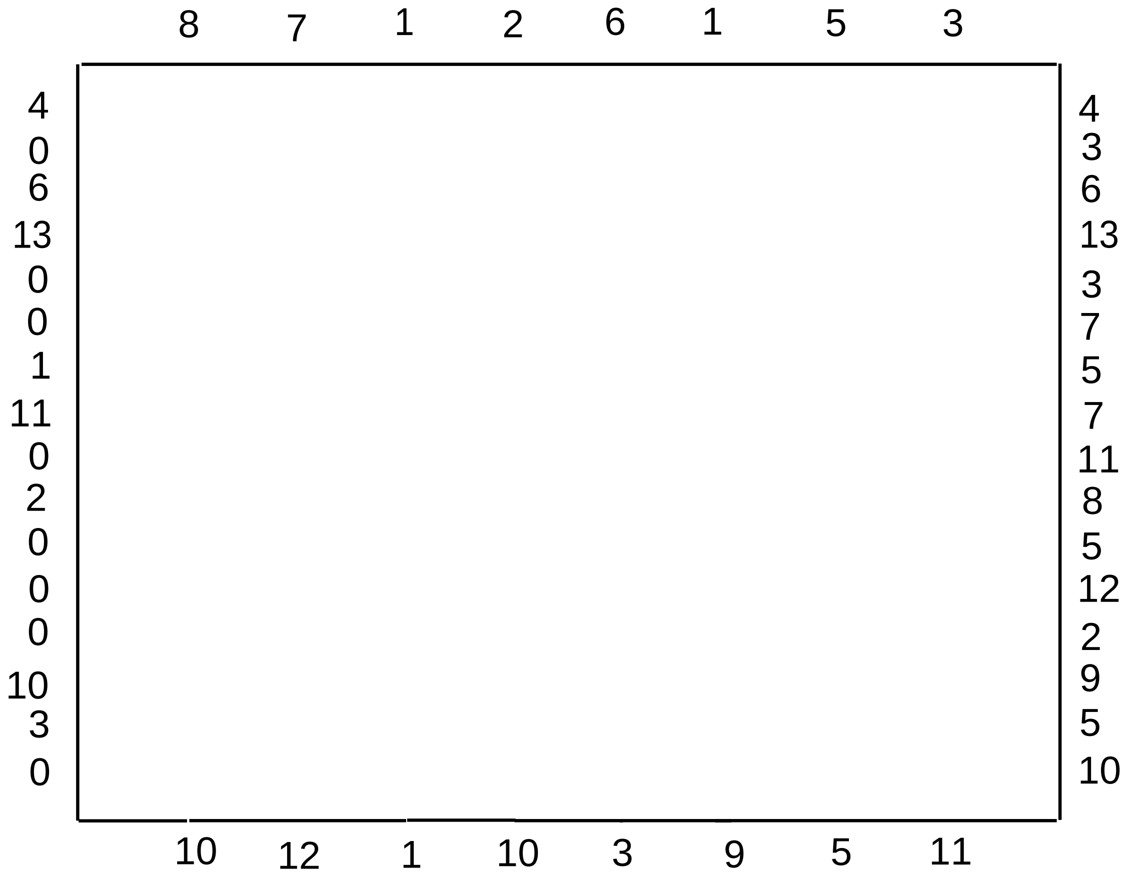
- Connections run horizontally or vertically along rows and columns along grid lines.

- As in previous cases, only a single wire is allowed to occupy each row and each column segment. The wires are allowed to cross.

- An example of a switchbox is shown in Figure below for which we have

  R={0,1,2,$\cdots$,7,8,9} $\times$ {0,1,2,$\cdots$, 15,16,17}

  TOP = (1,$\cdots$,8) = [8,7,1,2,6,1,5,3]
  BOT = (1,$\cdots$,8) = [10,12,1,10,3,9,5,11]
  LEFT = (1,$\cdots$,16) = [0,3,10,0,0,0,2,0,11,1,0,0,13,6,0,4]
  RIGHT = (1,$\cdots$,16) = [10,5,9,2,12,5,8,11,7,5,7,3,13,6,3,4]



Switchbox Routing Algorithm

- We now present an efficient switchbox routing algorithm due to Luk.

- This algorithm is a modification and extension of the greedy routing heuristic of Rivest and Fiduccia discussed earlier.

- Some operations of the greedy heuristic that are not vital to its operation are relaxed and modified to overcome the additional constraints of switchbox routing.

- The additional constraints are:

(1) the matching of the terminals on the LEFT and RIGHT of the routing region,

(2) bringing in left-edge terminals directly into the routing region as horizontal tracks at the column,

(3) instead of jogging to the next top and bottom terminals as in Step 4 of the greedy router, the horizontal tracks must be jogged keeping in mind the target row, which is, a row where a right edge terminal is located. This jogging is to ensure matching the nets with their right edge terminals.

Jogging Strategies

The main modification to the greedy channel router of Rivest and Fiduccia is in the jogging schemes applied to accommodate the additional switchbox constraints discussed above.

The various jogging schemes are defined as follows.

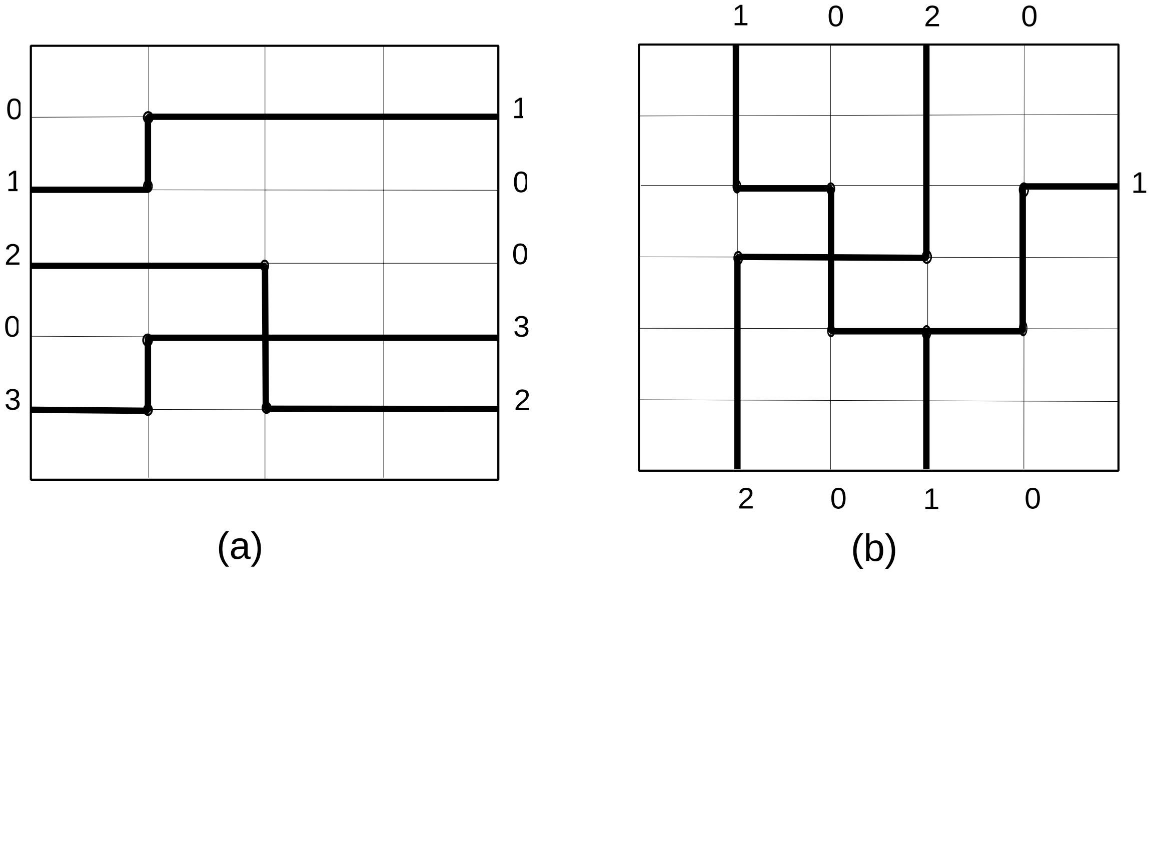
(1) (Jog_R). For nets that have terminals on the RIGHT, this jog is performed until the net occupies a track that matches one of the right edge terminal positions.

(2) ($Jog_{T/B}$). For nets that only have terminals on TOP and BOT, this jog is similar to the one in the greedy channel router.

(3) ($Jog_{T/B}; Jog_R$). In this jogging scheme, first ($Jog_{T/B}$) is performed on every net and then a switch is made to perform (Jog_R) at the column where the last top and/or bottom terminals appear.

Examples of the above jogging schemes are illustrated in Figure below:

Jogging schemes. (a) (Jog_R). Nets 1,2, and 3 jog to right side target terminal. (b) ($Jog_{T/B}$; Jog_R). ($Jog_{T/B}$) performed until column 3 and then (Jog_R).



General Structure of Switchbox Algorithm

The general structure of the switchbox routing algorithm is given in Figure below:

```
**Algorithm** SwitchboxRouter
**Begin**
0.   Determine Scan Direction;
     Bring in LEFT terminals into column.
**Loop**  for $i$ from 1 to $m-1$ **Do**
1.   **If** empty tracks exist then bring TOP($i$)
     and BOT($i$) into empty rows;
2.   Join split nets as much as possible;
3a.  **For** net with no right terminal **Do**
         Bring split nets closer by jogging;
3b.  **For** net with right terminal **Do** SWJOG;
4.   When close to right edge **Do**
         Fanout to targets;
5.   **If** Step 1 failed then increase number of rows;
     **Repeat**  Step1; update columns 1 to $i$;
**While** split nets exist **Do**;
6.   Increase number of columns by 1;
     Join split nets as much as possible;
**End**.
```

Steps of the Algorithm

- The algorithm begins by assigning one of the four edges of the switchbox as the LEFT edge.

- Then the direction of scanning is determined (this is done in Step 0). The quality of the final solution depends on the direction of the scan.

- A good heuristic based on augmented channel density distribution is proposed by Luk.

- Once the scan direction is decided, the LEFT edge terminals are brought into the first column.

- Then, for each column the first four steps listed below are repeated.

- In Step 1 the nets TOP(i) and BOT(i) are brought into empty rows.

- In the second step split nets are joined as much as possible to increase the number of free tracks.

- Step 3 comprises of jogging. In Step 3a, as in the case of the greedy channel routing algorithm, trunks of each split net, which have no terminals on the right are brought closer by jogging.

- And in Step 3b, for those nets which have terminals on the right we use the combination of jogging strategies discussed above.

- This procedure is called SWJOG. It divides the routing region into a left p-portion and a right p-portion.

- The jogging strategy to be applied depends upon the location of the column (in left or right p-portion) where the decision is to be made.

- The value of p is between 0 and 1. Below we now enumerate the rules for SWJOG.

  (a) For nets that do not have right side terminals, always perform $Jog_{T/B}$.

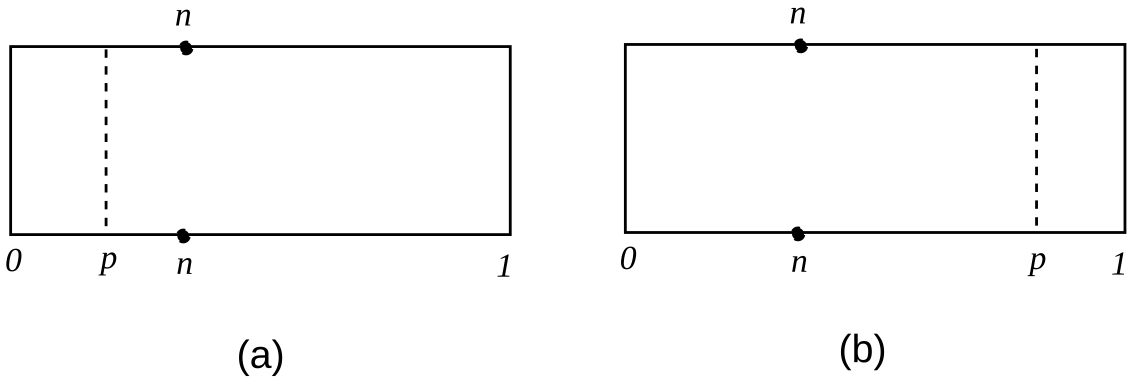
  (b) For nets that have a right side terminal and whose rightmost top/bottom terminal is on the right p-portion of the routing region, perform Jog_R for that net.

  (c) For nets that have a right side terminal and whose rightmost top/bottom terminal is on the left p-portion of the routing region, perform $(Jog_{T/B}; Jog_R)$, that is, $(Jog_{T/B})$ before the last top/bottom terminal and (Jog_R) at and after the last top/bottom terminal.

  (d) The value of p may vary between 0 and 1. If $p = 0$ perform (Jog_R). Obviously, if $p = 1$ perform $(Jog_{T/B}; Jog_R)$. A typical value of p is 0.5.

- In the implementation, a distance dependent threshold scheme is used to avoid excessive jogging. A net is allowed to jog to its target row only if it can be brought to or beyond half-way between the initial position and final target position.

- For nets that have a terminal in RIGHT and whose right-most top/bottom terminal is (a) on right p-portion of routing region perform (Jog_R). (b) on left p-portion of routing region perform $(Jog_{T/B}; Jog_R)$.



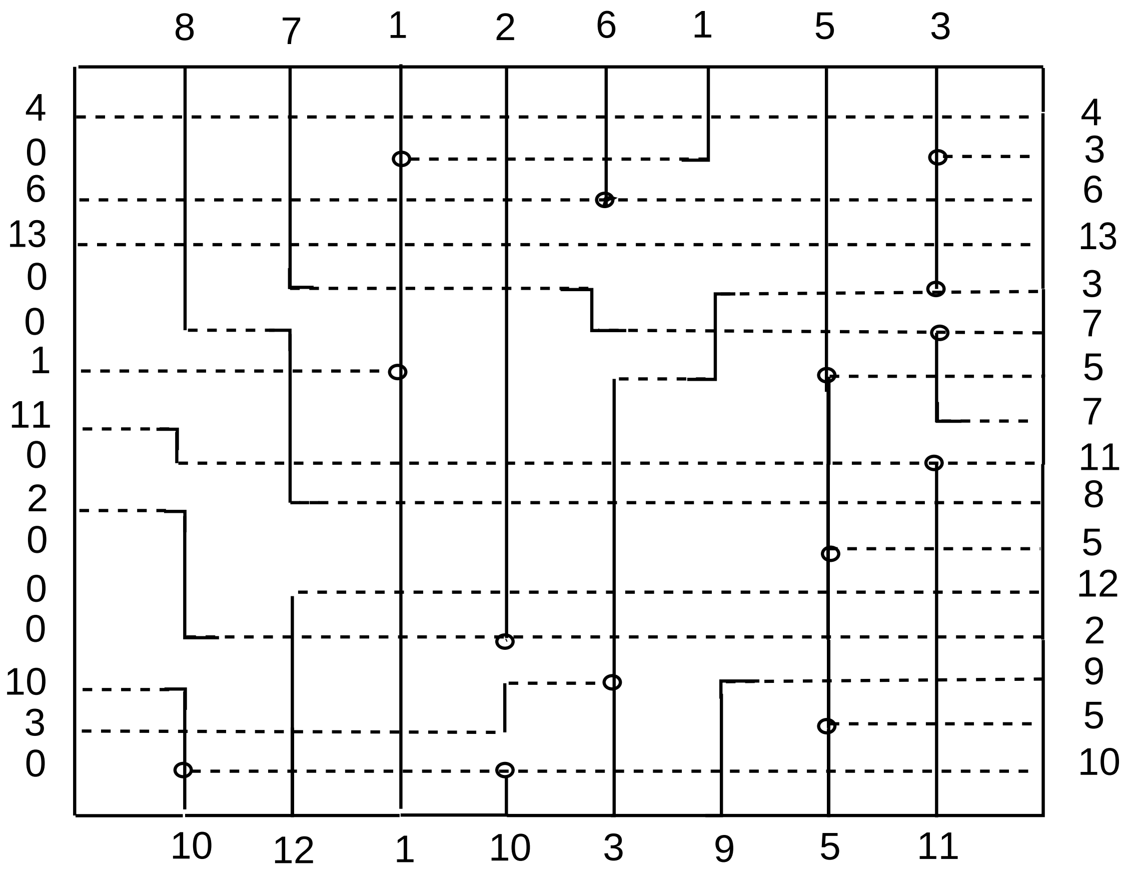
(a) (b)

- In Step 4 for nets that occupy more than one location on the RIGHT, when they get closer to the right edge, these nets are made to fan-out to their final terminal locations.

- Step 5 consists of increasing the number of rows if Step 1 failed.

- And in Step 6, if split nets exist then the number of columns is incremented and split nets are joined as much as possible.

- The complete routed solution of of the earlier illustrated problem is shown in Figure below.



- The time efficiency of the switchbox router is the same as the greedy channel router. The router can be modified to route a region with terminals fixed on any three sides.

Other Approaches and Recent Work

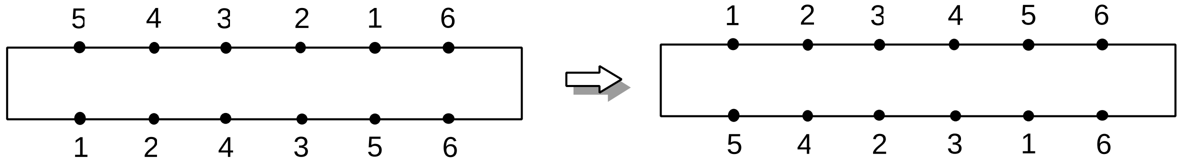
- We now look at some related work in the area of channel and switchbox routing.
- Recent algorithms for both channel and switchbox routing.
- We will look into a channel routing approach proposed by Chaudry and Robinson based on sorting.
- Their approach assumes that wires, in addition to running horizontally and vertically, can also run at 45° and 135°.
- Techniques for multilayer and over the cell channel routing are discussed.

Channel Routing by Sorting

- Without loss of generality the nets in TOP can be assumed to be numbered in sequence $1, 2, \cdots, n$.

- For example the channel routing problem TOP = [5, 4, 3, 2, 1, 6] and BOT=[1, 2, 4, 3, 5, 6] can be also be specified as TOP=[1, 2, 3, 4, 5, 6] and BOT = [5, 4, 2, 3, 1, 6], where the terminal labels in TOP are reordered to be in sequence and corresponding changes are made to the labels in BOT

- See Figure below for equivalent channel routing problems.

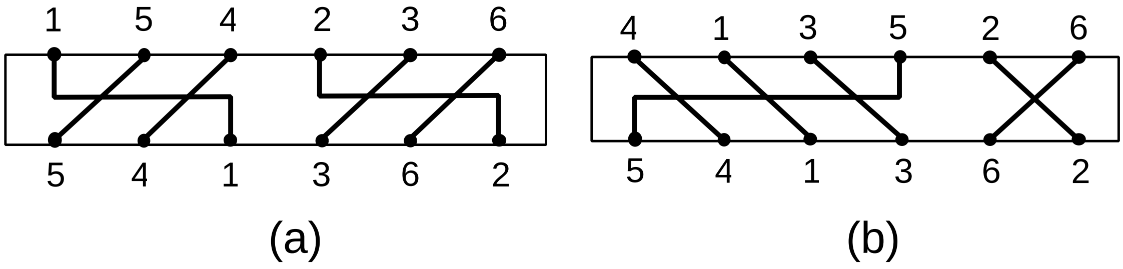


- The problem can also be specified as [5,4,2,3,1,6].

- Then, the nets in BOT are a permutation of the sequence in TOP.

- Two permutations p_i and p_{i+1} are said to be adjacent if the routing problem obtained by assigning p_i to the lower side and p_{i+1} to the upper side of the channel can be routed in one track.

- Possible adjacent permutations and the associated routing are shown in Figure below.



- The solution to the channel routing problem is represented as a series of permutations $\{p_i\}$, $i = 1, 2, 3, \cdots, w$ such that p_1 is the given permutation (BOT) and $p_w = (1, 2, \cdots, n)$ (TOP), and p_i is adjacent to p_{i+1}, for $0 \leq i \leq w$.

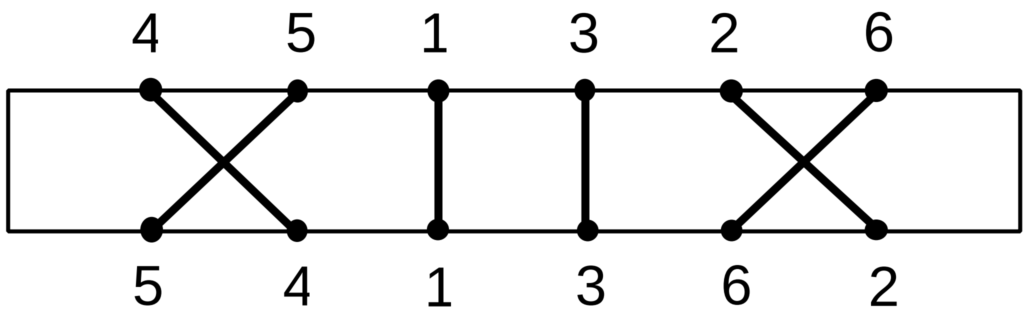
- The channel routing problem then amounts to finding a series of intermediate adjacent permutations $\{p_i\}$ such that the number of permutations w is minimized.

- We now present the basic idea behind two routers, namely *Swap-router* and *Sort-router*. These routers are based on permutations and sorting.

Swap Router

- In swap router, two nets that have adjacent terminals in the wrong order are interchanged.

- These nets can be connected using X routing as shown in Figure below. Note that it is assumed that connections can run at 45° and 135°.



- A series of adjacent permutations can be built using only X routing.

- This corresponds to factoring the permutation as a product of transpositions.

- Routing is done from bottom to top. If $(a_1, a_2, \cdots, a_n)$ is the bottom permutation, we compare a_i and a_{i+1} for $i = 1, 3, 5, \cdots, n$; and swap the terminals if $a_i > a_{i+1}$.

- In the next step the process is repeated for $i = 2, 4, 6, \cdots, n$.

- The above two steps are repeated until all the terminals are in the correct order.

- Since two nets cross only once if their terminals are not in order the routing obtained by this swap-router is a minimal crossing solution.

- Bounds on the channel width are obtained in terms of *span* number. The span of a terminal in a permutation is the difference of the terminal number and its position in the permutation.

- For example, in the permutation (5,4,1,3,6,2), number 1 has a span of -2, number 2 has a span of -4, number 3 has a span of -1, and so on.

- The span number tells us how far the number is from its correct position.

- Since in each step a net moves by at most 1 column, a net with span of y will require y steps. It can therefore be concluded that

$$number\ of\ steps \geq \max(|span_i|), \quad 1 \leq i \leq n \tag{1}$$

- Clearly, channel width can be reduced by removing the restriction of moving only one column at each step.

Example

Determine the number of tracks required to route the channel instance specified by [5,4,6,2,1,3] using the swap-router.

Solution

- For the problem under consideration, TOP=[1, 2, 3, 4, 5, 6], BOT=[5, 4, 6, 2, 1, 3].

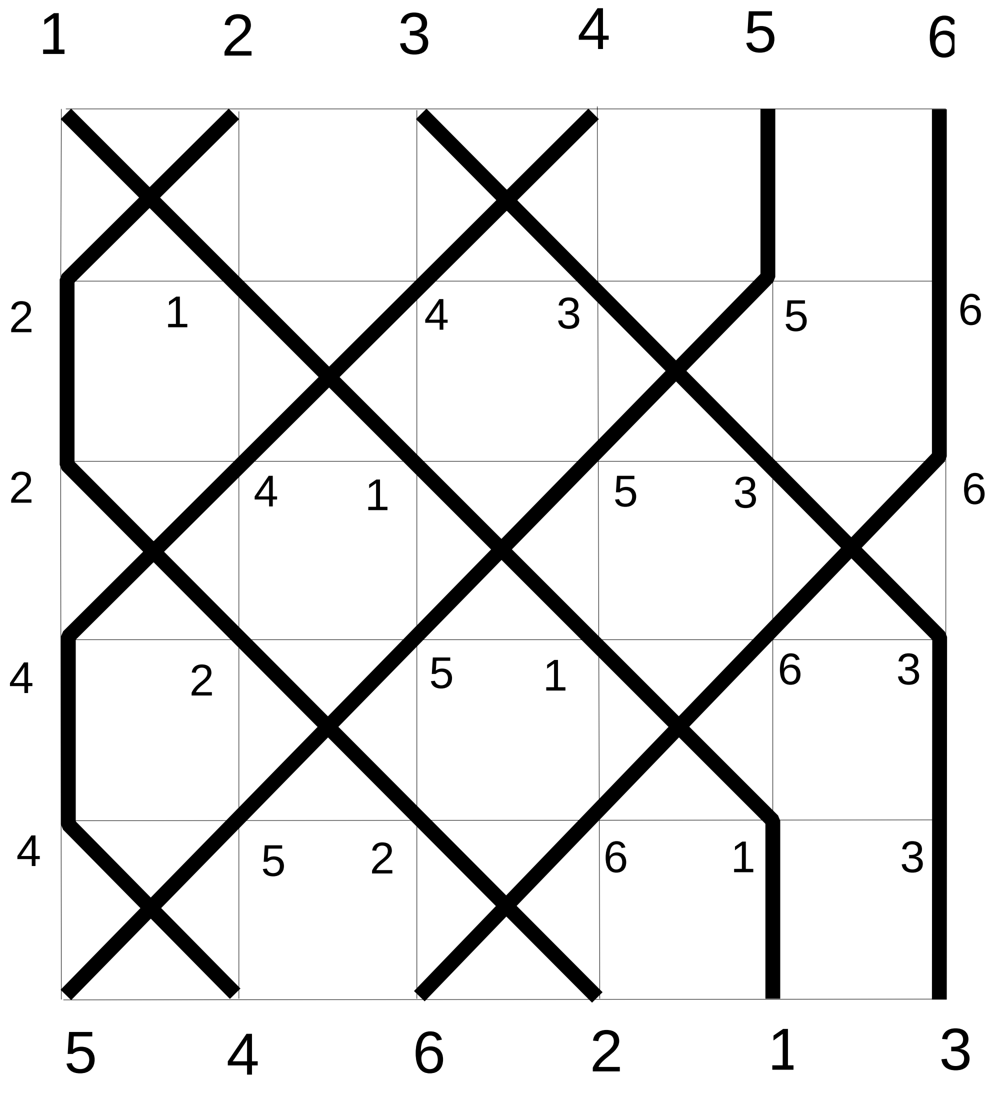
- We begin from bottom. In the first pass we compare a_i and a_{i+1} for $i = 1, 3, 5, \cdots$.

- This leads to a permutation (4,5,2,6,1,3). In the next pass, we repeat the above, but for $i = 2, 4, 6, \cdots$.

- The resulting permutation is (4,2,5,1,6,3). In the third pass, again we apply swapping for $i = 1, 3, 5, \cdots$. and the resulting permutation is (2,4,1,5,3,6).

- The last two permutations are (2,1,4,3,5,6) and (1,2,3,4,5,6) respectively.

- The routed solution is illustrated in Figure below.



Sort Router

- From the previous discussion it is clear that any sorting algorithm based on exchanges can be easily converted to a channel router.

- An algorithm based on *bubble-sort* is presented by Chaudry and Robinson.

- The steps of *bubble-sort* swap a pair of numbers only once if they are in the wrong order. Therefore, as in the case of *swap-router*, the *sort-router* always produces a minimal crossing solution.

- Since in one pass of the *bubble-sort* at least one number moves to its final place, it would require at most n steps to sort the n numbers. Thus the channel width will be $\leq n$, where n is the number of nets.

- Here again 45° routing is allowed.

- We will illustrate this process with an example.

Example

Apply the sort-router based on bubble sort to the channel routing problem where TOP=[1, 2, 3, 4, 5, 6], and BOT=[5, 4, 6, 2, 1, 3].

Solution

- The problem instance to be routed is (5, 4, 6, 2, 1, 3).

- We now have to sort these numbers. Each intermediate step of sorting will produce an adjacent permutation and will require one track.

- The numbers in bubble-sort can be either scanned from left to right or vice-versa; and the number of passes required to complete sorting will depend on the direction of scan.

- If the numbers are scanned from left to right we call this a *right-step*, and if they are scanned from right to left we call this a *left-step*.

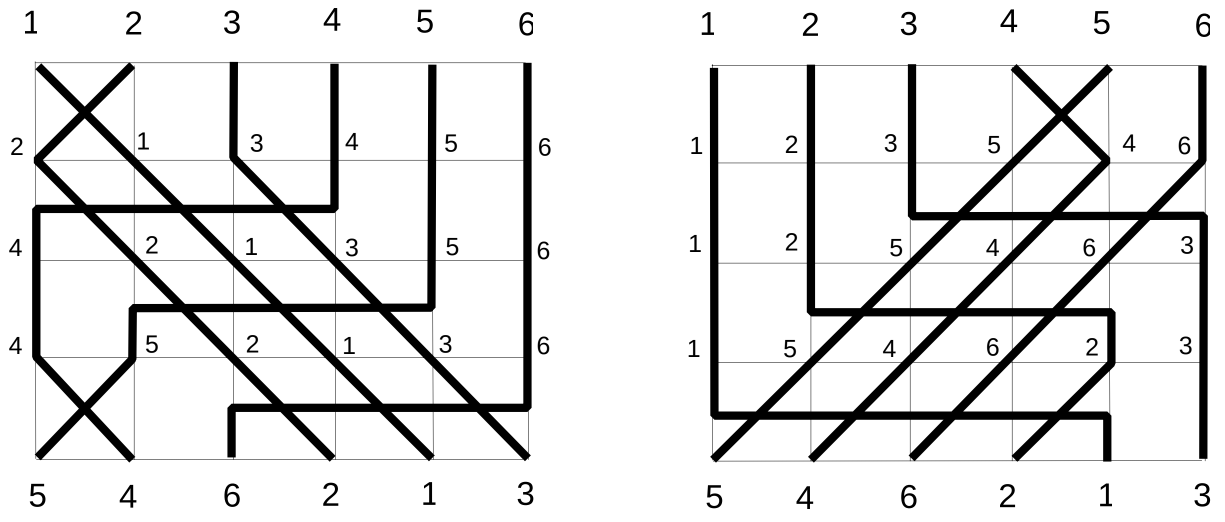
- The number of steps required to route varies depending on the direction of scan. The intermediate permutations for both the *right-step* and *left-step* are shown below as *R-step* and *L-step* respectively.

$$R - step = \left| \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 3 & 4 & 5 & 6 \\ 4 & 2 & 1 & 3 & 5 & 6 \\ 4 & 5 & 2 & 1 & 3 & 6 \\ 5 & 4 & 6 & 2 & 1 & 3 \end{array} \right|$$

$$L - step = \left| \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 5 & 4 & 6 \\ 1 & 2 & 5 & 4 & 6 & 3 \\ 1 & 5 & 4 & 6 & 2 & 3 \\ 5 & 4 & 6 & 2 & 1 & 3 \end{array} \right|$$

- Coincidentally, in this example, the number of intermediate permutations for both the right step and the left step are the same.

- The channel routing solution of the above problem for both scan directions are given in Figure below: (a) Left to right scanning. (b) Right to left scanning.



(a)

(b)

- Details of choice of optimal step-type, and extensions to multiterminal nets, and multilayer routing are available in literature.

Over-the-cell Channel Routing

- Another extension to the classical channel routing problem is *over-the-cell* channel routing.

- This method is employed when there are at least two layers in the routing channel, and one routing layer over the channel.

- Certain nets can be partially or totally routed on one side over the channel using the single available layer. Then, the remaining net segments are chosen for routing.

- Therefore, a common approach to over-the-cell channel routing is to divide the problem into three steps, namely, (1) routing over the channel, (2) choosing the net segments, and (3) routing within the channels.

- The third step can be accomplished easily using one of the conventional techniques discussed in Chapter 7 of the text.

- Cong and Liu showed that the first step can be formulated as the problem of finding a maximum independent set of a circle graph and can be solved optimally in quadratic time.

- In this step a row of terminals are routed on one side of the channel using a single routing layer.

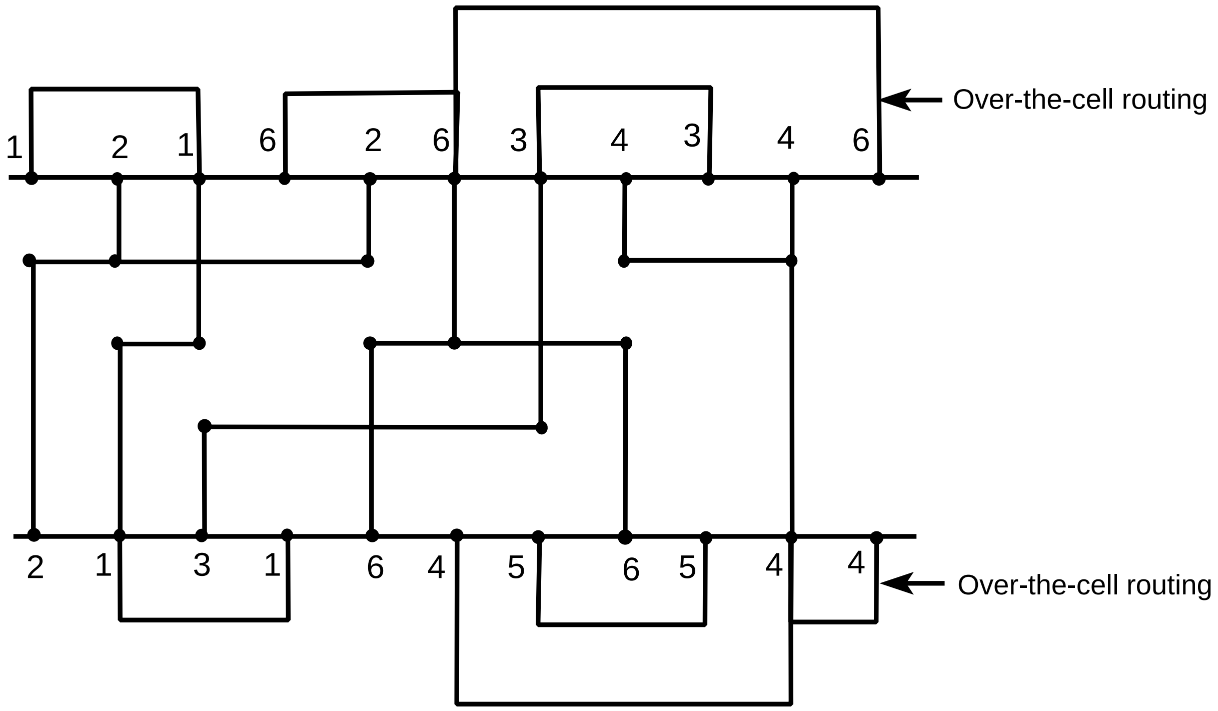
- The result is that the number of hyperterminals are minimized.

- Cong and Liu called this problem *multiterminal single-layer one-sided routing problem* (MSOP).

- The second step is formulated as the problem of finding a minimum density spanning forest of a graph.

- An efficient heuristic which produces satisfactory results is proposed.

- A channel routing problem and its over-the-cell solution are illustrated in Figure below.



Summary

- In this session, we presented the problem of channel routing.

- Graph theoretic approaches to solve the channel routing problem were presented.

- The algorithm which performs splitting of nets, known as doglegging was described.

- We also presented two algorithms due to Kuh and Yoshimura.

- The first uses merging of nodes so that the longest path length after merging is minimized as much as possible.

- The second algorithm achieves longest path minimization through matching techniques.

- Both techniques report better results than the dogleg algorithm.

- Further, the greedy router, which, unlike the above mentioned methods, routes the channel column by column, was presented.

- A modification of the greedy heuristic to route switchboxes was also presented.

- Heuristics that employ sorting and swapping to route channels, and ‘Over-the-cell channel routing’, were presented in the section dedicated to ‘other approaches’.