Elective II: VLSI Design

Code: CISM 402

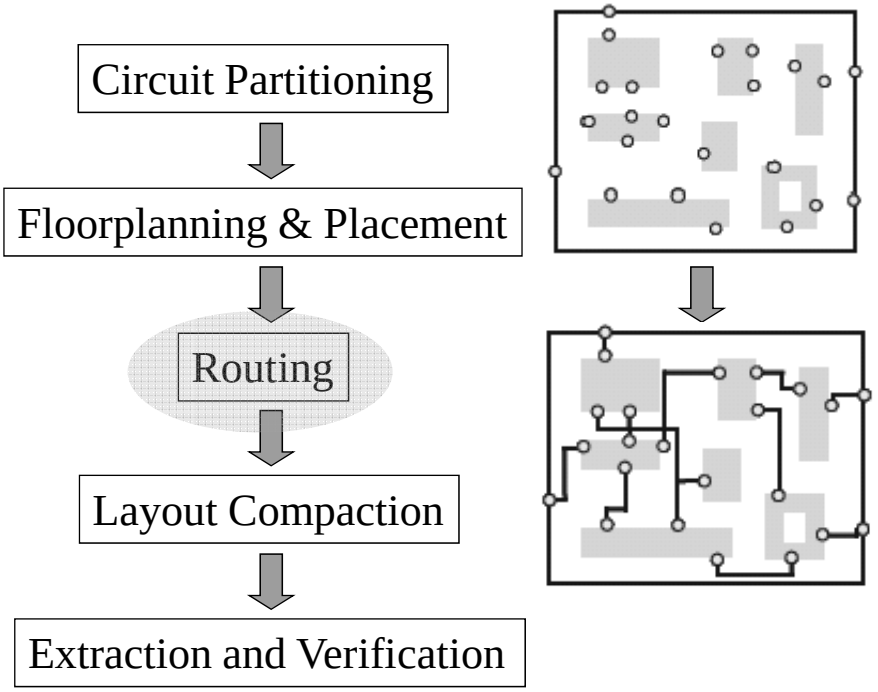
Pritha Banerjee

Courtsey for slides: Debasis Mitra, NIT, Durgapur

Routing

- Books:
  - Chapter 6,7 of Naveed A. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers

Physical Design Flow

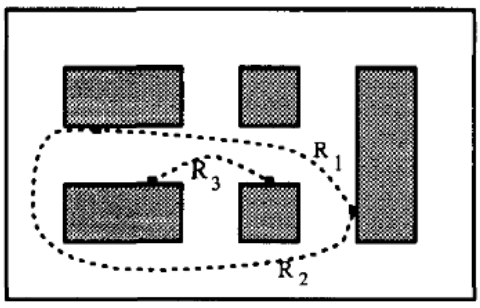


Routing

The process of finding the geometric layouts of all the nets

usually routing of signal nets and global nets
(e.g. power, ground, clock) are considered separately

Applied after the exact locations of blocks and pins are determined through placement phase



Problem Formulation

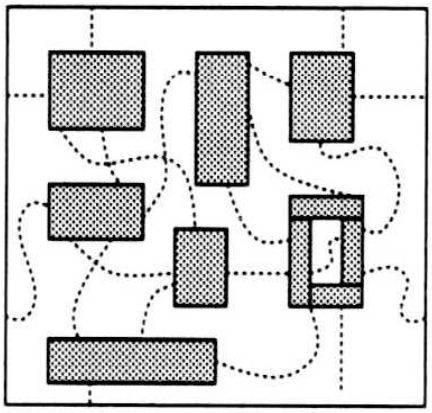
- Input:
  - Netlist
  - Timing budget for, typically, critical nets
  - Locations of blocks and locations of pins and I/O pads
  - A set of design rules for manufacturing process, such as resistance, capacitance, and the wire/via width and spacing of each layer
- Output:
  - Geometric layouts of all nets
- Objective:
  - Minimize the total wire length, the number of vias, or just completing all connections without increasing the chip area
  - Each net meets its timing budget
  - Satisfy design rules

Kinds of Routing

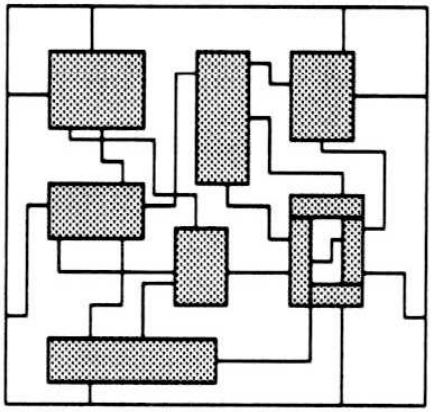
- Routing: NP-complete problem.
- Global Routing
  - Partition routing regions into tiles and generates a ‘loose’ route for each net
  - Assigns a list of routing regions to each net without specifying the actual geometric layout of wires
- Detailed Routing
  - Finds the actual geometric layout of each net within the assigned routing regions

Kinds of Routing

- Global Routing
- Detailed Routing



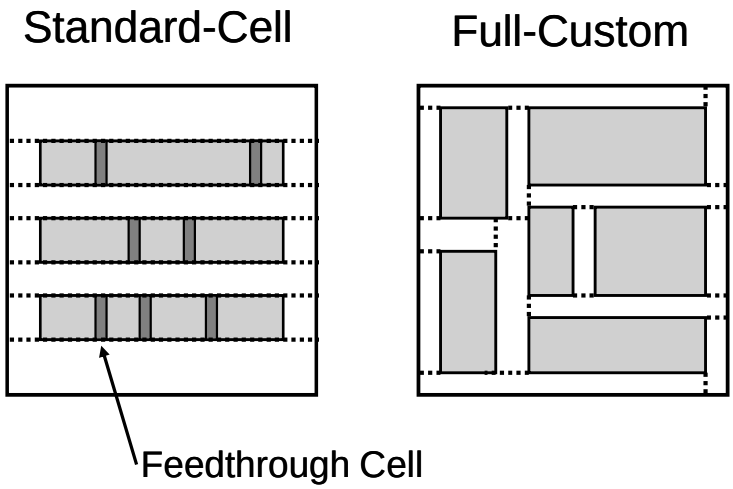
Global Routing



Detailed Routing

Routing Regions

Space not occupied by the blocks can be viewed as a collection of regions



Global Routing

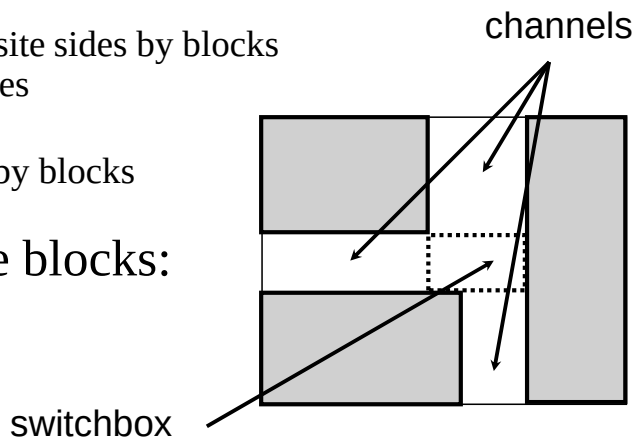
Divided into 3 phases:

1. Region definition
2. Region assignment
3. Pin assignment to routing regions

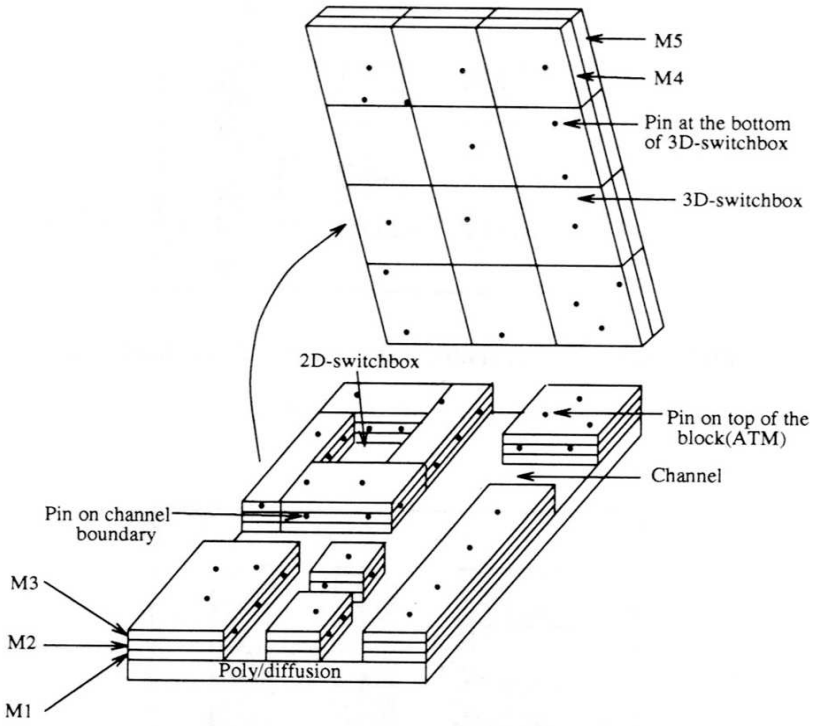
Region Definition

Routing area is divided into routing regions of simple shape (rectangular):

- Routing regions between blocks:
  - Channel:
    - Bounded in two opposite sides by blocks
    - Pins on 2 opposite sides
  - 2-D Switchbox:
    - Bounded on all sides by blocks
    - Pins on 4 sides
- Routing regions above blocks:
  - 3-D Switchbox:
    - Pins on all 6 sides

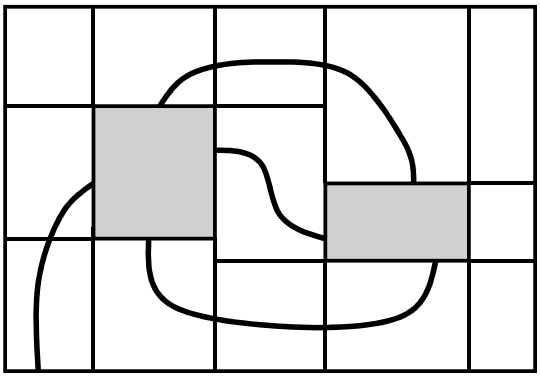


Region Definition



Region Assignment

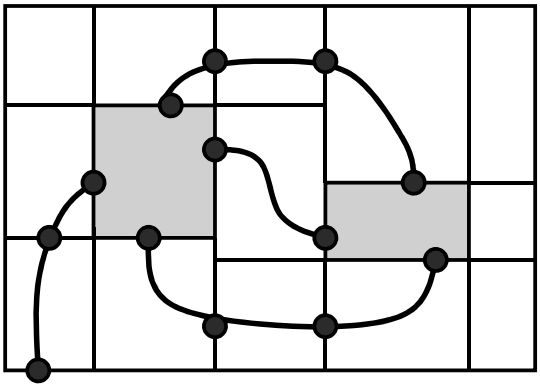
- Assign routing regions to each net
- Need to consider timing budget of nets and routing congestion of the regions
- Each region has a capacity : # of nets that can be routed through it



- Channel Capacity: fn of # of layers(l), height(h) of channel, wire width(w), wire separation(s)= l*h / (w+s)

Pin Assignment

Assign pins on routing region boundaries for each net
(Prepare for the detailed routing stage for each region)



Approaches for Global Routing

Sequential Approach:

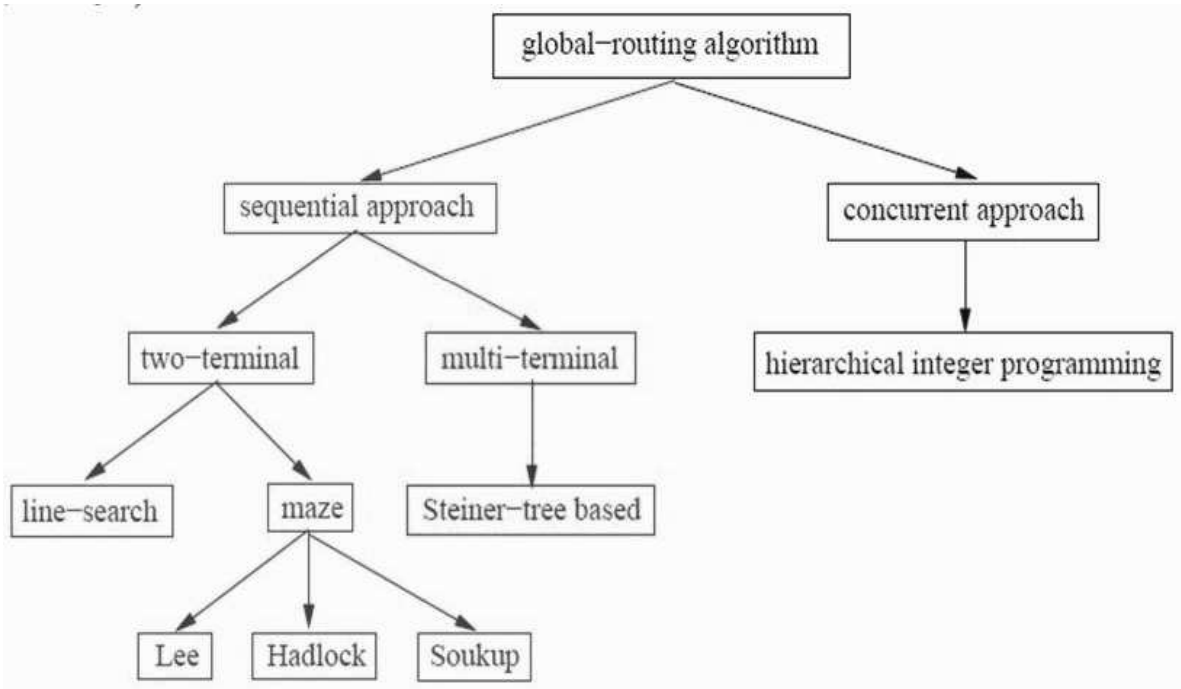
- Route the nets one at a time according to priority
- Priority is assigned depending on factors like criticality, estimated wire length, etc
- If further routing is impossible because some nets are blocked by nets routed earlier, apply Rip-up and Reroute technique.
- This approach is much more popular

Concurrent Approach:

- Consider all nets simultaneously
- Can be formulated as an integer program

Algorithms for Global Routing

Classification:



Extraction & timing analysis

- After global routing and detailed routing, information of the nets can be extracted and delays can be analyzed.

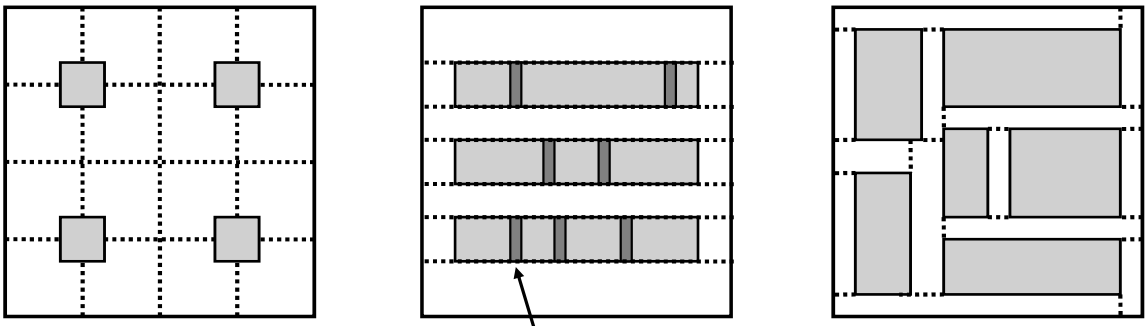
- If some nets fail to meet their timing budget, detailed routing and/or global routing needs to be repeated.

Routing regions for different design styles

Gate-Array
- Fixed channel Capacity
- Routability
- Min wirelength

Standard-Cell
- Capacity, location fixed (FT)
- Min Height of channel

Full-Custom
No restriction



Feedthrough Cell

Graph Model for area routing

Grid graph model:

Checker board model:

Channel Intersection graph model:

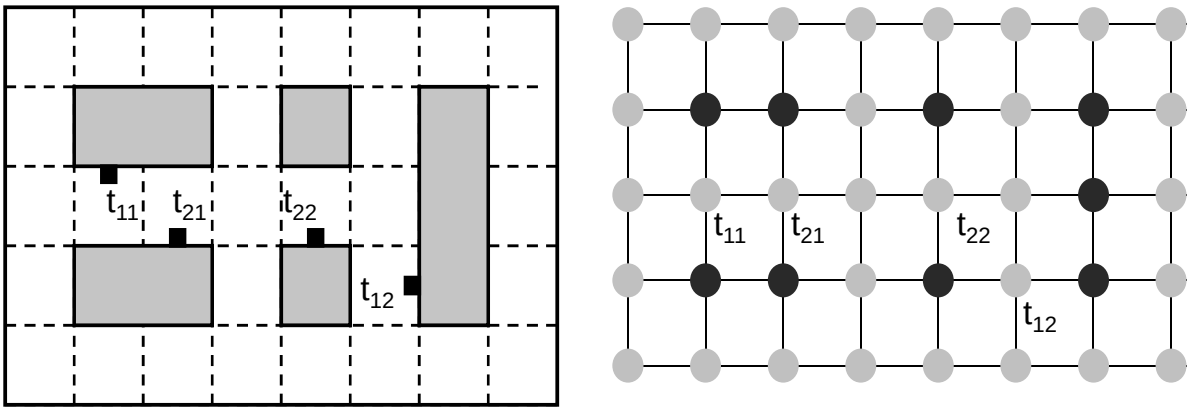
Representation of Layout in Routing

Grid graph model:

- A layout is considered to be a collection of unit side squares arranged in a *h* x *w* array
- Each such square is represented by a vertex
- Vertices are labeled either blocked or unblocked
- There is an edge between two vertices if the corresponding squares are adjacent.

Representation of Layout in Routing

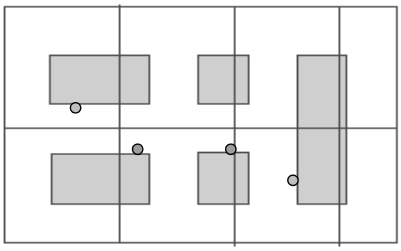
Grid graph model:

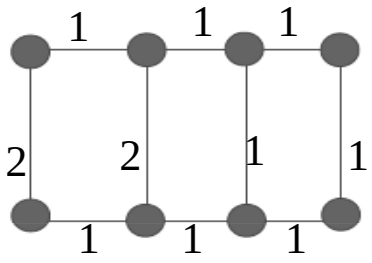


Capacity, length (weight) of edge = 1, blocked cells : 0
Find path between terminals in a graph

Representation of Layout in Routing

Checker Board model:

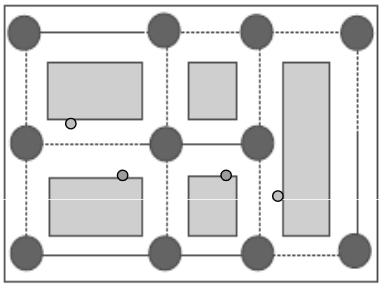


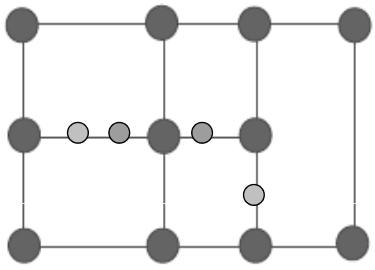


- Checker Board Graph is generated in a manner similar to the grid graph except that the superimposed grid is a “coarse grid.”

Representation of Layout in Routing

Channel Intersection graph model:





- Channels are represented as edges
- Channels intersections are represented as vertices
- Edge weight represents channel capacity

Steiner Tree

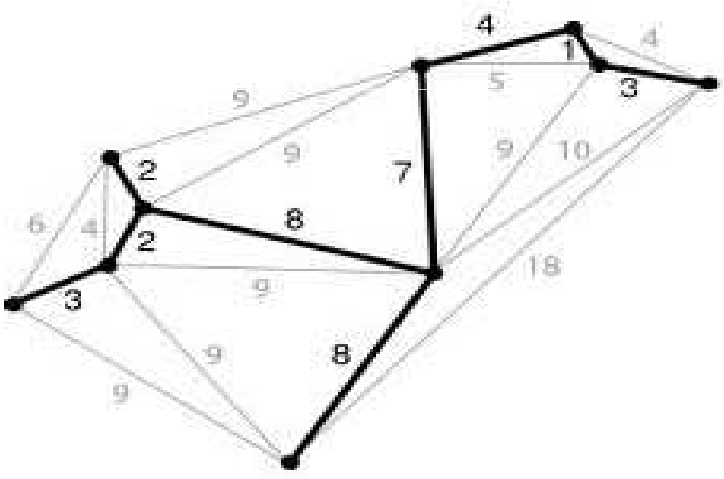
Minimum Spanning Tree:

– Given a connected undirected edge-weighted graph, a spanning tree (a subgraph which is a tree and connects all the vertices together) of minimum total weight

Steiner Tree

Minimum Spanning Tree:

– Given a connected undirected edge-weighted graph, a spanning tree (a subgraph which is a tree and connects all the vertices together) of minimum total weight



Steiner Tree

Minimum Steiner Tree:

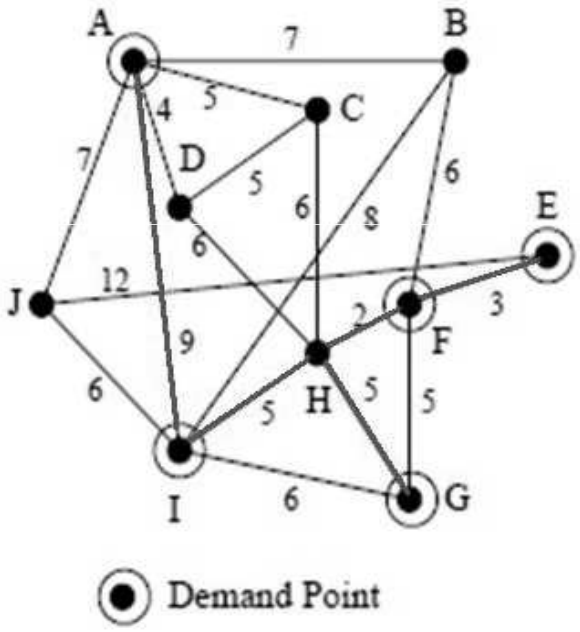
- Given a connected undirected edge-weighted graph G(V,E), and a subset D $\subseteq$ V, select a subset V' $\subseteq$ V such that D $\subseteq$ V' and V' induces a tree of minimum total weight over all such trees

  The set D is refereed to as the set of demand points

  The set V' – D is referred to as the set of ***Steiner points***

Steiner Tree

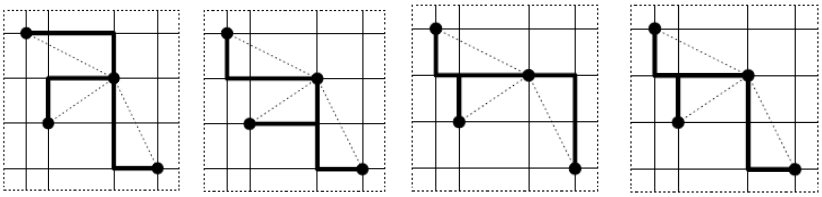
Minimum Steiner Tree:

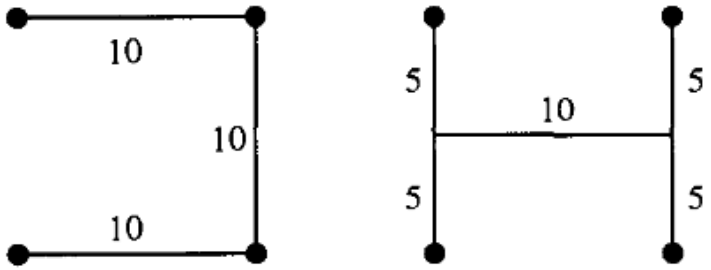


Steiner Tree for multi terminal nets

Rectilinear Steiner Tree:

– Edges are restricted to be rectilinear





•Different rectilinear Steiner trees constructed from Minimum Spanning tree

•Same length for two Steiner tree but different diameter; go for same smaller diameter

•Diameter of ST: the longest path between any two terminal in steiner tree

Maze Routing Algorithms

– Used to find a path between a pair of points (source and the destination), in a planar rectangular grid graph without using any blocked vertex

– The process of finding a path begins with the ***exploration*** phase

  – Several path start at the source, and are expanded until one of them reaches the target

– Finally, the path is identified by ***retracing***

  – Once the target is reached, the vertices are retraced to the source to identify the path

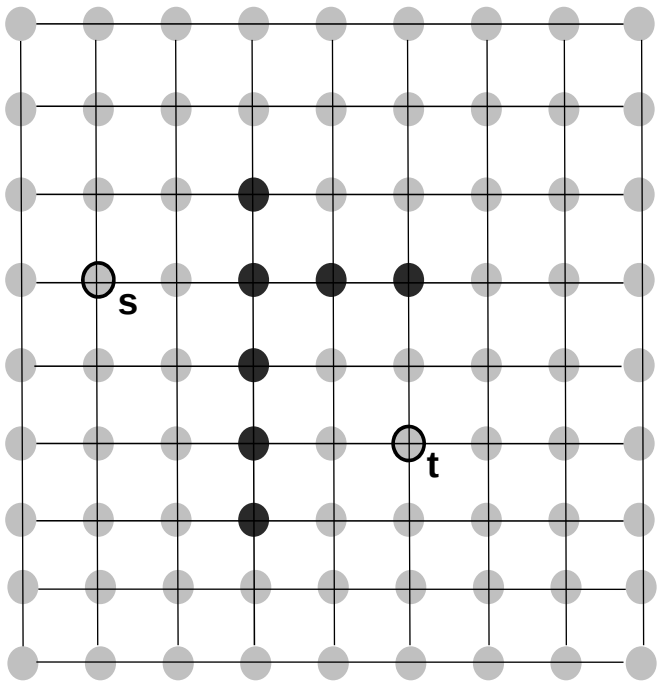
Lee's Algorithm

Basic Idea:

- Exploration phase is an improved version of BFS
- Can be visualized as a wave propagating from the source
  - The source is labeled '0'
  - Every unblocked vertex adjacent to the source is marked with a label '1'
  - Then every unblocked vertex adjacent to vertices with label '1' is marked with a label '2' and
  - So on..
  - The process is continued until the target vertex is reached or no further expansion of the wave can be carried out
- Once the target is reached, the vertices are retraced from the target to the source to identify the path

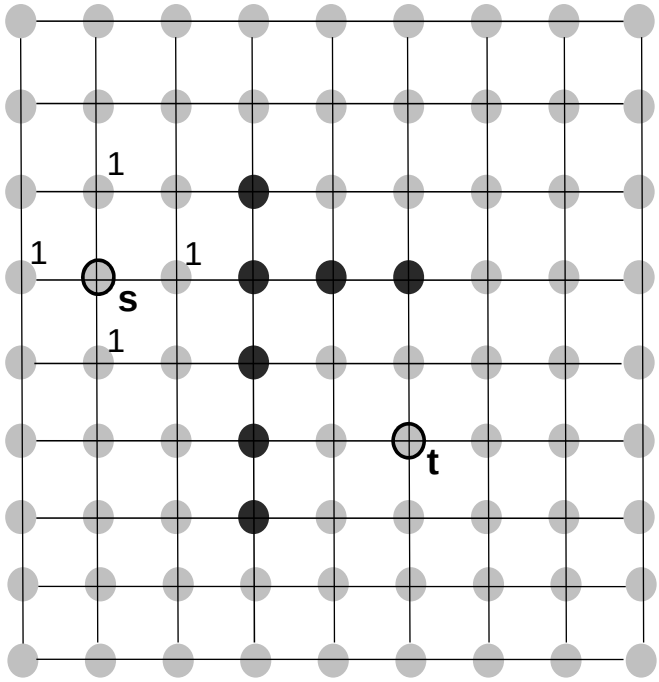
Lee's Algorithm

Illustration:



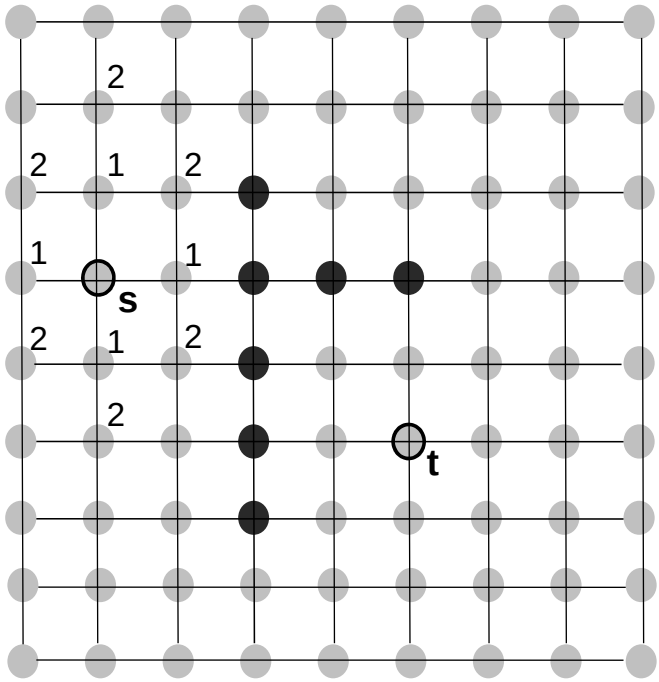
Lee's Algorithm

Illustration:



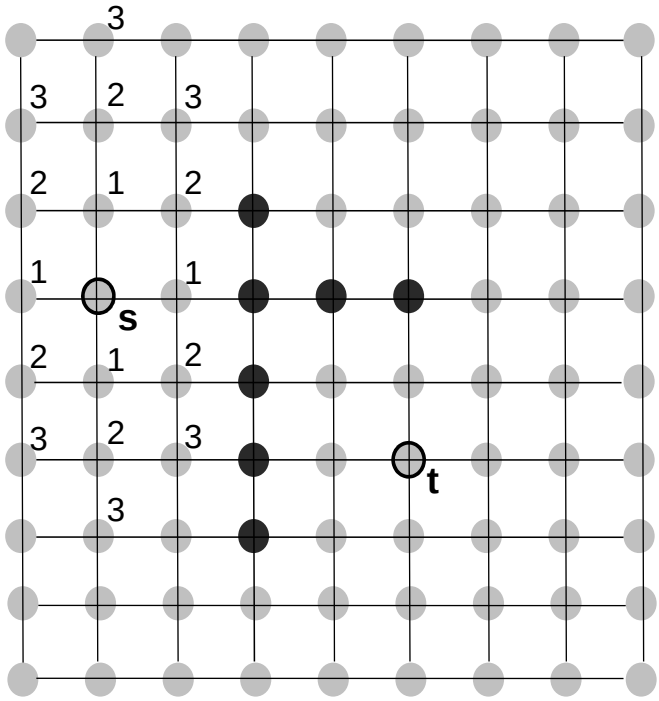
Lee's Algorithm

Illustration:



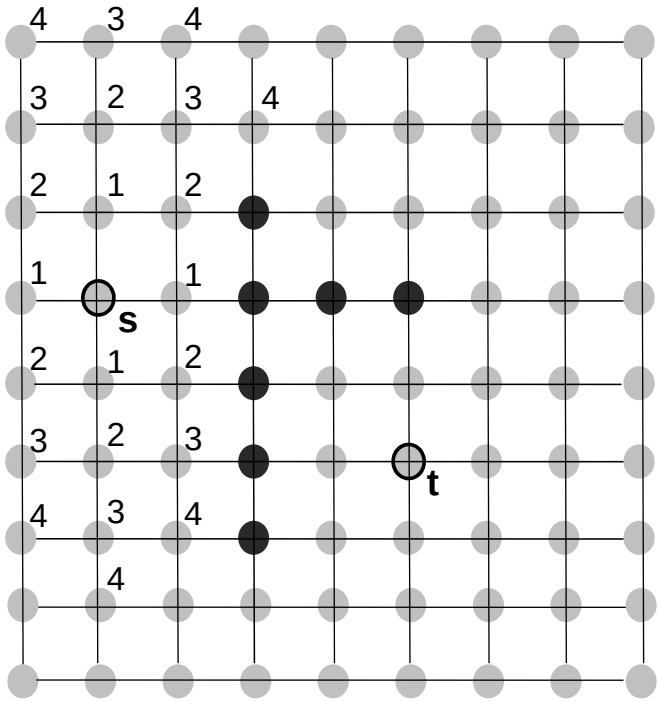
Lee's Algorithm

Illustration:



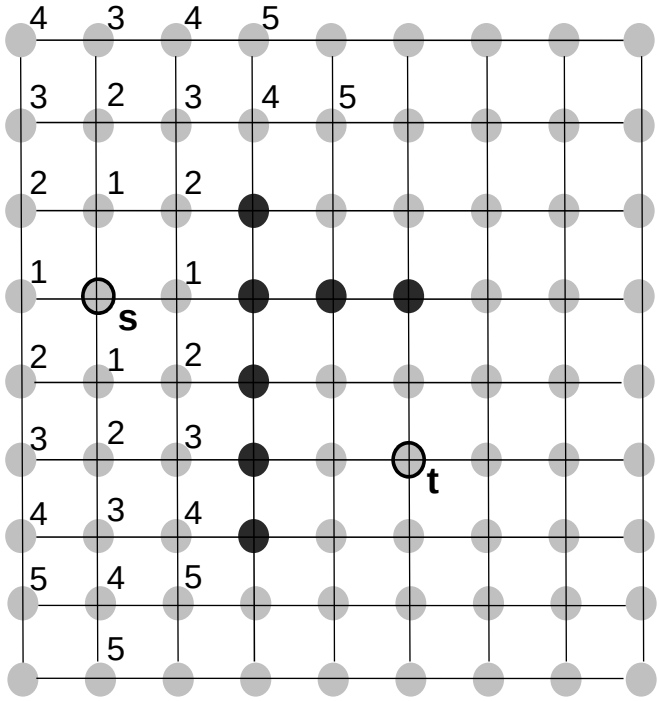
Lee's Algorithm

Illustration:



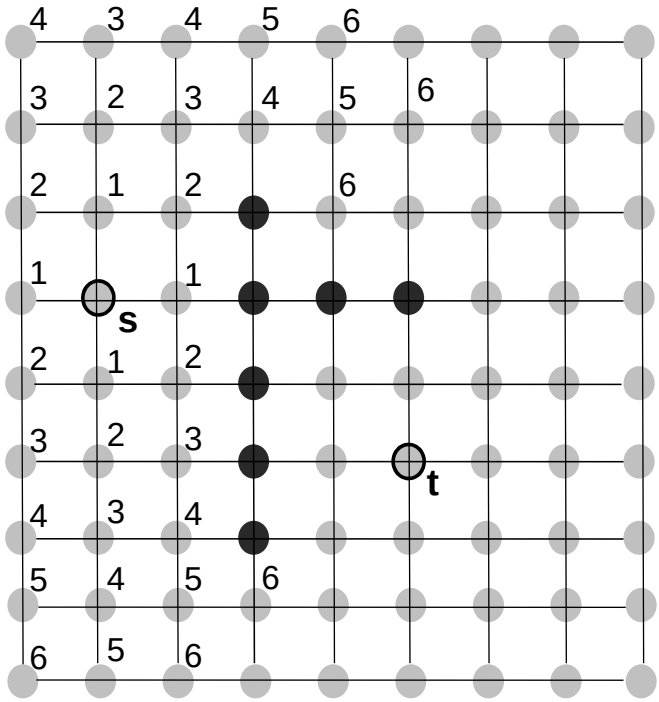
Lee's Algorithm

Illustration:



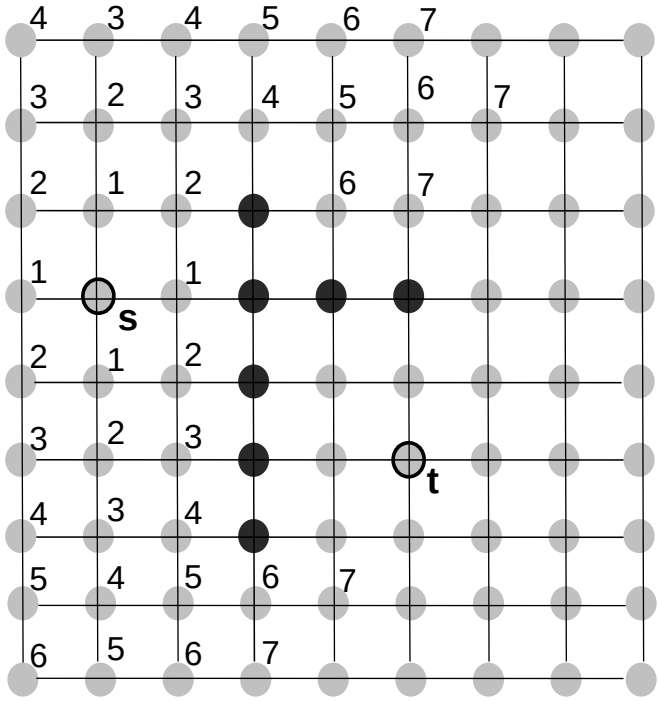
Lee's Algorithm

Illustration:



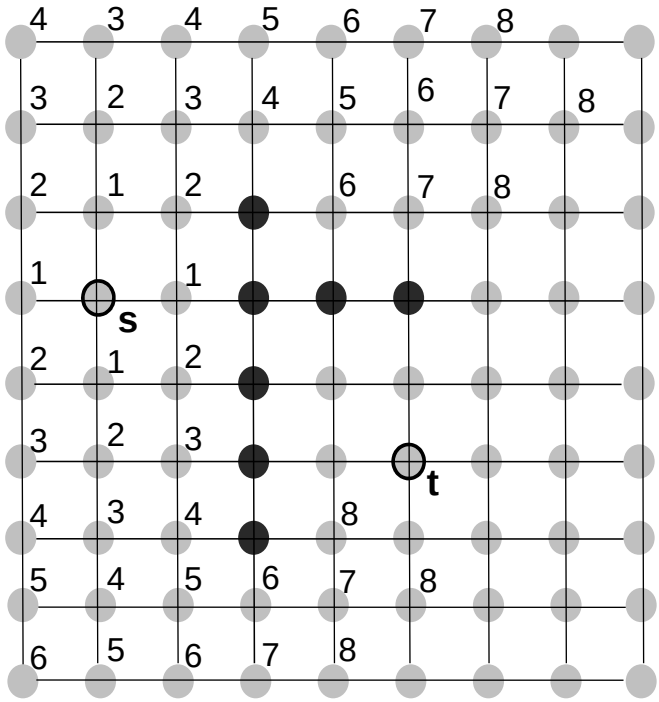
Lee's Algorithm

Illustration:



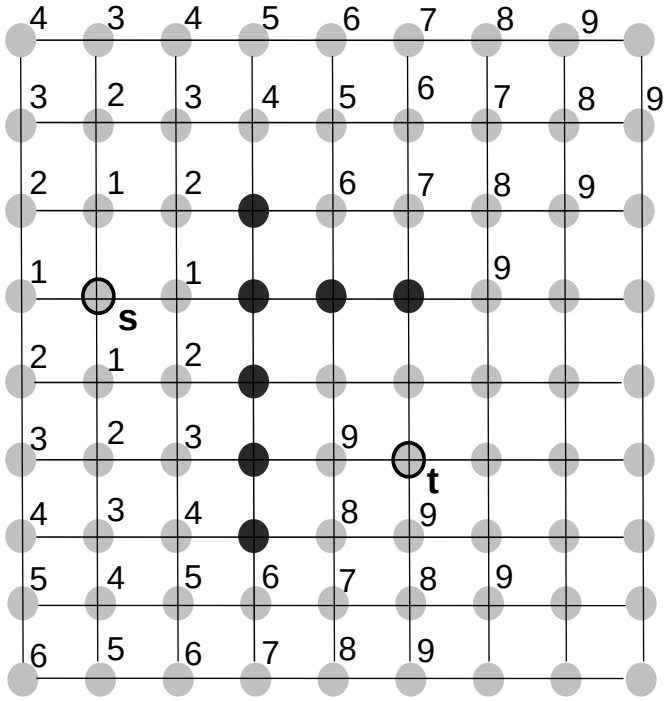
Lee's Algorithm

Illustration:



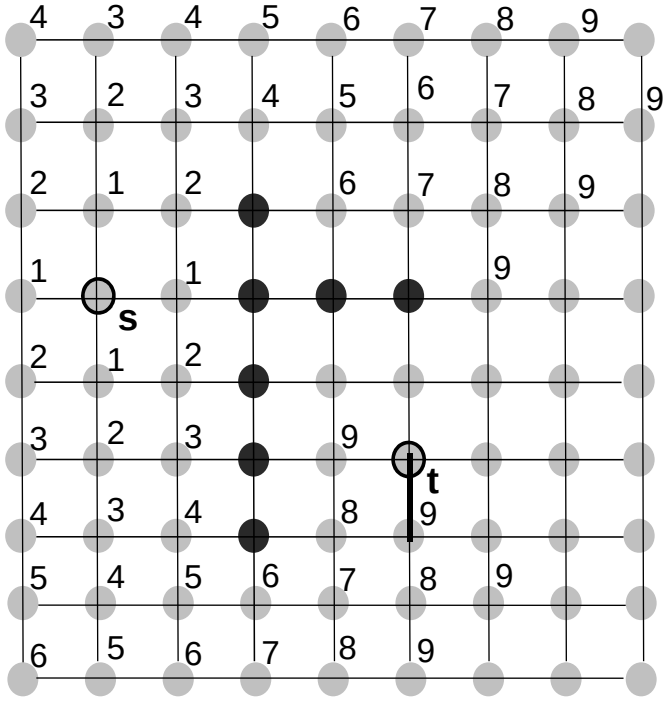
Lee's Algorithm

Illustration:



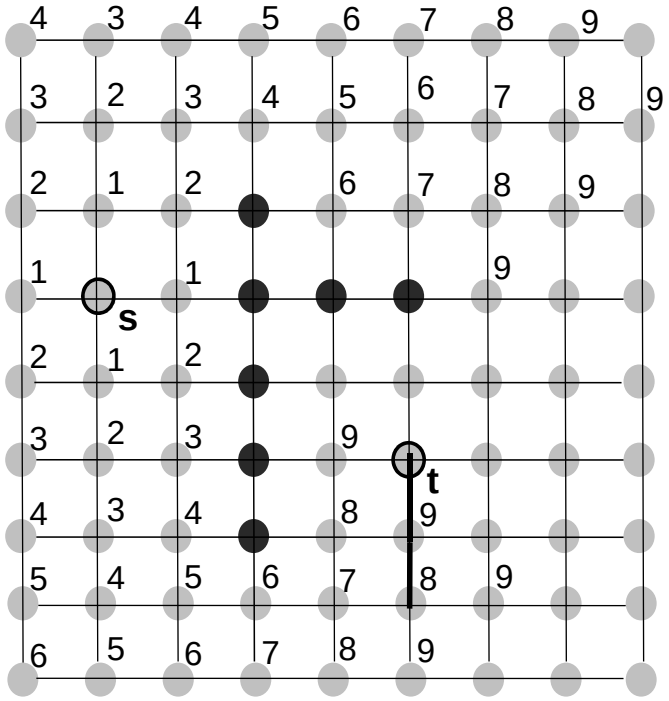
Lee's Algorithm

Illustration:



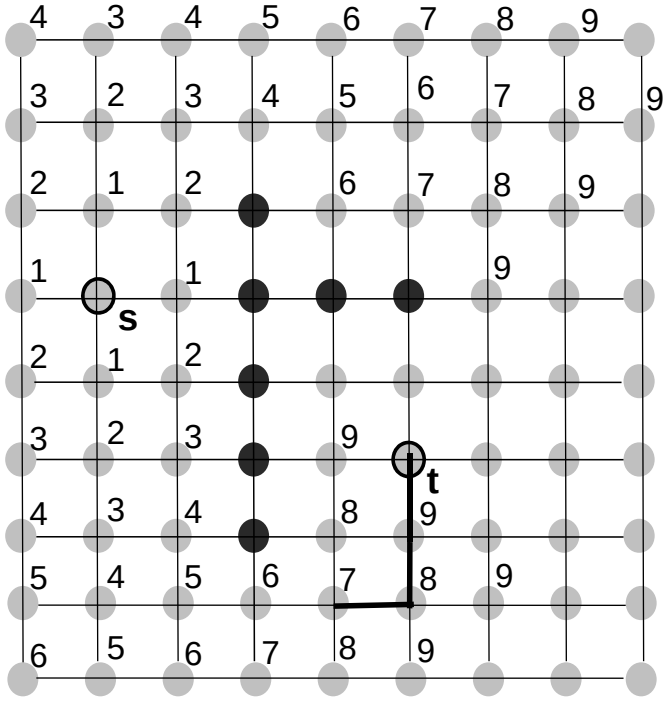
Lee's Algorithm

Illustration:



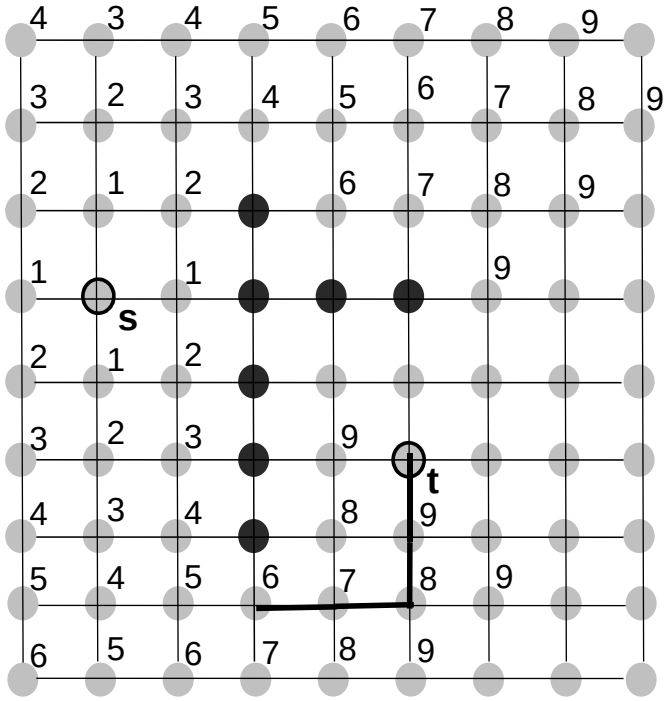
Lee's Algorithm

Illustration:



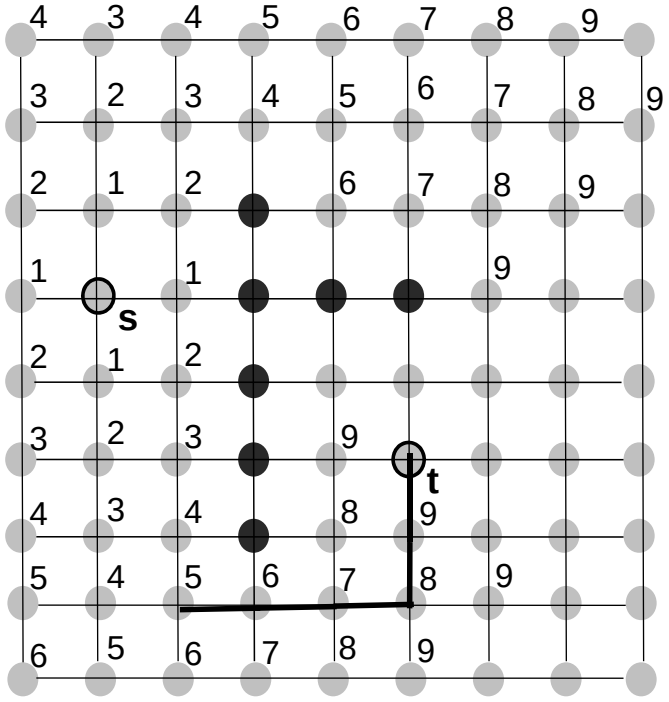
Lee's Algorithm

Illustration:



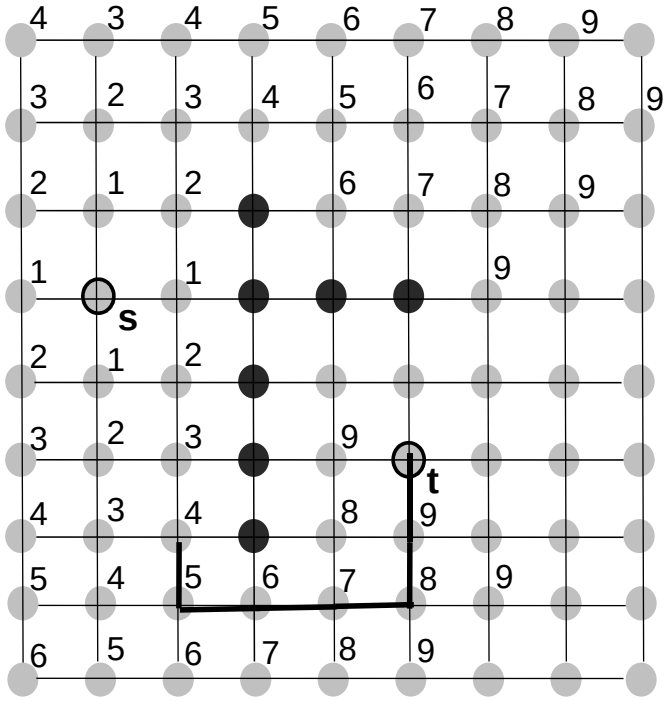
Lee's Algorithm

Illustration:



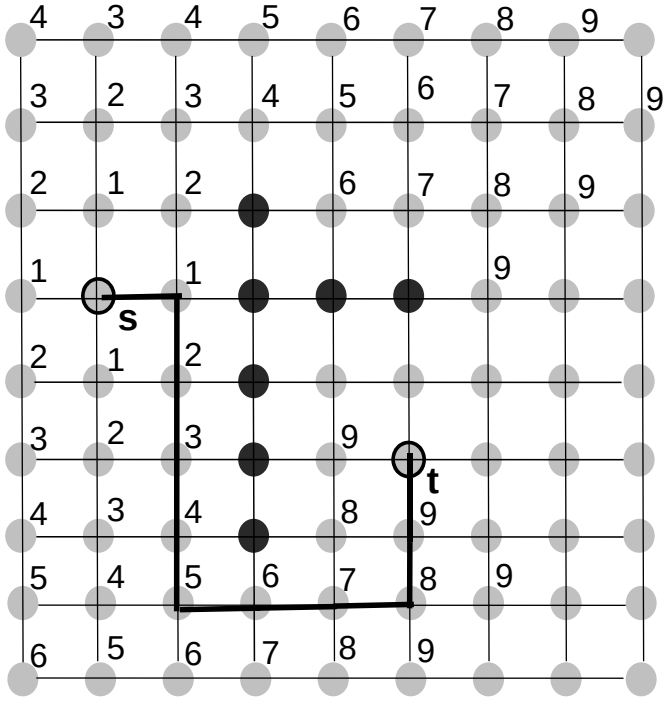
Lee's Algorithm

Illustration:



Lee's Algorithm

Illustration:



Lee's Algorithm

Time and Space Complexities:

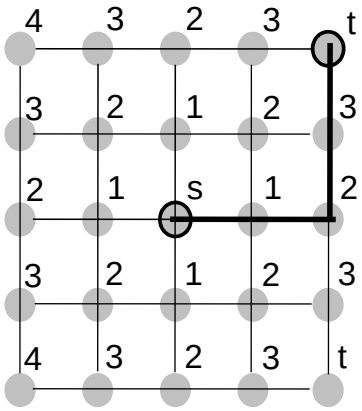
- For a grid structure of size $h \times w$:
  - Time per net = O(h*w)
  - Space = O(w*h log (h*w))
    - Labels can go 1,2,3… h x w
    - O(log (h*w)) bits are needed to store the label of each vertex
    - Another bit is needed per vertex to indicate whether it is blocked
- For a 4000×4000 grid structure:
  - 24 bits per label
  - Total 48 Mbytes of memory!

Lee's Algorithm

Worst case:

All the vertices have to be scanned before the target is reached

Occurs when the source is located at the center and the target is located at a corner



Lee's Algorithm

Strengths:

- Guaranteed to find a path between the source and the target, if one exists
  - Due to the Breadth first nature of the search
- The path is guaranteed to be the shortest path between them

Weakness:

- Requires a large amount of storage space
- Performance degrades rapidly with the increase in grid size

Lee's Algorithm

Improvements:

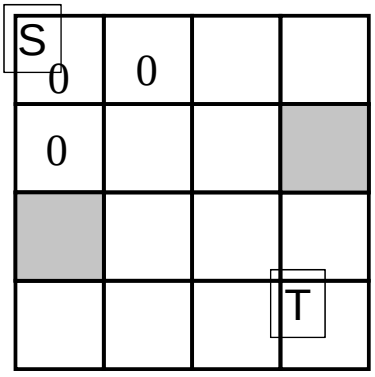
- Improvement on memory:
  - Aker's Coding Scheme
- Improvement on run time:
  - Starting point selection
  - Double fan-out
  - Framing
  - Soukup's Algorithm
  - Hadlock's Algorithm

Aker's Coding Scheme

- Labels are needed during the retrace phase
- Only two types of neighbors of a vertex need to be distinguished
  - Vertices toward the target and vertices toward the source (to identify forward or backward direction)
- The vertices in wavefront *L* are always adjacent to the vertices in wavefront *L-1* and *L+1*
  - i.e., there are only two possible labels for neighbors of each vertex labeled *i*, which are, *i*-1 and *i*+1
- Hence, the sequence of labels should be such that the predecessor of any wavefront is labeled differently from its successor
- So, is there any other sequence (except 1,2,3,4,….) having this property?

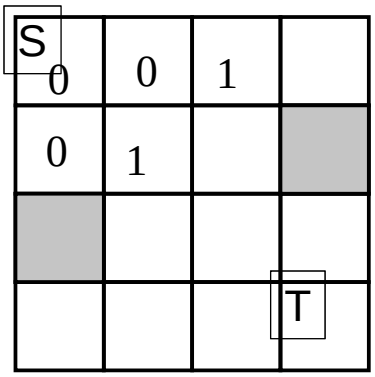
Aker's Coding Scheme

- 0,0,1,1,0,0,1,1,0,0,……
- Each scanned vertex is either labeled '0' or '1'
- One bit (independent of grid size) is enough



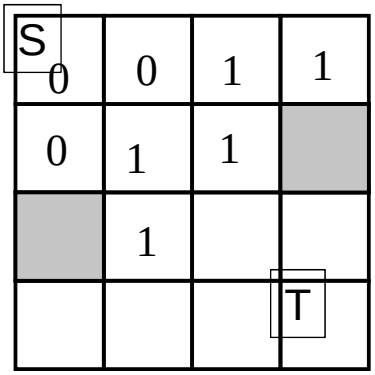
Aker's Coding Scheme

- 0,0,1,1,0,0,1,1,0,0,……
- Each scanned vertex is either labeled '0' or '1'
- One bit (independent of grid size) is enough



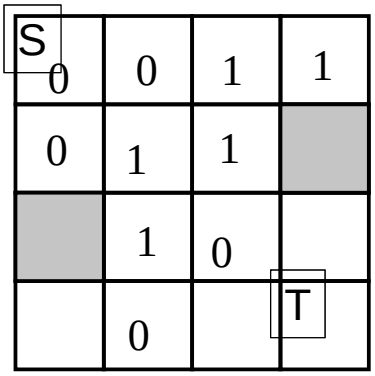
Aker's Coding Scheme

- 0,0,1,1,0,0,1,1,0,0,……
- Each scanned vertex is either labeled '0' or '1'
- One bit (independent of grid size) is enough



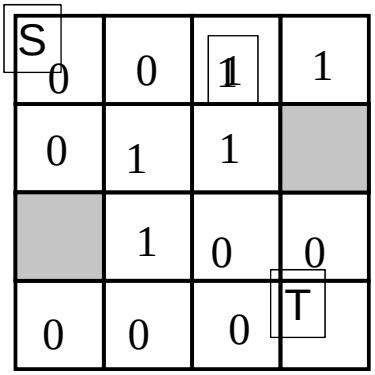
Aker's Coding Scheme

- 0,0,1,1,0,0,1,1,0,0,……
- Each scanned vertex is either labeled '0' or '1'
- One bit (independent of grid size) is enough



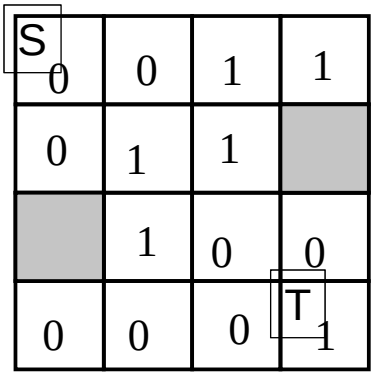
Aker's Coding Scheme

- 0,0,1,1,0,0,1,1,0,0,……
- Each scanned vertex is either labeled '0' or '1'
- One bit (independent of grid size) is enough



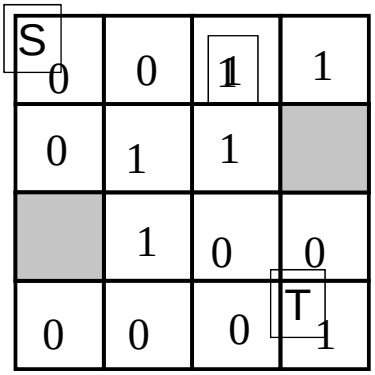
Aker's Coding Scheme

- 0,0,1,1,0,0,1,1,0,0,……
- Each scanned vertex is either labeled '0' or '1'
- One bit (independent of grid size) is enough



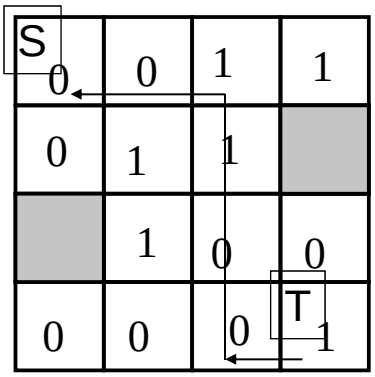
Aker's Coding Scheme

- 0,0,1,1,0,0,1,1,0,0,……
- Each scanned vertex is either labeled '0' or '1'
- One bit (independent of grid size) is enough



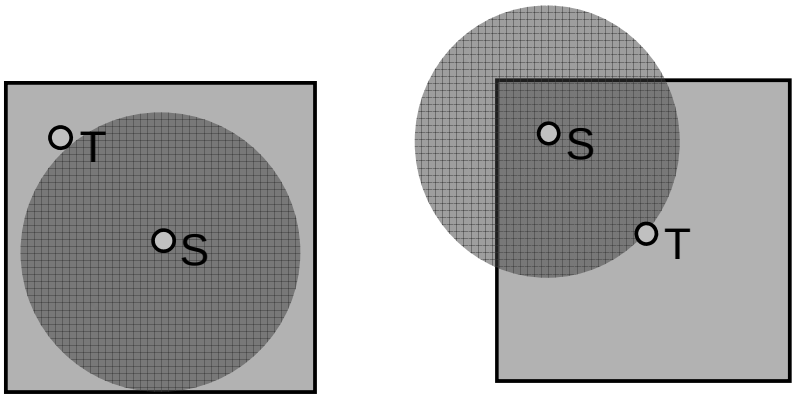
Aker's Coding Scheme

- 0,0,1,1,0,0,1,1,0,0,……
- Each scanned vertex is either labeled '0' or '1'
- One bit (independent of grid size) is enough



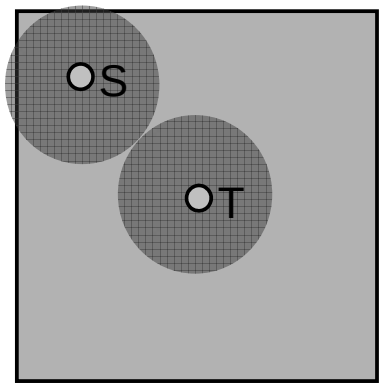
Starting Point Selection

- Spreading of the wavefront depends on the starting point
- Any one of source and target can be suitably selected as the starting point



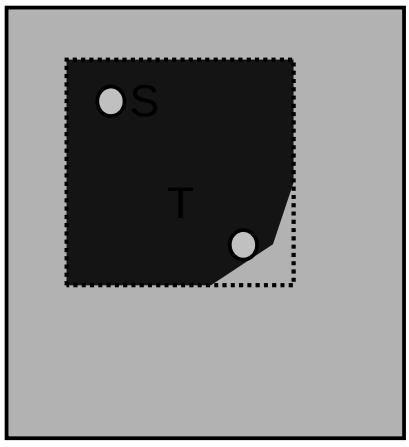
Double Fan-Out

- Wave propagates from both ends



Framing

- A subset of the entire grid (induced by the smallest rectangle covering the source and the target) is considered



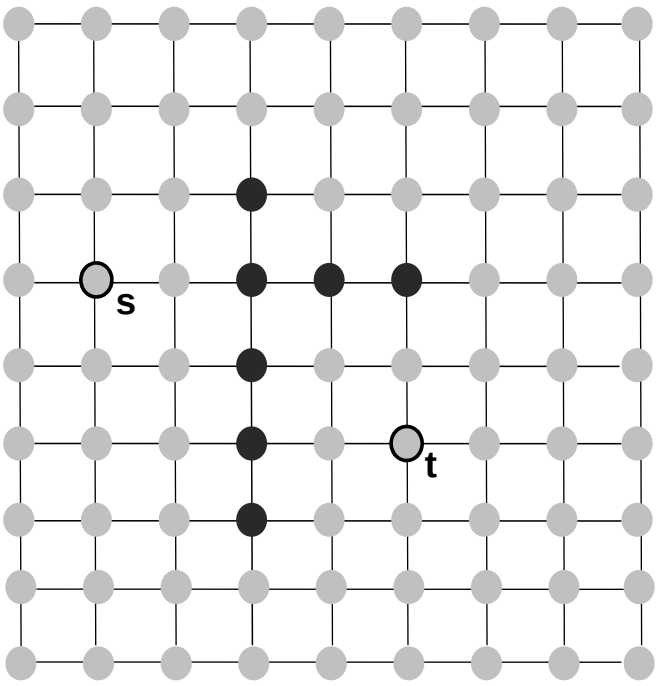
Soukup's Algorithm

Basic Idea:

- Exploration phase combines both BFS and DFS
  - Depth-first (line**)** search is first directed toward target T until an obstacle or T is reached
  - Breadth-first (Lee-type) search is used to "bubble" around an obstacle if an obstacle is reached
- Once the target is reached, the vertices are retraced from the target to the source to identify the path

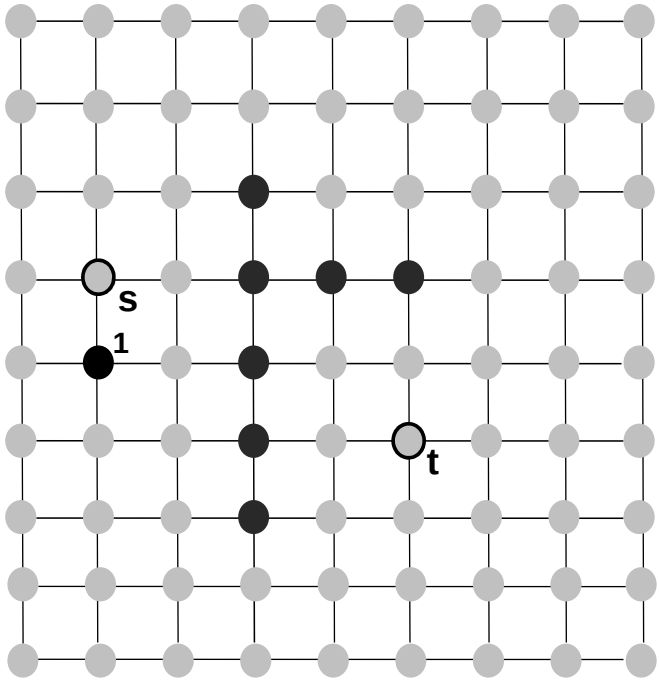
Soukup's Algorithm

Illustration:



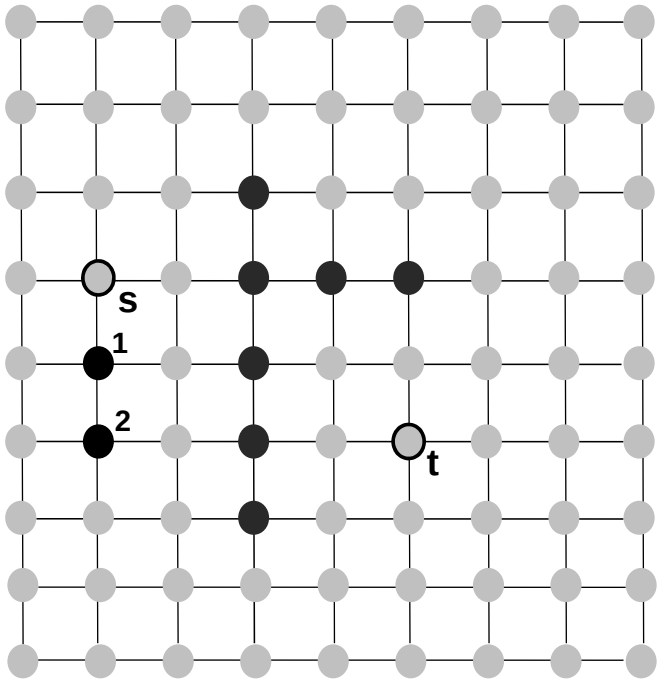
Soukup's Algorithm

Illustration:



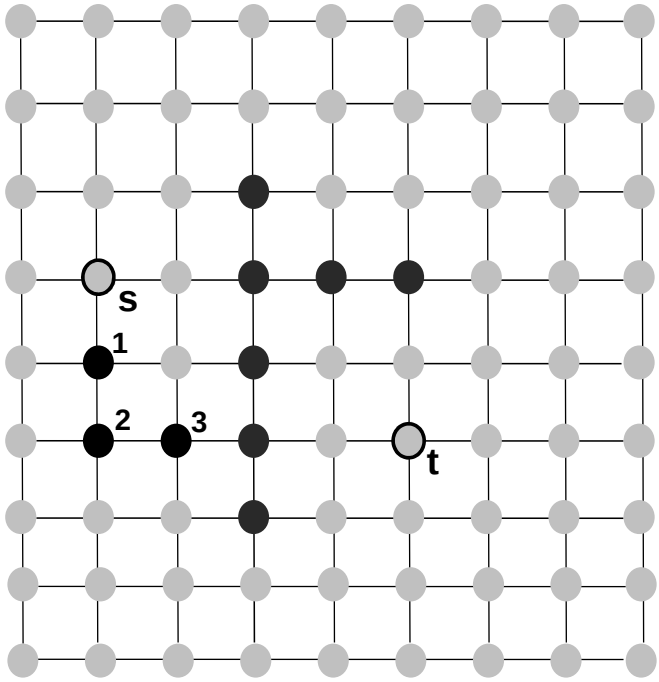
Soukup's Algorithm

Illustration:



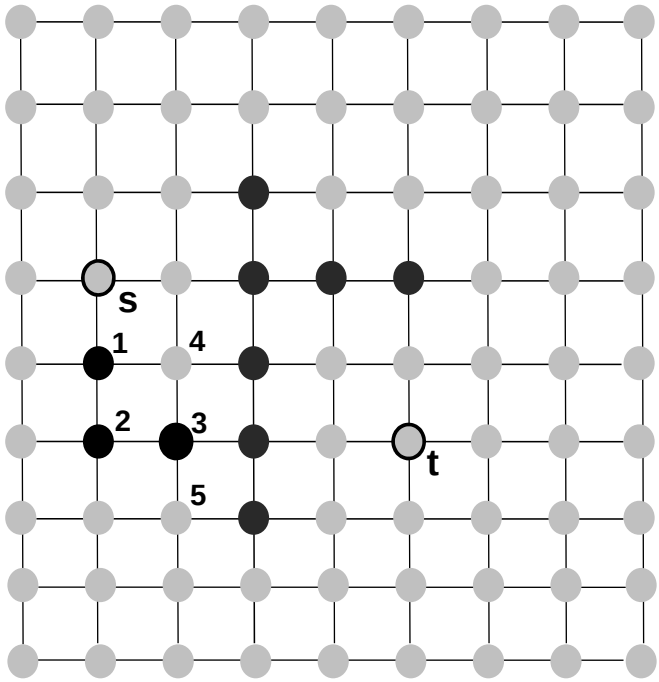
Soukup's Algorithm

Illustration:



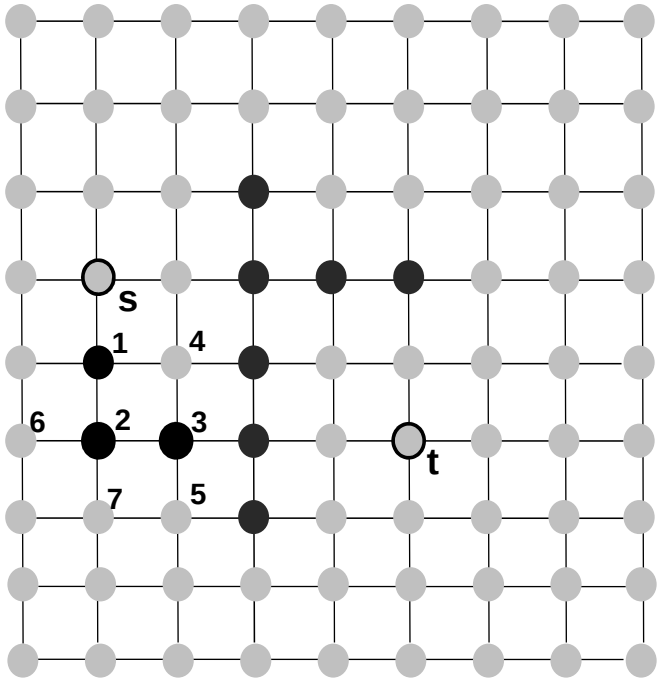
Soukup's Algorithm

Illustration:



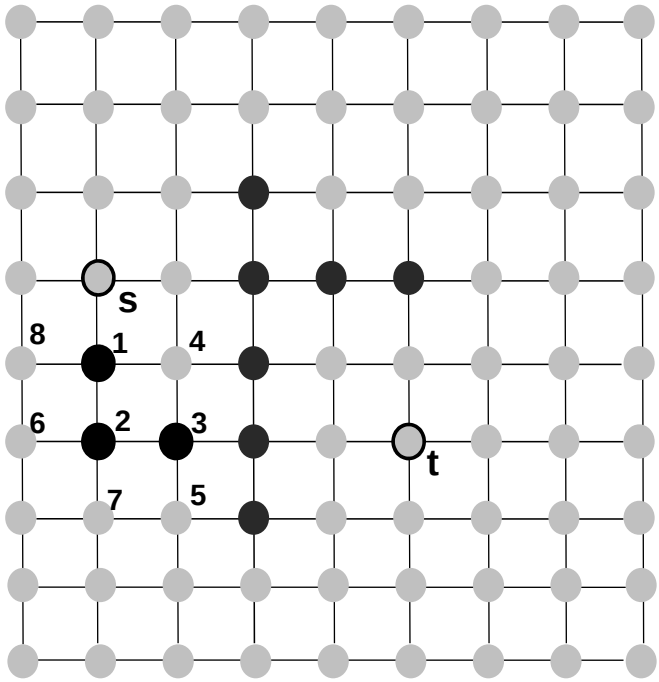
Soukup's Algorithm

Illustration:



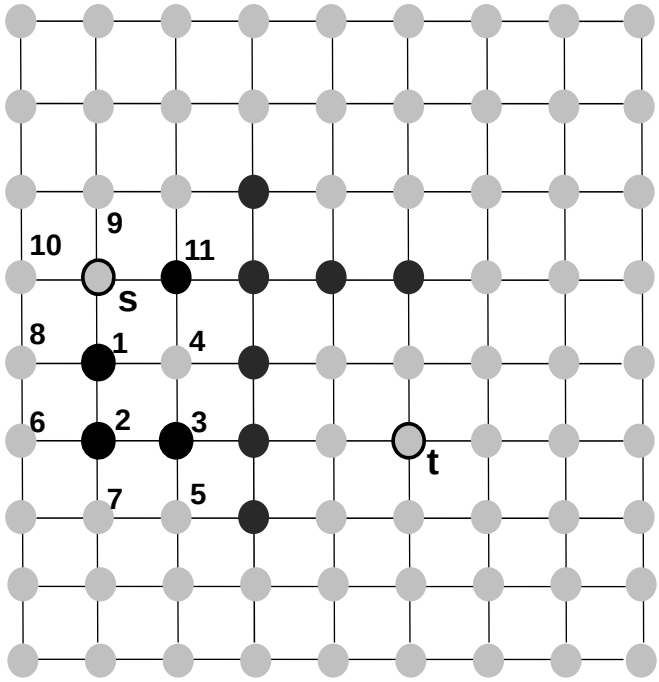
Soukup's Algorithm

Illustration:



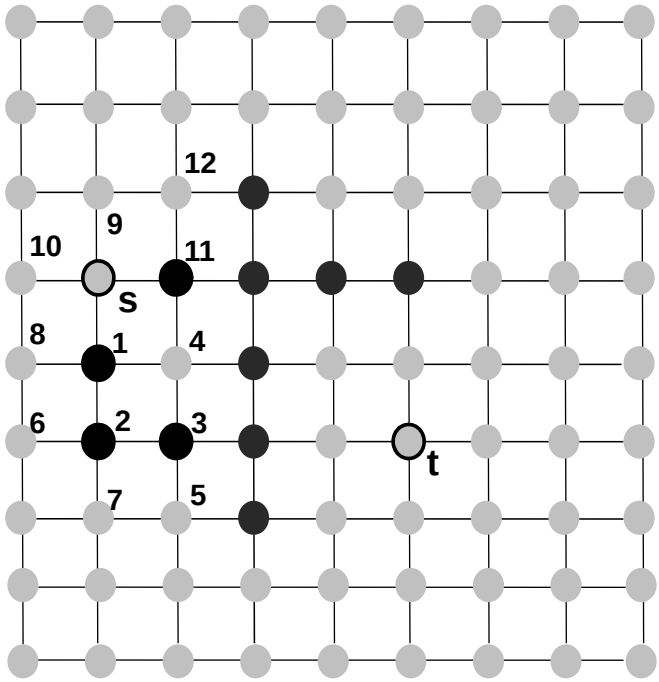
Soukup's Algorithm

Illustration:



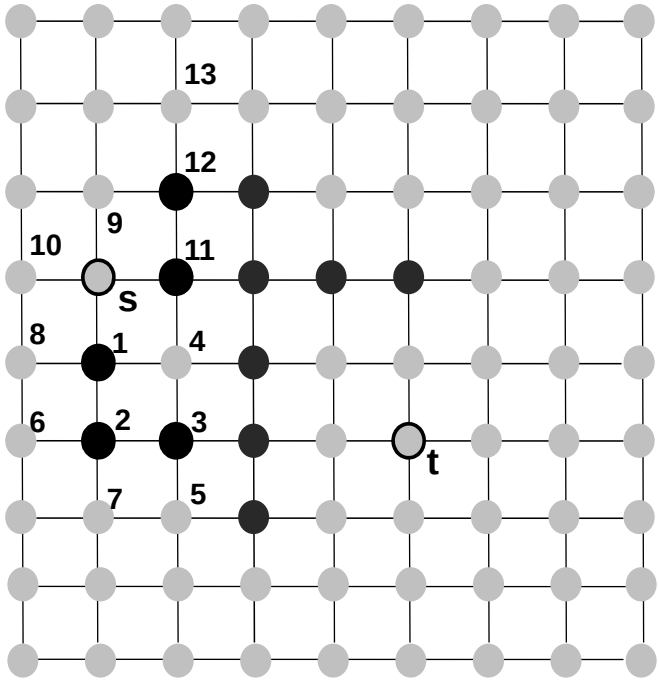
Soukup's Algorithm

Illustration:



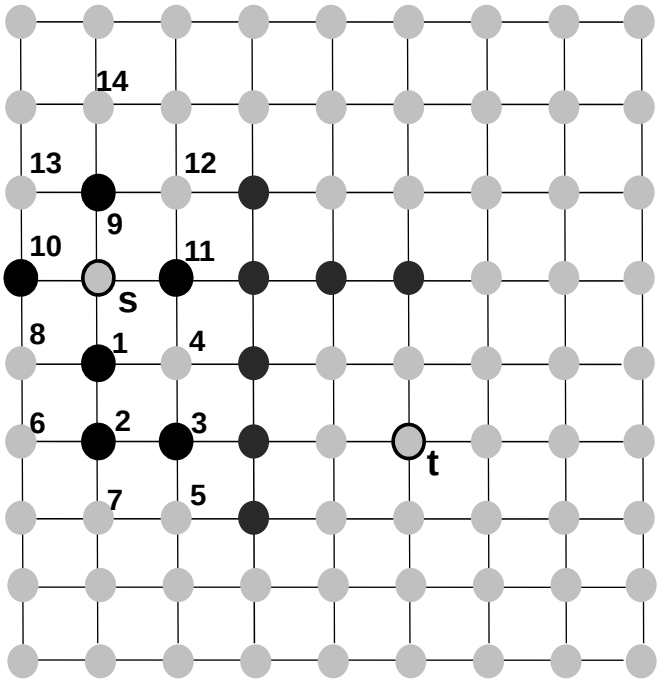
Soukup's Algorithm

Illustration:



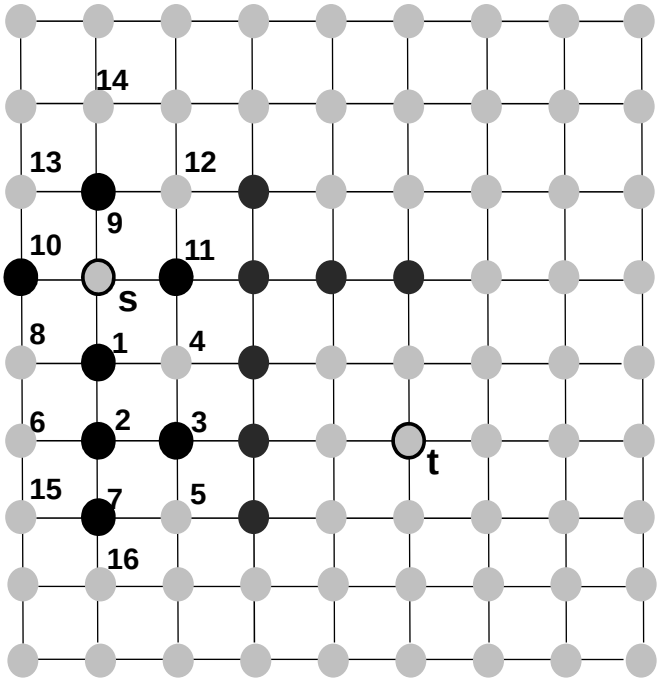
Soukup's Algorithm

Illustration:



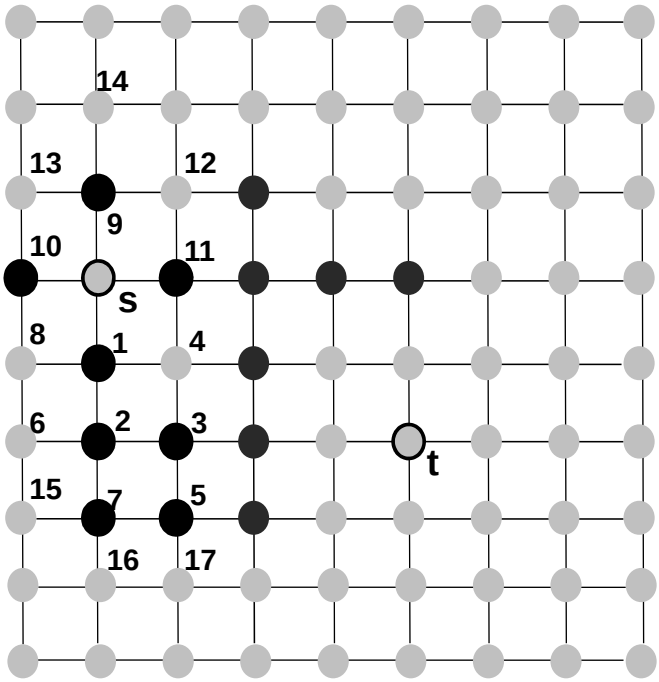
Soukup's Algorithm

Illustration:



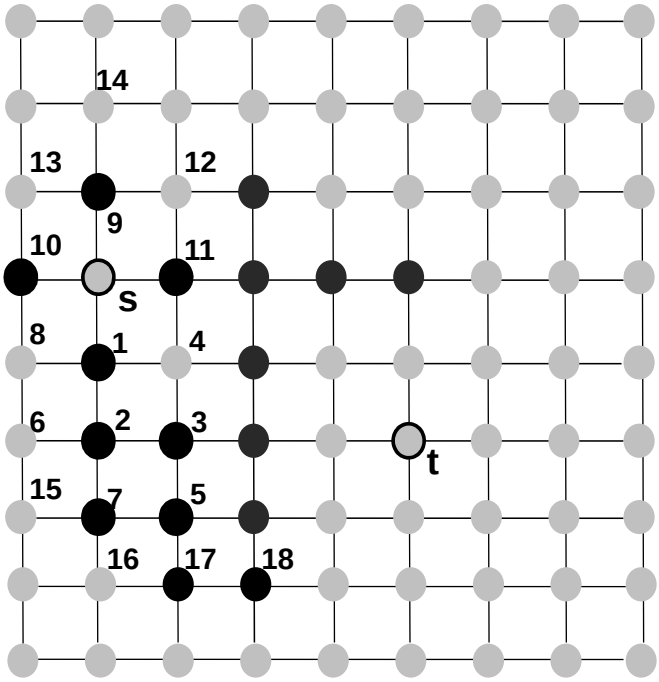
Soukup's Algorithm

Illustration:



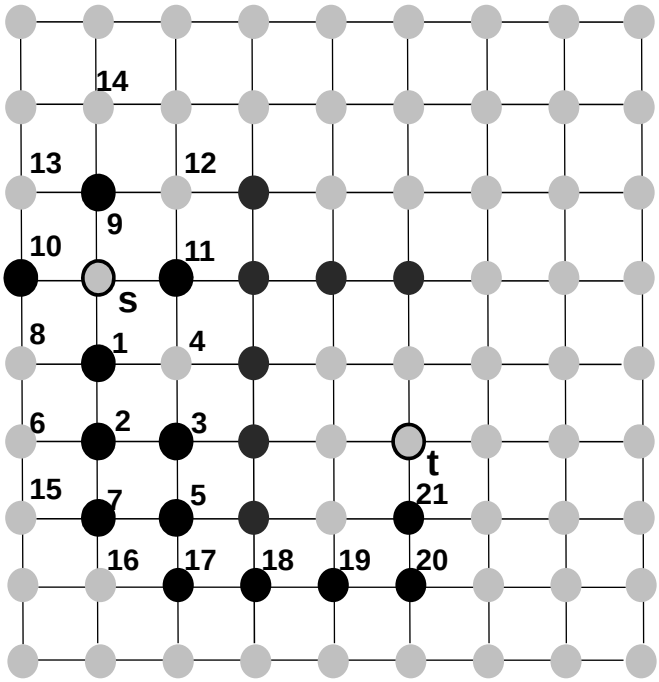
Soukup's Algorithm

Illustration:



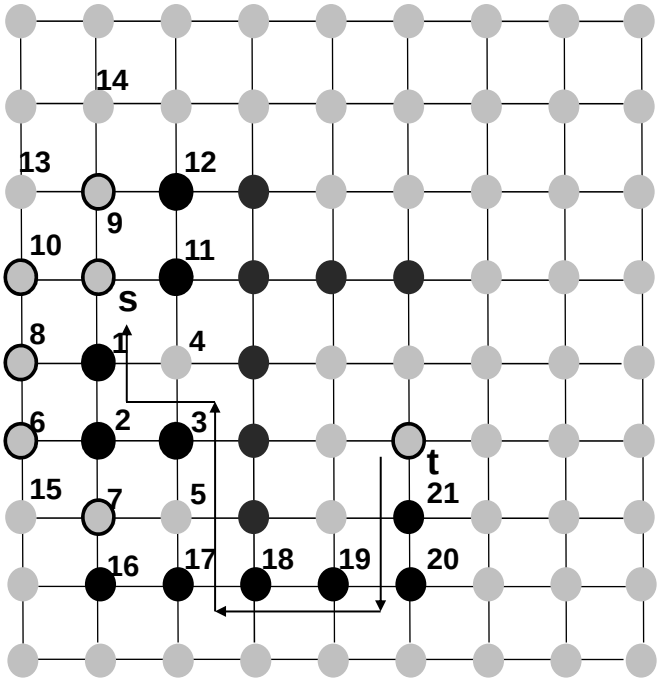
Soukup's Algorithm

Illustration:



Soukup's Algorithm

Illustration:

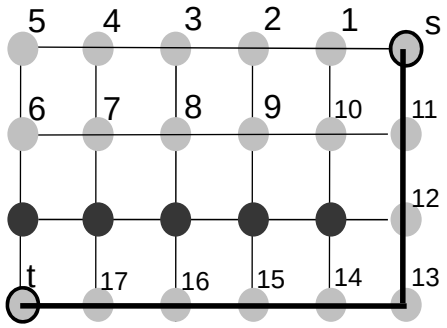


Soukup's Algorithm

Worst case:

All the vertices have to be scanned before the target is reached

Occurs when the search goes in the direction of the target, which is opposite the direction of the passageway through the obstacle



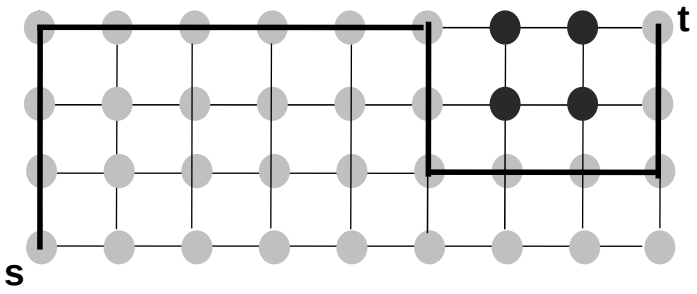
Soukup's Algorithm

Time and Space Complexities:

- O(*h***w*), but 10-50 times faster than Lee's algorithm

May get Sub-Optimal solution

- Finds a path between *S* and *T*, but may not be the shortest!



Lee's Vs Soukup's

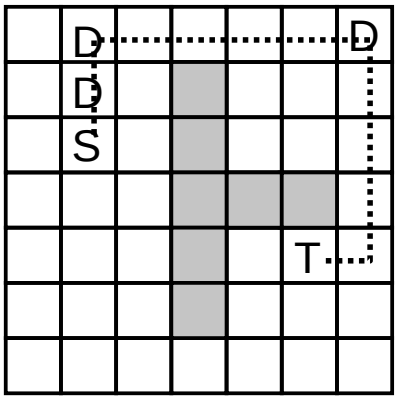
- Both are capable of finding a path if one exists
  - Lee's algorithm finds the shortest path
  - Soukup's algorithm may not find the shortest path
- Both have worst case time complexity *O(h*w)*
  - Soukup's algorithm works 10-50 times faster than Lee's algorithm

Hadlock's Algo

- To reduce runtime, use detour numbers
  - Detour number: For a path P from S to T, let detour number d(P) = # of grids directed away from T, then

  L(P) = MD(S,T) + 2d(P)

  MD : Manhattan distance
- So minimizing L(P) and d(P) are the same.

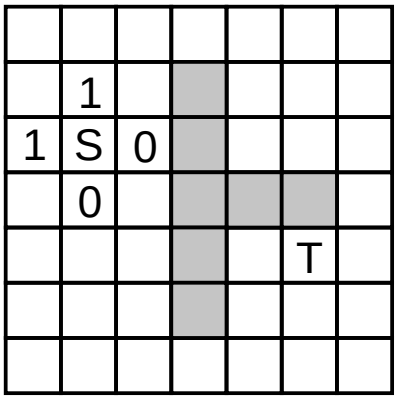


D: Detour

d(P) = 3
MD(S,T) = 6
L(P) = 6+2x3 = 12

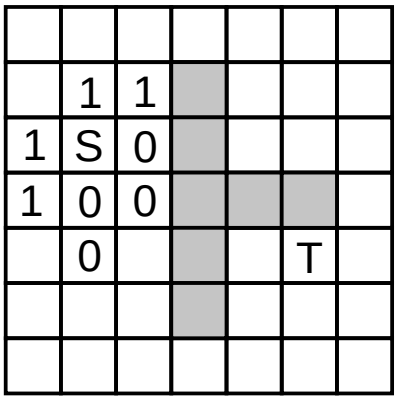
Hadlock's Algo

- Label vertices with detour numbers.
- Vertices with smaller detour number are expanded first.
- Therefore, favor paths without detour.



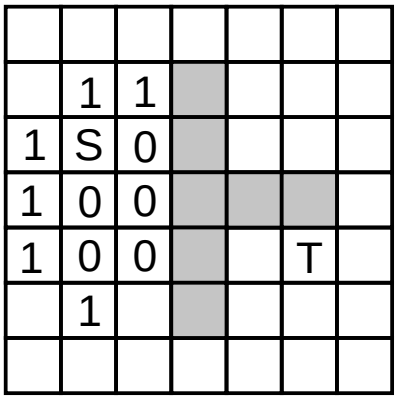
Hadlock's Algo

- Label vertices with detour numbers.
- Vertices with smaller detour number are expanded first.
- Therefore, favor paths without detour.



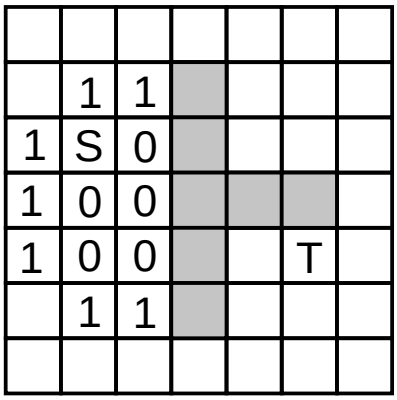
Hadlock's Algo

- Label vertices with detour numbers.
- Vertices with smaller detour number are expanded first.
- Therefore, favor paths without detour.



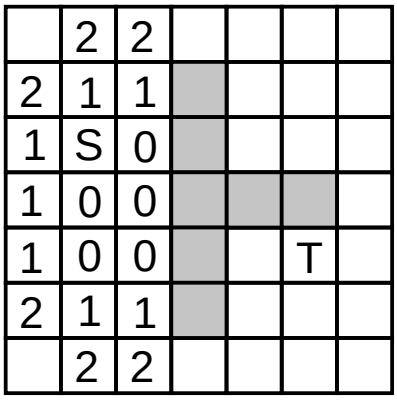
Hadlock's Algo

- Label vertices with detour numbers.
- Vertices with smaller detour number are expanded first.
- Therefore, favor paths without detour.



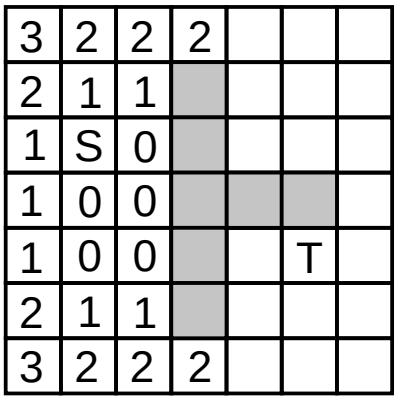
Hadlock's Algo

- Label vertices with detour numbers.
- Vertices with smaller detour number are expanded first.
- Therefore, favor paths without detour.



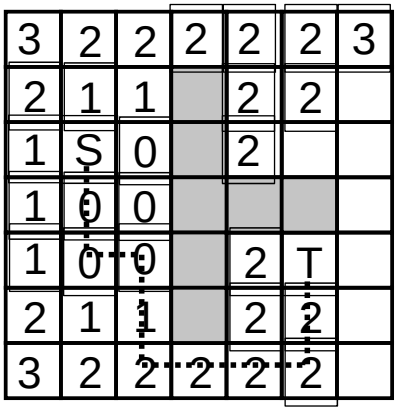
Hadlock's Algo

- Label vertices with detour numbers.
- Vertices with smaller detour number are expanded first.
- Therefore, favor paths without detour.



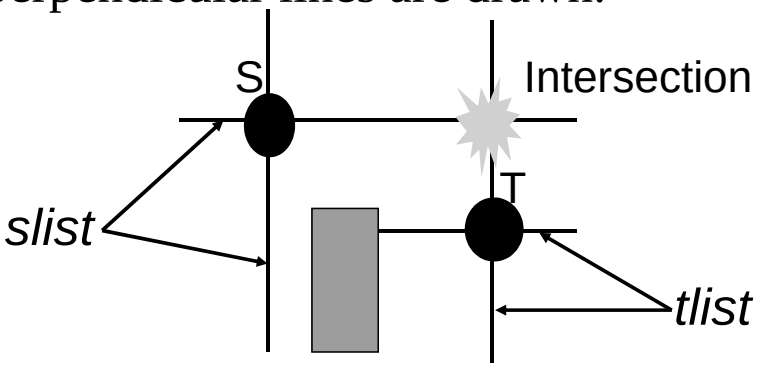
Hadlock's Algo

- Label vertices with detour numbers.
- Vertices with smaller detour number are expanded first.
- Therefore, favor paths without detour.



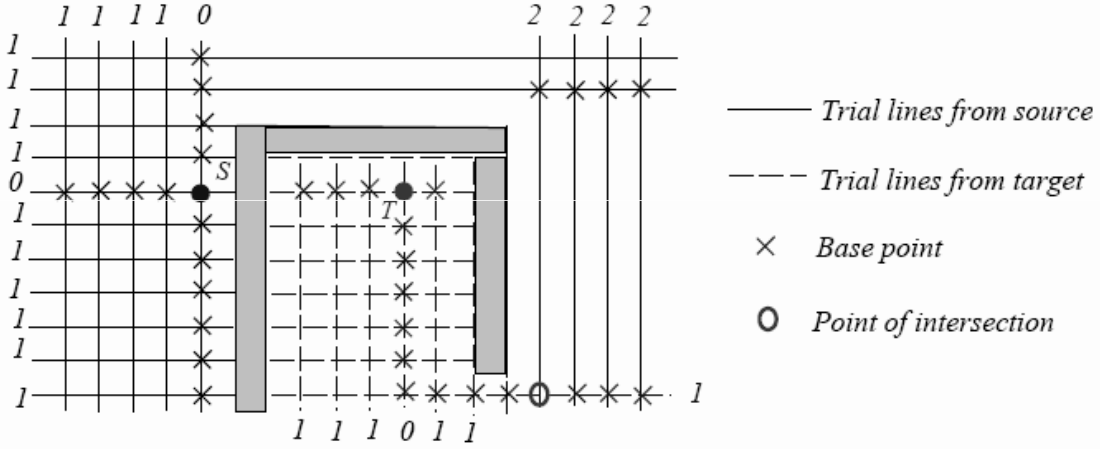
Line Probe algorithm

- Keep two lists of line segments, *slist* and *tlist*, for the source and the target respectively.
- If a line segment from *slist* intersects with one from *tlist*, a route is found;
- else, new line segments are generated from the escape points.
- Escape points: a grid point on line segment, from which perpendicular lines are drawn.



Line Probe : Mikami's algorithm

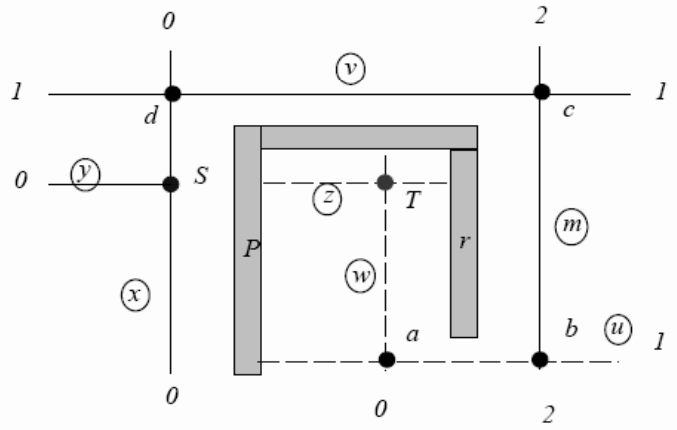
- Mikami & Tabuchi, "A computer program for optimal routing of printed circuit connectors," IFIP, H47, 1968.
- Every grid point is an escape point.



- every grid point on line : escape point
- Time Complexity: O(L), L: number of line segments produced

Line Probe: Hightowers' algorithm

- Hightower, "A solution to line-routing problem on the continuous plane," DAC-69.
- A single escape point on each line segment.
- If a line parallels to the blocked cells, the escape point is placed just past the endpoint of the segment.



Global routing Algorithm

	Maze routing			Line search	
	Lee	Soukup	Hadlock	Mikami	Hightower
Time	$O(MN)$	$O(MN)$	$O(MN)$	$O(L)$	$O(L)$
Space	$O(MN)$	$O(MN)$	$O(MN)$	$O(L)$	$O(L)$
Finds path if one exists?	yes	yes	yes	yes	no
Is the path shortest?	yes	no	yes	no	no
Works on grids or lines?	grid	grid	grid	line	line

- Soukup, Mikami, and Hightower all adopt some sort of line-search operations $\Rightarrow$ cannot guarantee shortest paths.