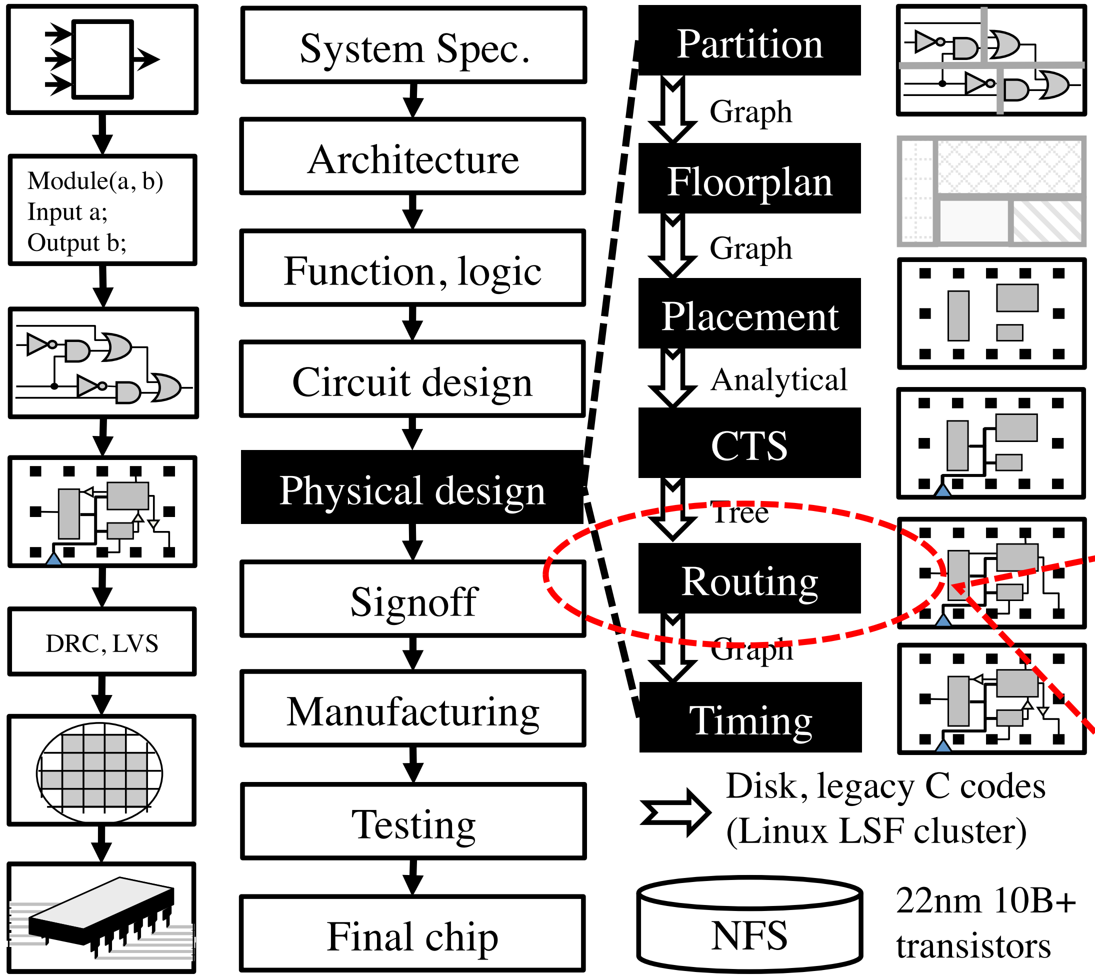
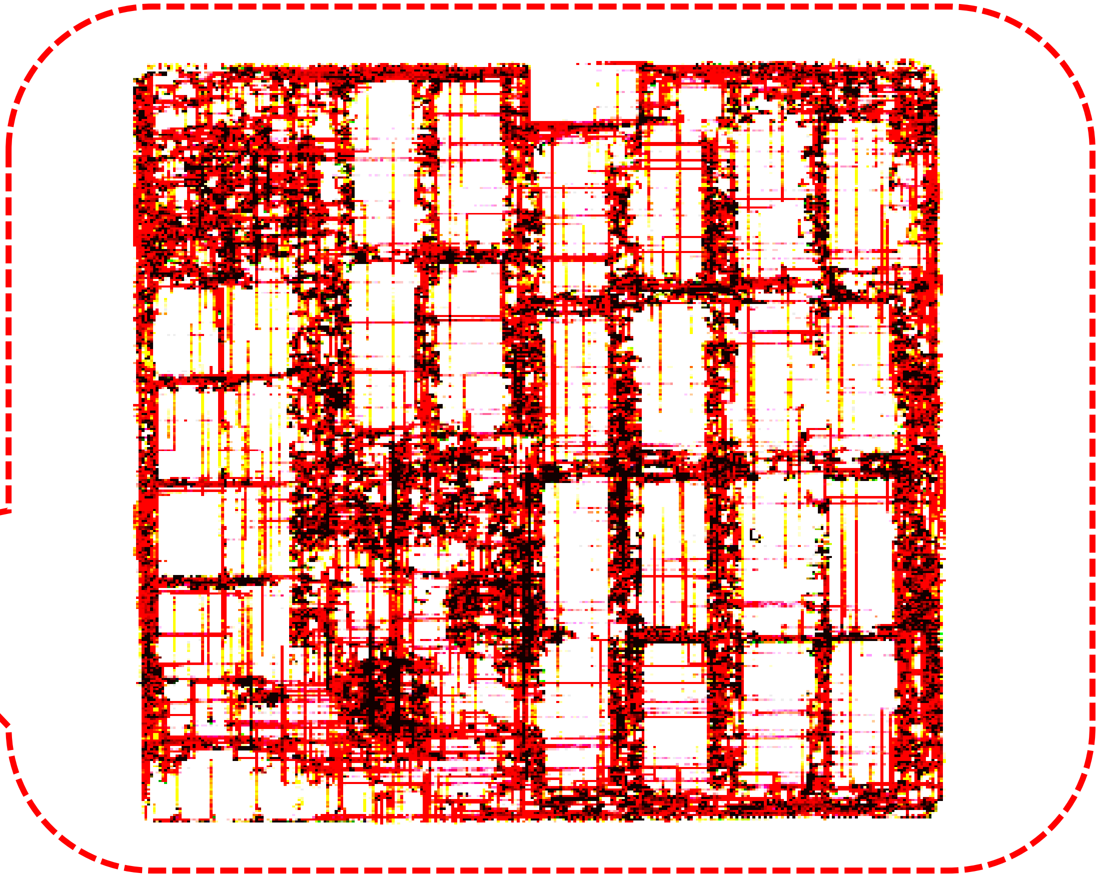
Lecture 16: Routing – I

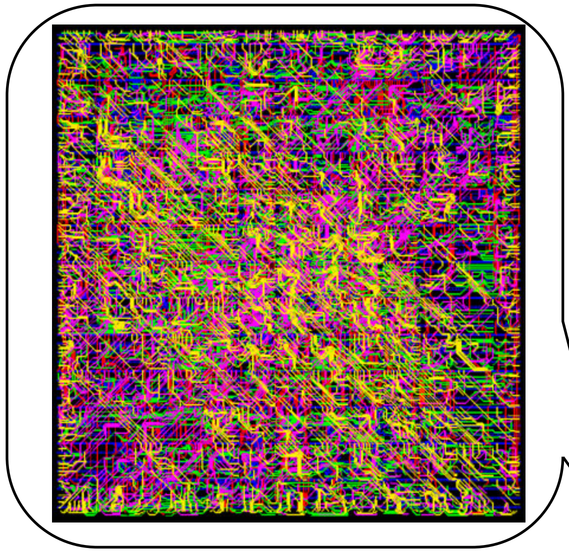
Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

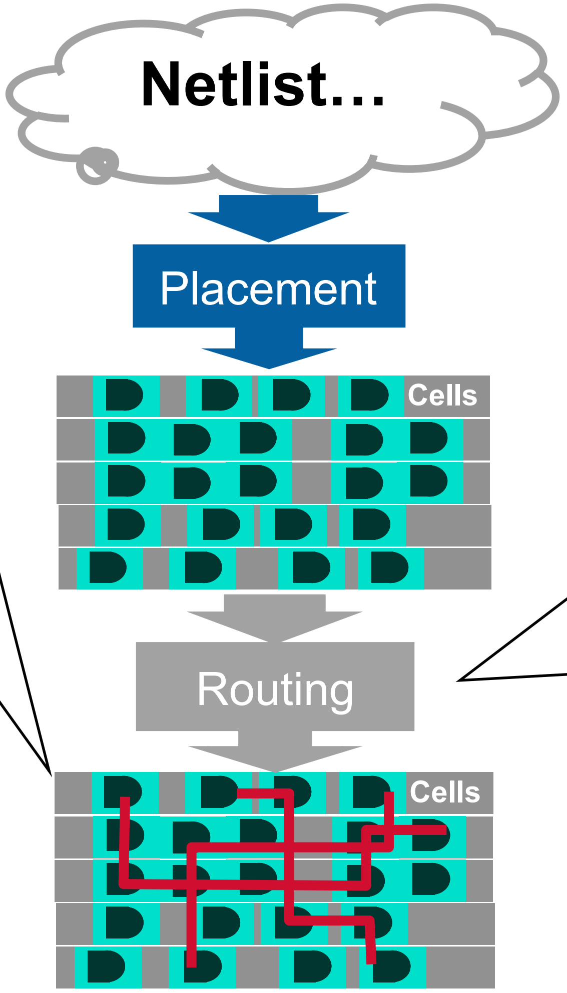
Physical Design Flow – Routing

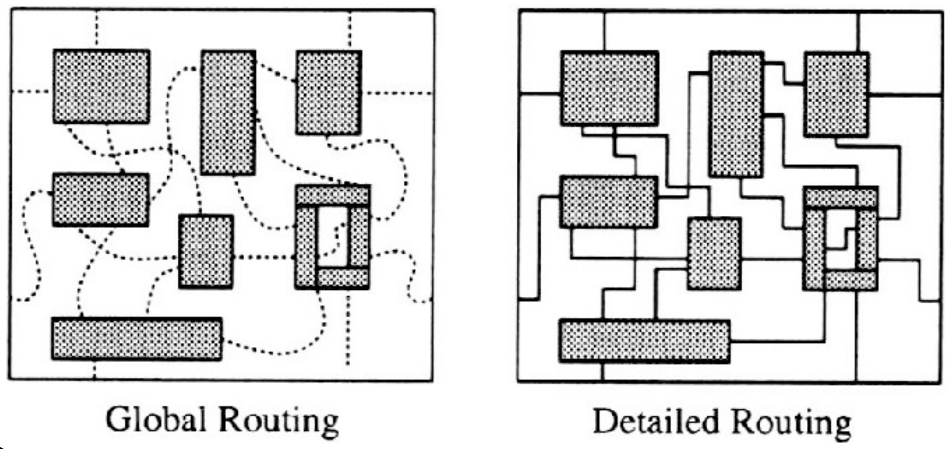




Routing Problem

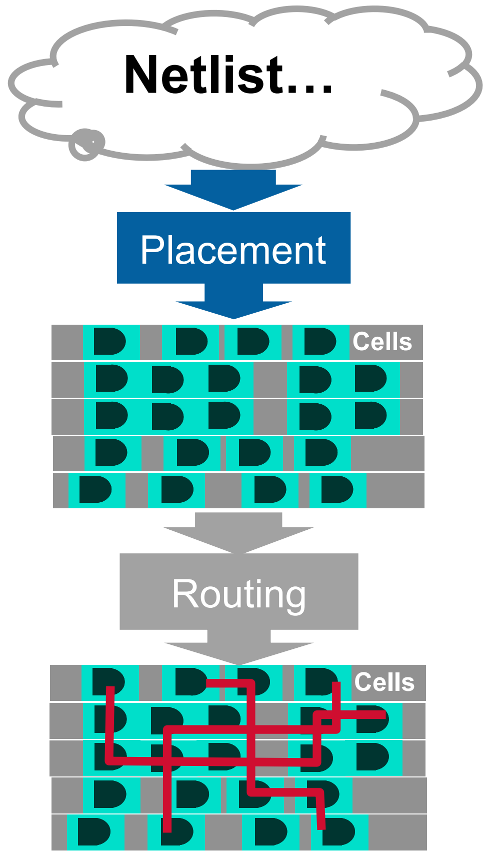






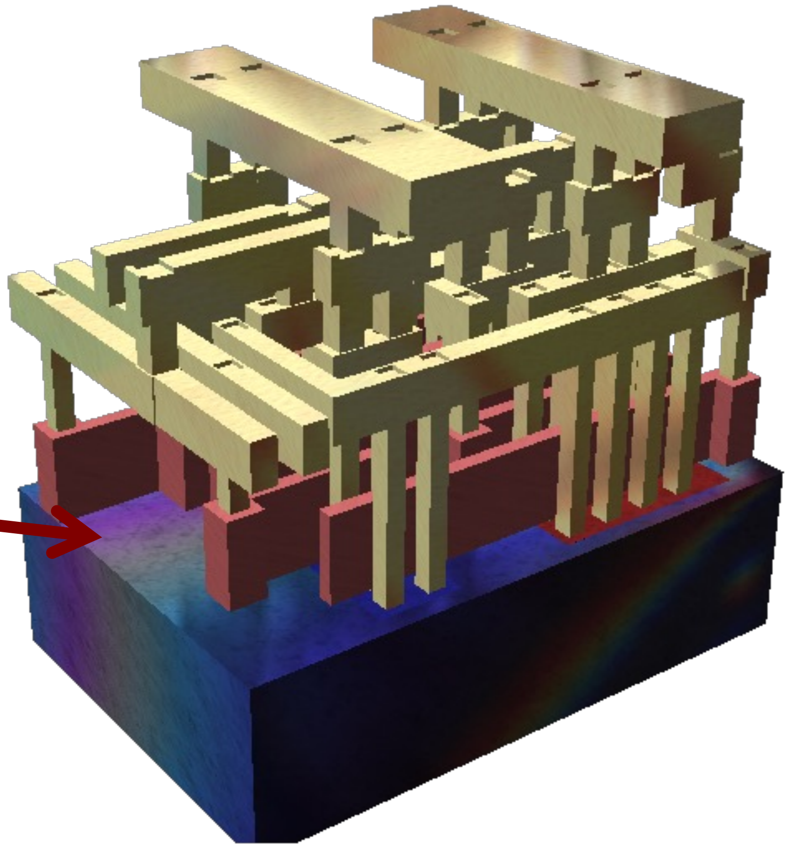
Challenges of Routing

- **Scale**
  - Big chips have an enormous number (**millions**) of wires
  - Not every wire gets to take an "easy" path to connect its pins
  - **Must** connect them all--can't afford to tweak many wires manually
- **Geometric complexity**
  - It used to be representing the layout was a simple "grid"
  - No longer true: at nanoscale, **geometry rules are complex** – makes routing hard
- **Electrical complexity**
  - It's not enough to make sure you connect all the wires
  - Must ensure **delays** thru the wires are not too big
  - And wire-to-wire **interactions** (crosstalk) don't mess up



Physical Assumptions

- **Many layers of wires**
  - Made of metal (today, copper)
  - We can connect wires in different layers with **vias**
- **A simplified view**
  - Standard cells are using metal on layers 1,2
  - Routing wires on layers 3-8
  - Upper layers (9, 10) reserved for power and clock distribution (e.g., clock tree network)



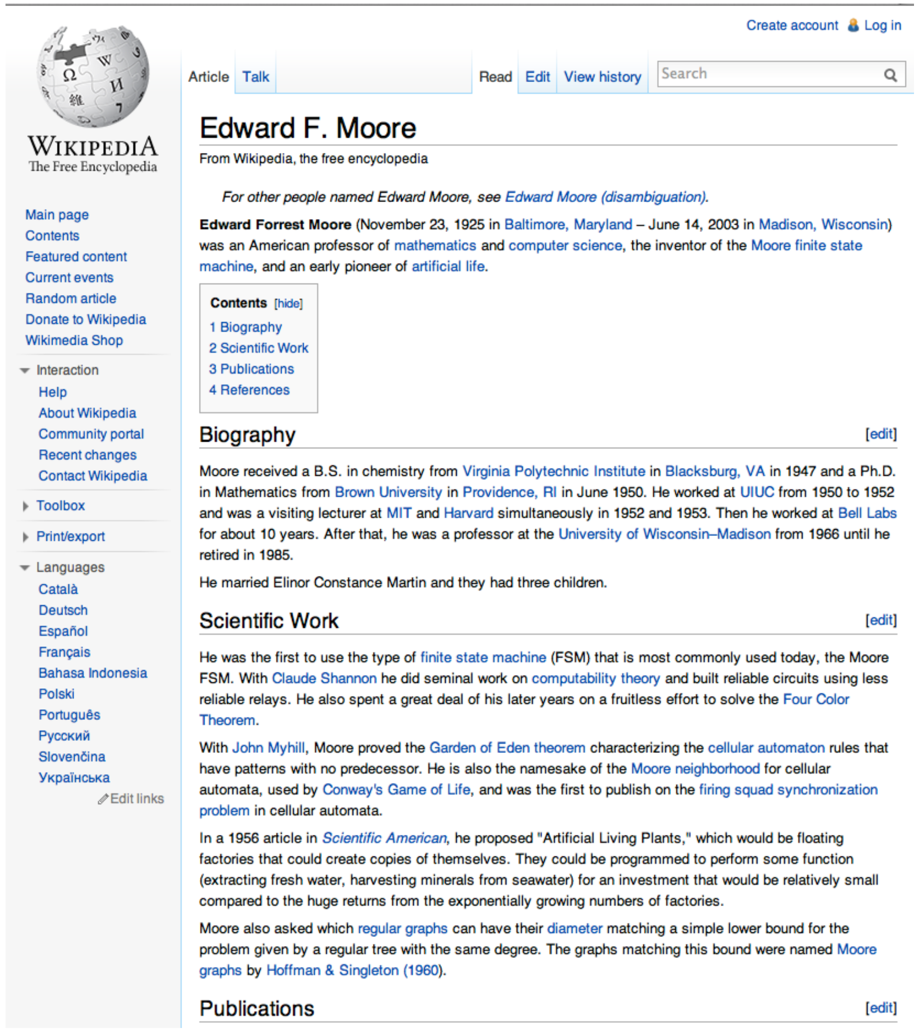
Wikipedia: search “standard cell”
http://en.wikipedia.org/wiki/File:Silicon_chip_3d.png

Placement vs Routing

- **There are lots of different placement algorithms**
  - Iterative methods. Used mainly for high-level floorplanning, not gate layout
  - Many analytical methods based on solving/optimizing large systems of equations
- **There are *not* quite so many routing algorithms**
  - There are lots of routing data structures – to represent the geometry efficiently
  - But there is one very, very big idea at core of most real routers

Big Idea: Maze Routing

- **From one famous early paper:**
  - E.F. Moore. “The shortest path through a maze,” *International Symposium on the Theory of Switching Proceedings*, pp 285--292, Cambridge, MA, Apr. 1959. Harvard University Press
  - Given a maze (or a graph), find a shortest path from entrance to exit
- **Yes – it’s that Moore of “Moore state machines” fame**



Edward F. Moore

From Wikipedia, the free encyclopedia

For other people named Edward Moore, see Edward Moore (disambiguation).

Edward Forrest Moore (November 23, 1925 in Baltimore, Maryland – June 14, 2003 in Madison, Wisconsin) was an American professor of mathematics and computer science, the inventor of the Moore finite state machine, and an early pioneer of artificial life.

Contents [hide]
1 Biography
2 Scientific Work
3 Publications
4 References

Biography [edit]

Moore received a B.S. in chemistry from Virginia Polytechnic Institute in Blacksburg, VA in 1947 and a Ph.D. in Mathematics from Brown University in Providence, RI in June 1950. He worked at UIUC from 1950 to 1952 and was a visiting lecturer at MIT and Harvard simultaneously in 1952 and 1953. Then he worked at Bell Labs for about 10 years. After that, he was a professor at the University of Wisconsin–Madison from 1966 until he retired in 1985.

He married Elinor Constance Martin and they had three children.

Scientific Work [edit]

He was the first to use the type of finite state machine (FSM) that is most commonly used today, the Moore FSM. With Claude Shannon he did seminal work on computability theory and built reliable circuits using less reliable relays. He also spent a great deal of his later years on a fruitless effort to solve the Four Color Theorem.

With John Myhill, Moore proved the Garden of Eden theorem characterizing the cellular automaton rules that have patterns with no predecessor. He is also the namesake of the Moore neighborhood for cellular automata, used by Conway's Game of Life, and was the first to publish on the firing squad synchronization problem in cellular automata.

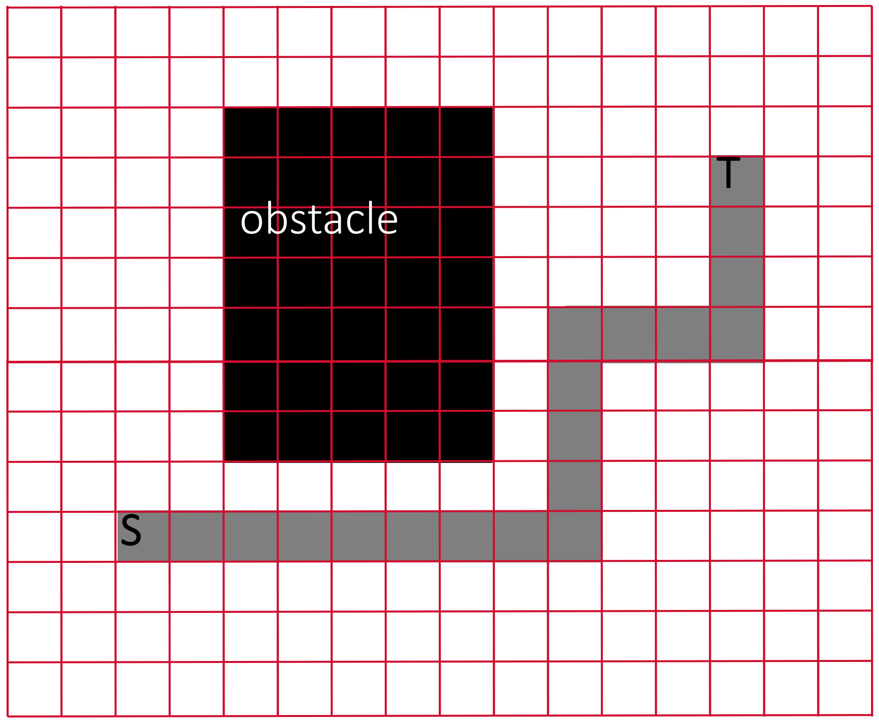
In a 1956 article in *Scientific American*, he proposed "Artificial Living Plants," which would be floating factories that could create copies of themselves. They could be programmed to perform some function (extracting fresh water, harvesting minerals from seawater) for an investment that would be relatively small compared to the huge returns from the exponentially growing numbers of factories.

Moore also asked which regular graphs can have their diameter matching a simple lower bound for the problem given by a regular tree with the same degree. The graphs matching this bound were named Moore graphs by Hoffman & Singleton (1960).

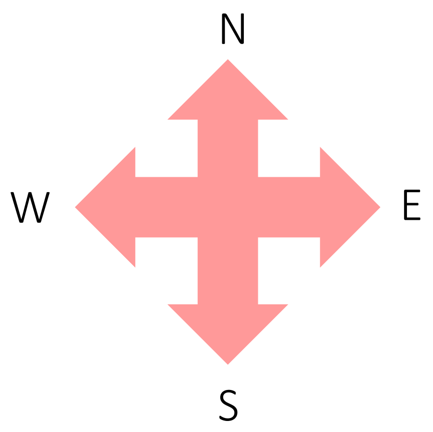
Publications [edit]

How Do We Get from “Mazes” to “Wires”?

- **Make a big geometric assumption: Gridded routing**
  - The layout surface is a grid of regular squares
  - A legal wire path = a set of connected *unobstructed* grid cells

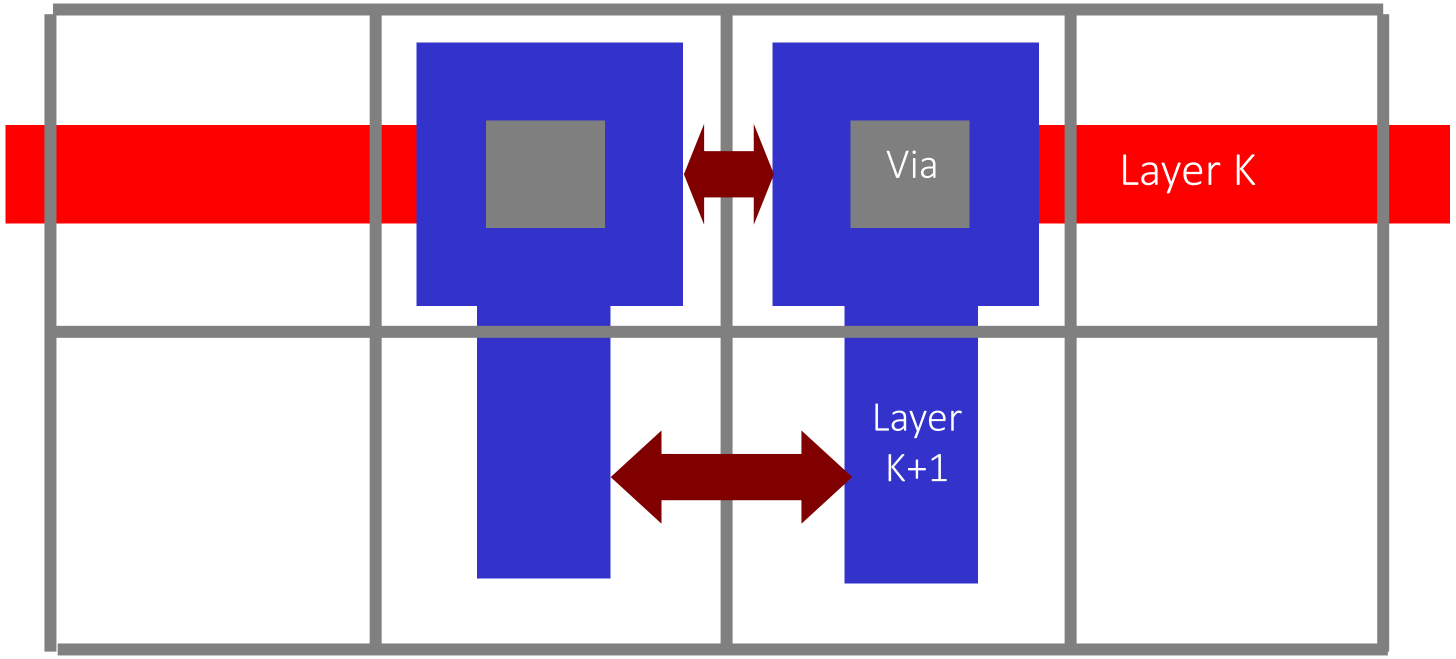


For us: wires are also **strictly horizontal and vertical.** No diagonal (eg, 45^o) angles. A path goes east/west, or north/south, in this grid.



Grid Assumptions

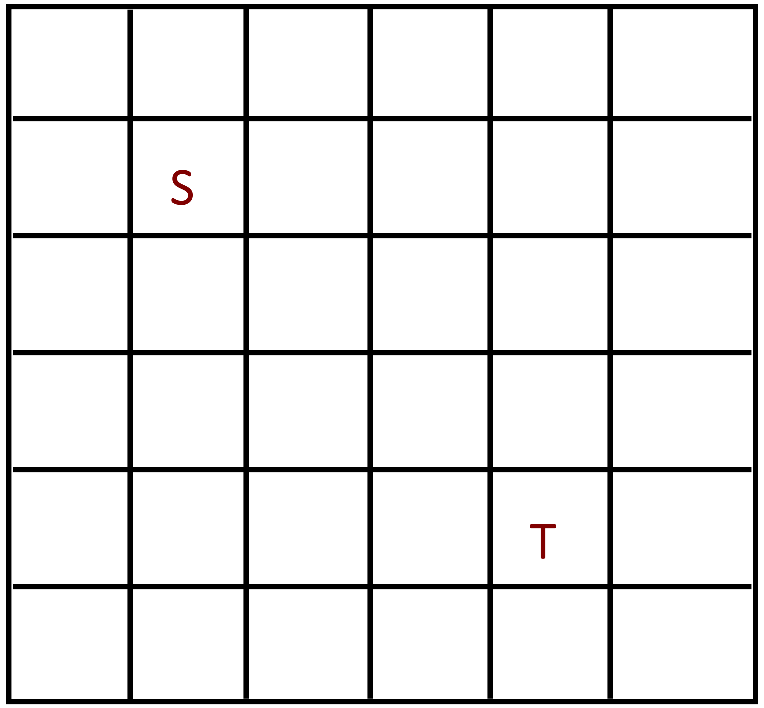
- **This is a critical assumption implying constraints on wires**
  - All wires are the same size (width)
  - All pins we want to connect are also "on grid" ie, center of grid cell
  - Wires and their vias fit in the grid, without any geometry rule (eg, spacing) violations



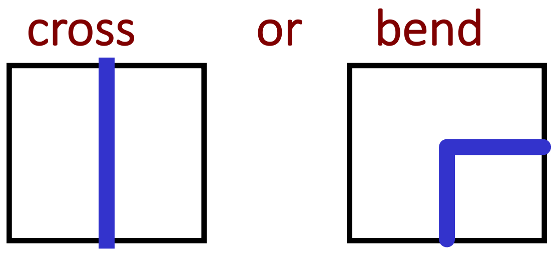
Maze Router: Strategy

- **Strategy**
  - One net at a time: completely wire **one** net, then move onto next net
  - Optimize net path: find the **best** wiring path
- **Problems**
  - Early nets wired may block path of later nets
  - Optimal choice for one net may block later nets
  - We are just going to ignore this one for the moment …

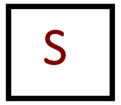
Maze Router: Basic Idea for 2-pin Net



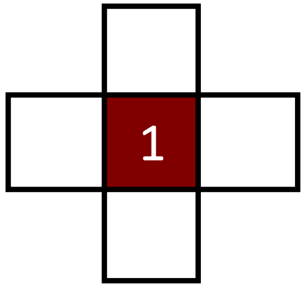
- **Given:**
  - **Grid** - each square cell represents where **one** wire can cross
  - A ***source (S)*** and ***target (T)***
- **Objective:**
  - Find shortest path connecting **source** cell (S) and **target** cell (T)
  - When using cells, a wire can:



Maze Router: Expansion

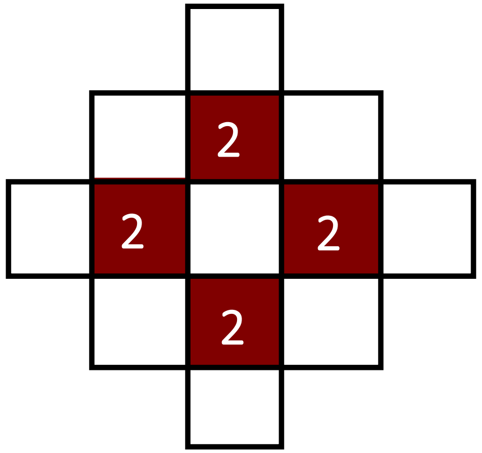


Start at the **source**



Find all new cells that are reachable at **pathlength 1**, ie, all paths that are just 1 unit in total length (just 1 cell) - mark all with this the pathlength

Repeat the expansion until the target is found!



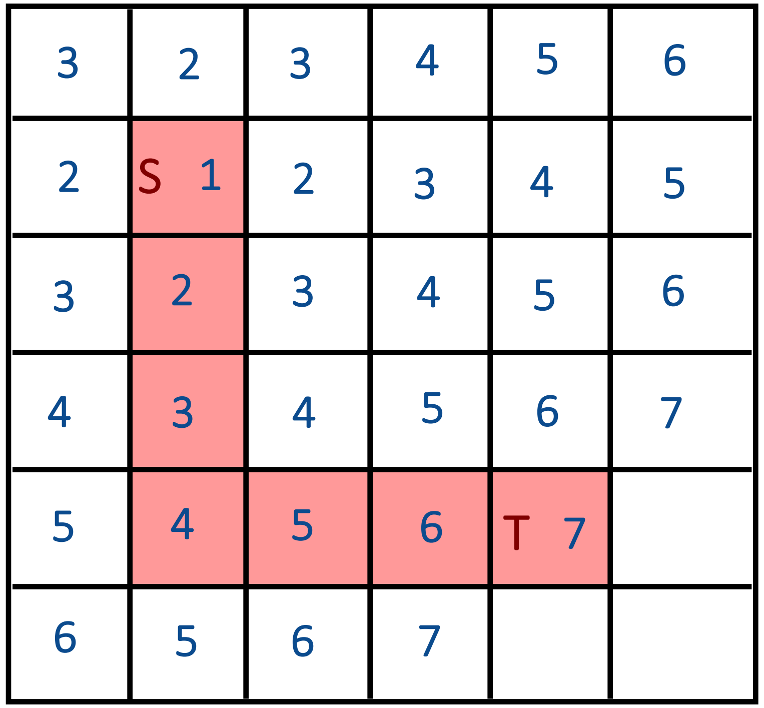
Using the **pathlength 1** cells, find all new cells which are reachable at **pathlength 2**

Maze Router Walkthrough – I

3	2	3	4	5	6
2	S 1	2	3	4	5
3	2	3	4	5	6
4	3	4	5	6	7
5	4	5	6	T 7	
6	5	6	7		

- **Strategy**
  - Expand **one cell at a time** until all the shortest paths from **S** to **T** are found.
  - Expansion creates a **wavefront** of paths that search broadly out from source cell until target is hit
  - Remember this!? We have done this using breadth-first-search (BFS) algorithm!

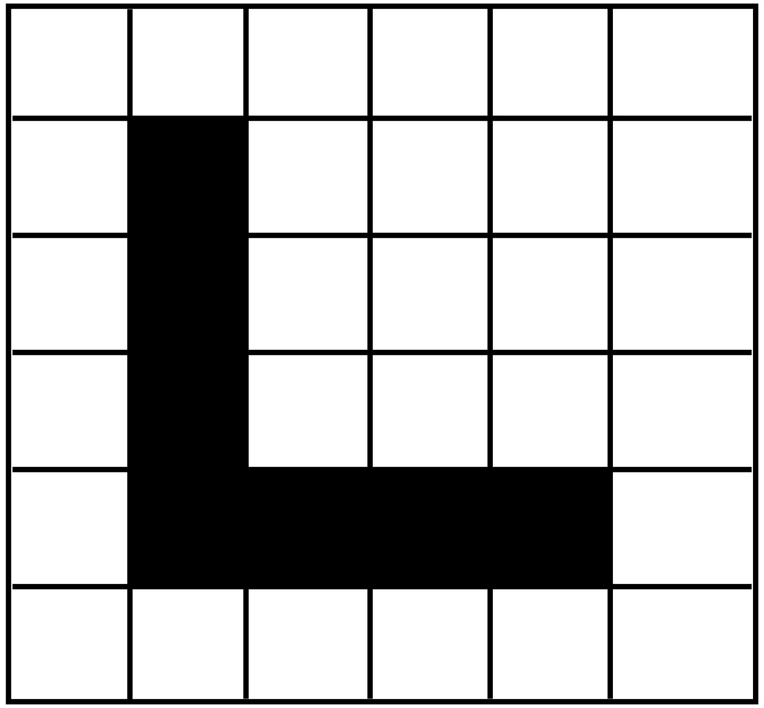
Maze Router Walkthrough – II



3	2	3	4	5	6
2	S 1	2	3	4	5
3	2	3	4	5	6
4	3	4	5	6	7
5	4	5	6	T 7	
6	5	6	7		

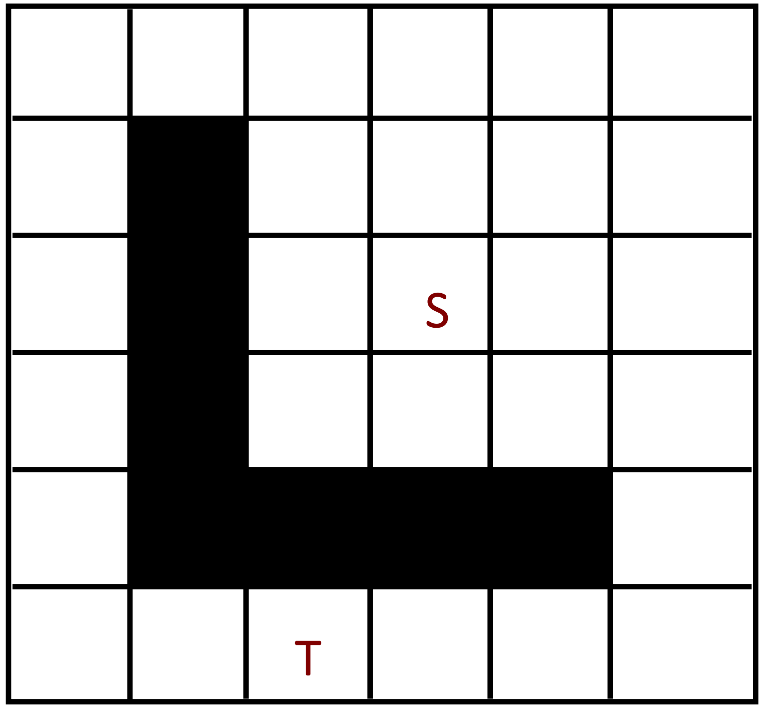
- **Now what? Backtrace**
  - Select a shortest-path (**any** shortest-path) from target back to source
  - Mark its cells so they cannot be used again – mark them as **obstacles** for later wires we want to route
  - Since there are many paths back, optimization information can be used to select the best one
  - Here, just follow the pathlengths in the cells **in descending order**

Maze Router Walkthrough – III



- **Now what? Clean-up**
  - Clean up the grid for the next net, leaving the **S** to **T** path as an **obstacle**
  - Now, ready to route the next net with the obstacles from the previously routed net in place in the grid

Maze Router Walkthrough – IV



- **Also called “Blockages”**
  - Any cell you cannot use for a wire is a an **obstacle** or a **blockage**
  - There may be parts of the routing surface you just cannot use
  - But most importantly, you **label each newly routed net as a blockage**
  - Thus, all future nets must **route around** this blockage

Classical Maze Router

- **Expand**
    - Breadth-first-search to find all paths from source to target
- **Backtrace**
    - Walk shortest path back to the source and mark path cells as used
- **Clean-Up**
    - Erase all distance marks from other grid cells before next net is route.

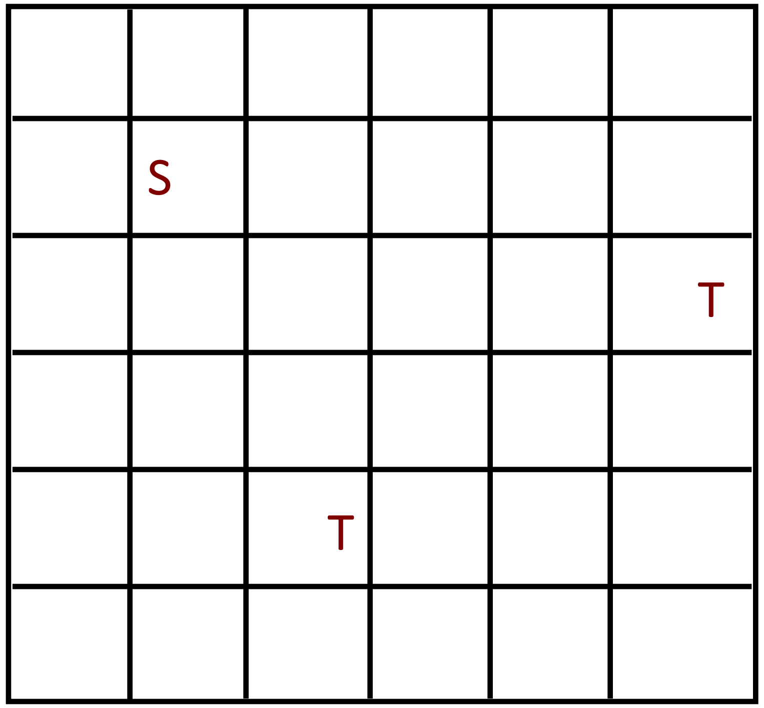
Problems of Maze Router

- **Storage**
  - Do we need a really big grid to represent a big routing problem?
  - What information is required in each cell of this grid?
- **Complexity**
  - Do we really have to search the whole grid each time we add a wire?
- **Technology**
  - Just 1 wiring layer? How do we do 2 layers? 3? 4? 6? 8? 10?
  - How do we deal with vias (connecting different routing layers?)
- **2 issues here**
  - **Applications** of basic algorithm versus **implementation** issues

Multi-point Net

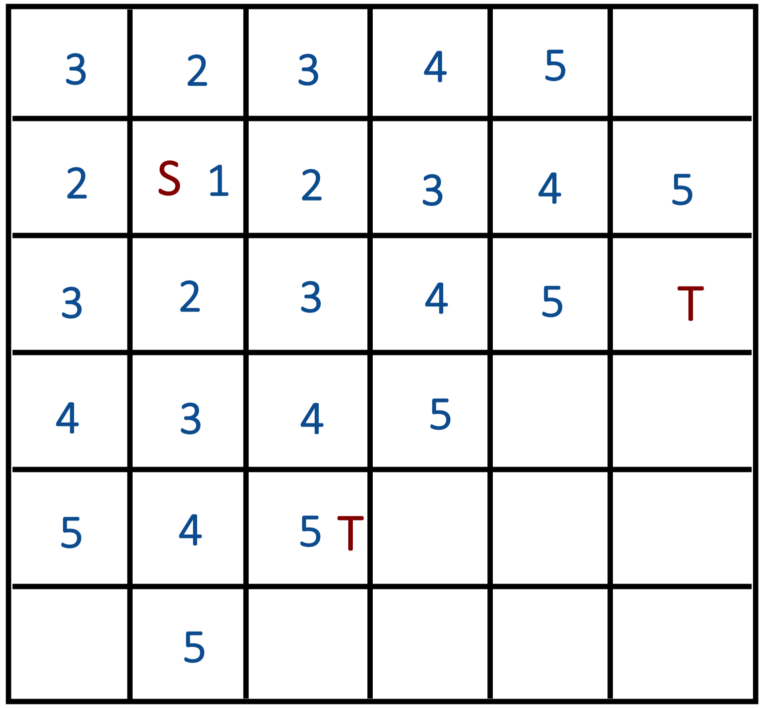
- **Multi-point Nets**
  - **One** source → **Many** targets
  - You get this with any net that represents **fanout** (almost all real nets)
- **Simple strategy**
  - **Start**: Pick **one** point as source, label **all the others** as targets
  - **First**: Use maze router to find path from source to **nearest** target
    - **Note**: You don't know which one this is up front, routing will find it
  - **Next**: Re-label all cells on found path as sources, then rerun maze router using all sources simultaneously
  - **Repeat**: For each remaining unconnected target point

Multi-point Net Walkthrough – I



- **Given:**
  - A source and **many targets**
- **Problem:**
  - Find a **short** path connecting source and targets

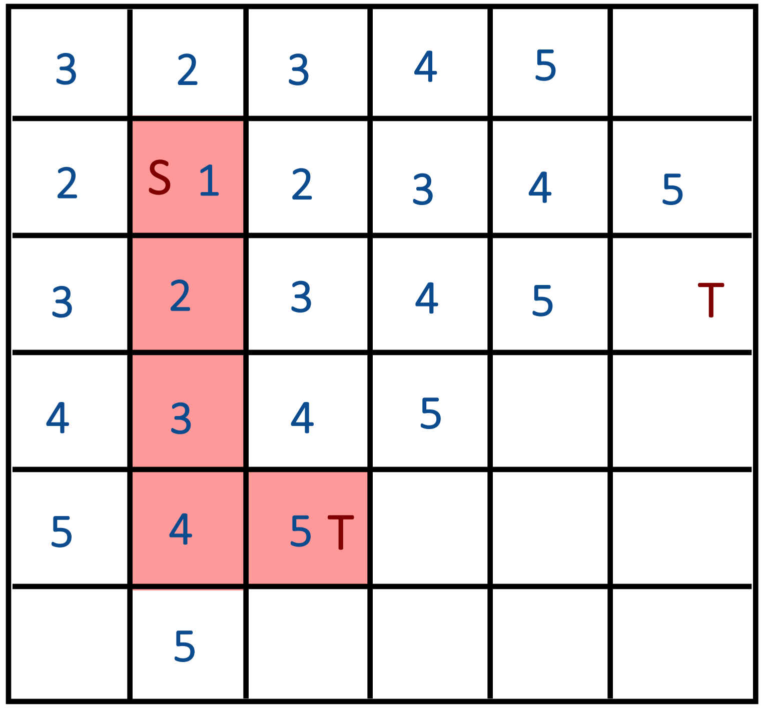
Multi-point Net Walkthrough – II



3	2	3	4	5	
2	S 1	2	3	4	5
3	2	3	4	5	T
4	3	4	5		
5	4	5 T			
	5				

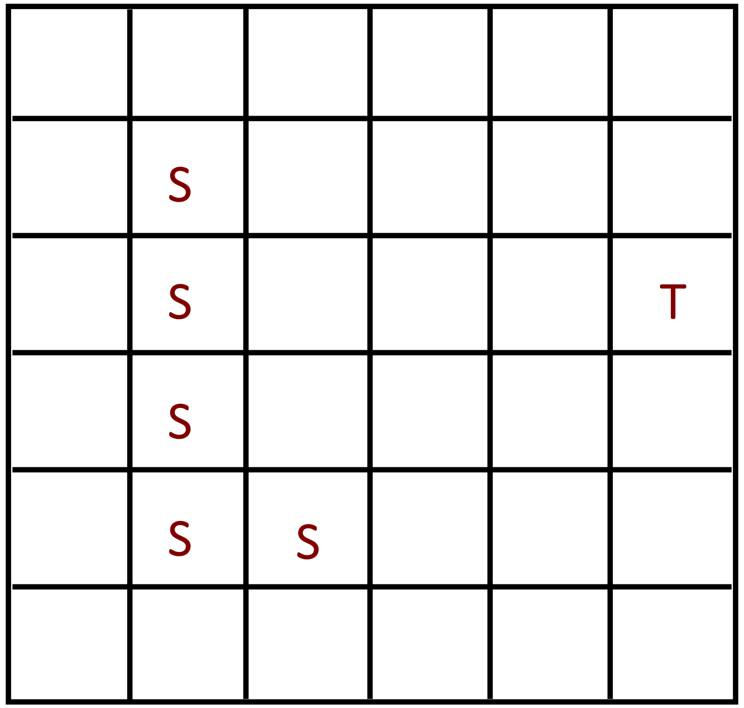
- **First segment of path...**
  - Run maze route to find the closest target
  - Start at source, go till we find **any target**

Multi-point Net Walkthrough – III



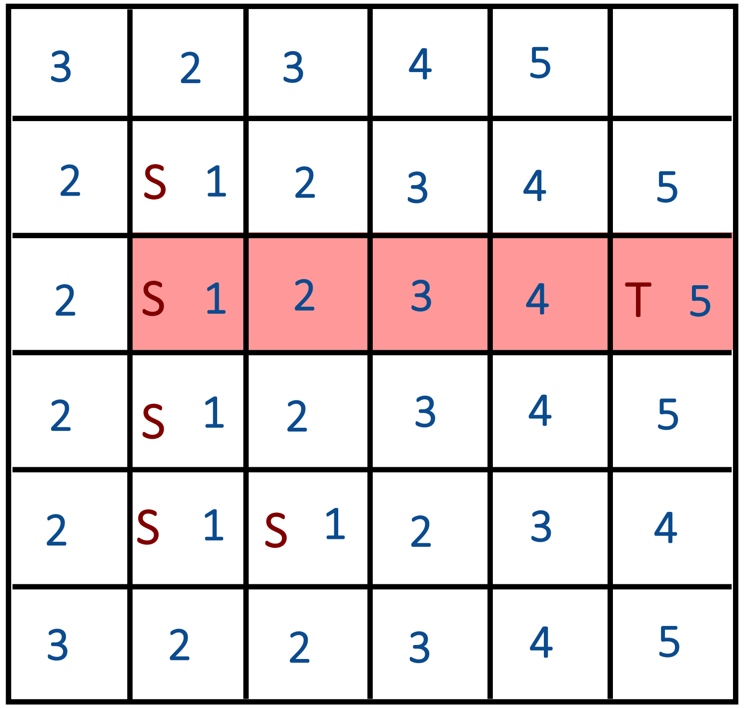
- **Second …**
  - **Backtrace and re-label the whole route as Sources for the next pass**
- **Note – this is different**
  - We don't relabel the path as blockage (yet), as we did before
  - We label it as source, so we can find paths from any point on this segment, to the rest of the targets

Multi-point Net Walkthrough – IV



- **Second…**
  - We will expand this entire set of source cells to find next segment of the net
  - Idea is we will look for paths of length 1 away from this whole set of sources, then length 2, 3, etc.
  - Go till hit another target

Multi-point Net Walkthrough – V

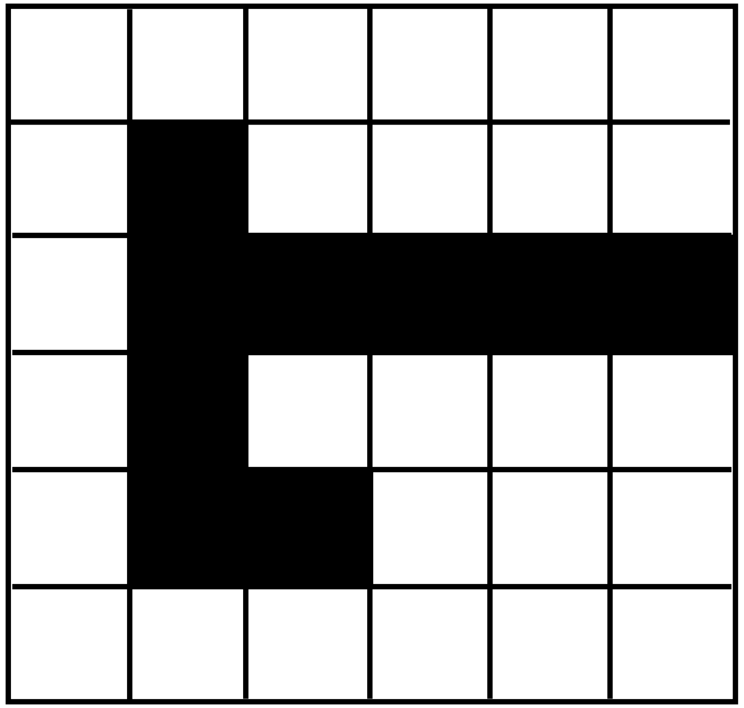


3	2	3	4	5	
2	S 1	2	3	4	5
2	🟥 S 1	🟥 2	🟥 3	🟥 4	🟥 T 5
2	S 1	2	3	4	5
2	S 1	S 1	2	3	4
3	2	2	3	4	5

- **Trick**
  - Expand from **all these sources** to find the shortest path from the existing route to the next target

- **Next: Backtrace as before**
  - Follow pathlengths in decreasing order from target, to **some** source cell

Multi-point Net Walkthrough – VI

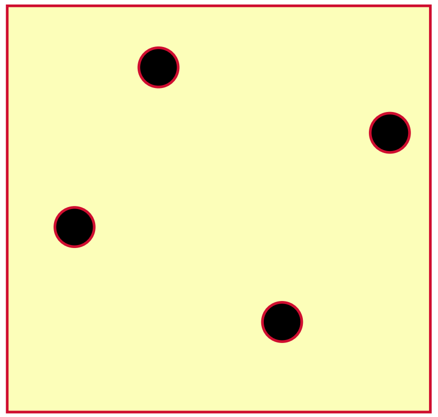


- **Finally**
  - Do usual cleanup
  - Mark all of the segment cells as used and clean-up the grid
  - Now, have embedded a multipoint net, and rendered it an obstacle for future nets

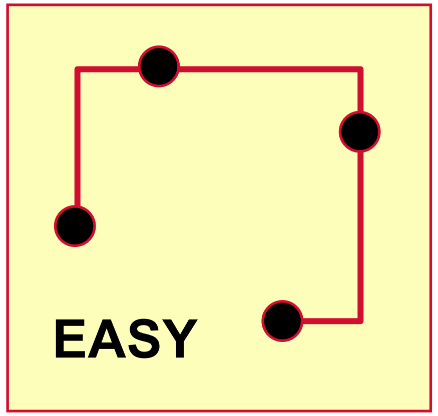
Is this Strategy Optimal?

- **No! This strategy is NOT optimal!**
  - Maybe surprising, but this is just a good heuristic
  - The optimal path has a name: called a **Steiner Tree**
- **How hard is to get the optimal Steiner Tree?**
  - NP-hard! (i.e., computationally hard)
  - Yet another example of why physical design is full of tough, important problems to solve
  - To date, making optimal Steiner Tree is still an active research area
    - Parallel Steiner Tree construction algorithms
    - Machine learning-based algorithms
    - …

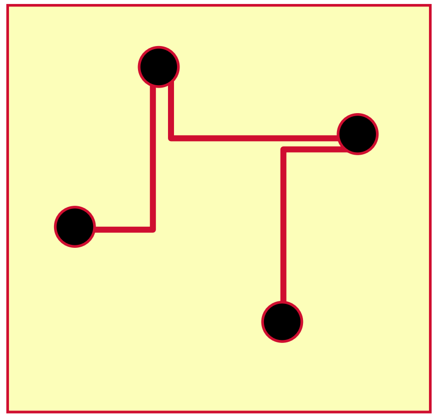
Steiner Tree Constructions



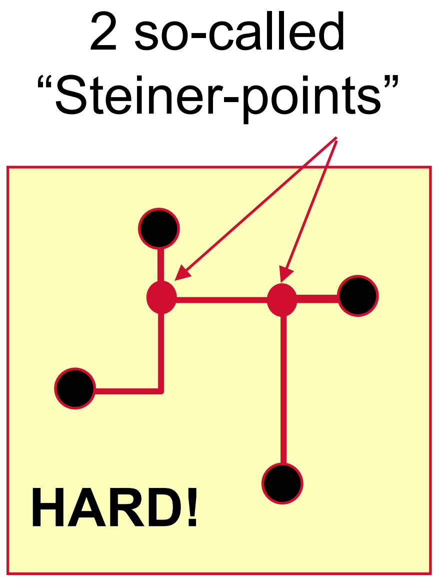
Pins to connect



Route it so we guarantee each 2-point path is shortest; this is **Minimum Spanning Tree**

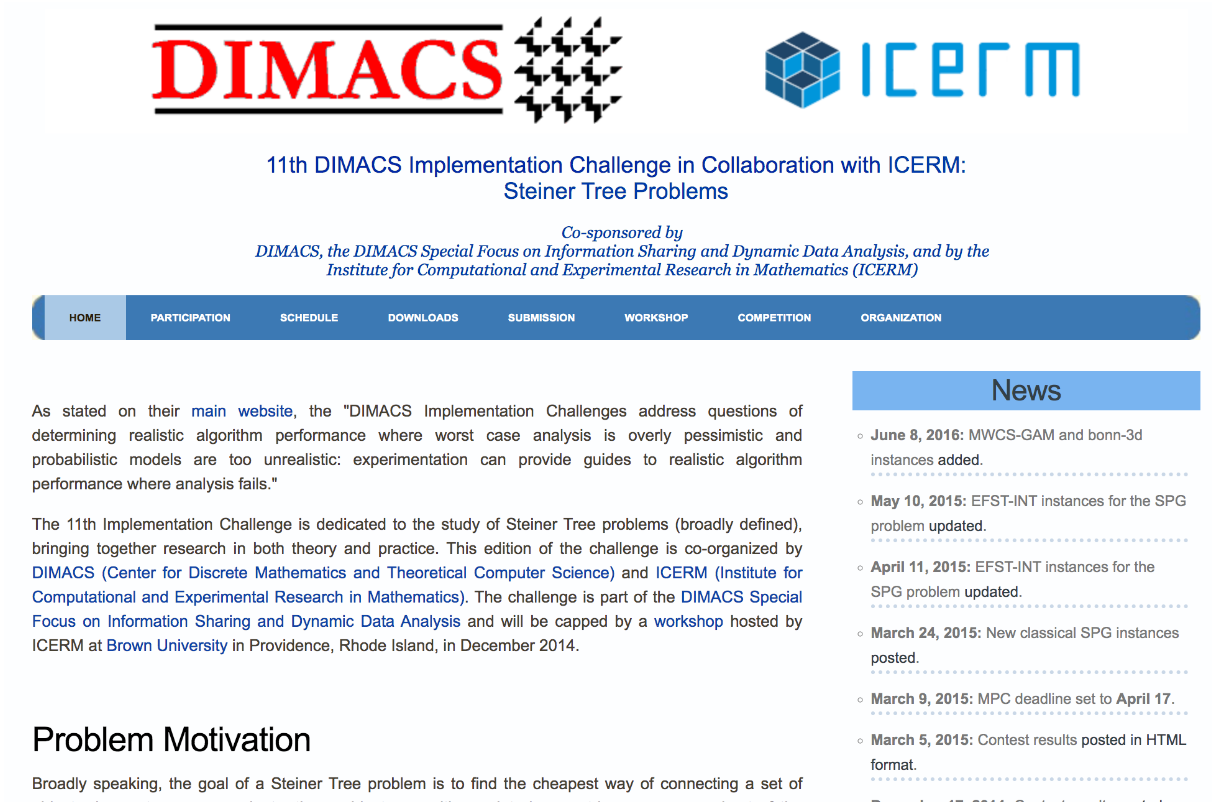


Redraw it--different orientations of 2-point paths



Now we can see the better (shorter) Steiner tree

11th DIMACS Implementation Challenge



11th DIMACS Implementation Challenge in Collaboration with ICERM:
Steiner Tree Problems

Co-sponsored by
DIMACS, the DIMACS Special Focus on Information Sharing and Dynamic Data Analysis, and by the Institute for Computational and Experimental Research in Mathematics (ICERM)

As stated on their main website, the "DIMACS Implementation Challenges address questions of determining realistic algorithm performance where worst case analysis is overly pessimistic and probabilistic models are too unrealistic: experimentation can provide guides to realistic algorithm performance where analysis fails."

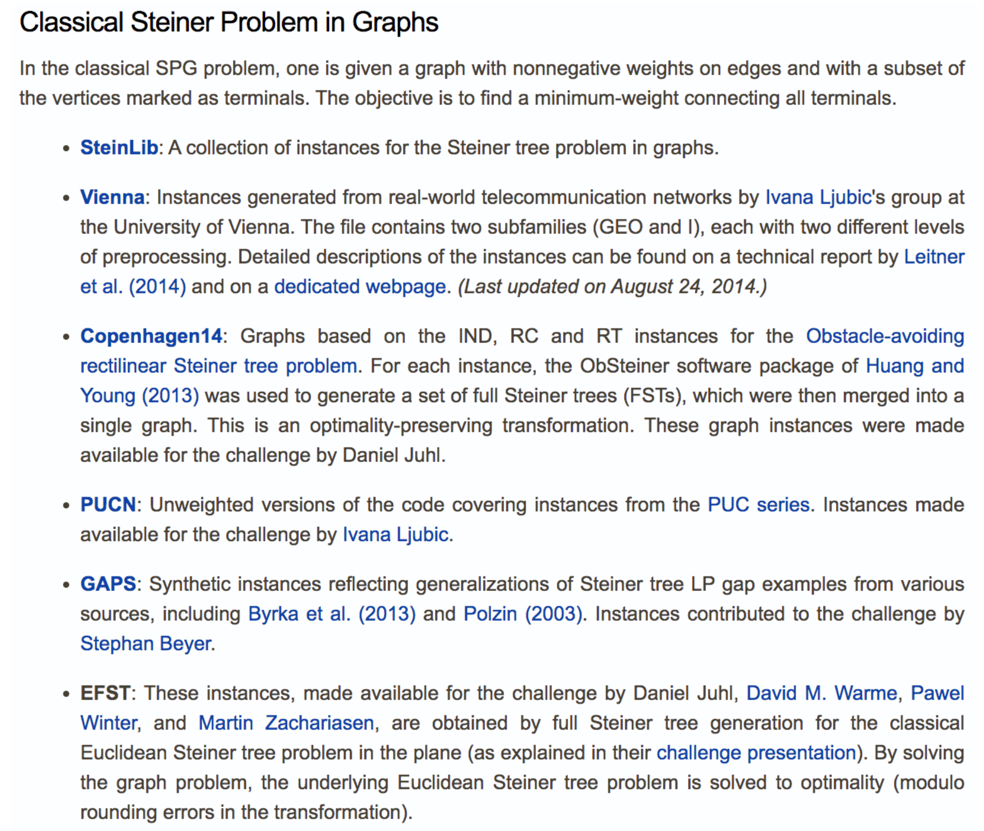
The 11th Implementation Challenge is dedicated to the study of Steiner Tree problems (broadly defined), bringing together research in both theory and practice. This edition of the challenge is co-organized by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science) and ICERM (Institute for Computational and Experimental Research in Mathematics). The challenge is part of the DIMACS Special Focus on Information Sharing and Dynamic Data Analysis and will be capped by a workshop hosted by ICERM at Brown University in Providence, Rhode Island, in December 2014.

Problem Motivation

Broadly speaking, the goal of a Steiner Tree problem is to find the cheapest way of connecting a set of

News

- **June 8, 2016:** MWCS-GAM and bonn-3d instances added.
- **May 10, 2015:** EFST-INT instances for the SPG problem updated.
- **April 11, 2015:** EFST-INT instances for the SPG problem updated.
- **March 24, 2015:** New classical SPG instances posted.
- **March 9, 2015:** MPC deadline set to **April 17**.
- **March 5, 2015:** Contest results posted in HTML format.



Classical Steiner Problem in Graphs

In the classical SPG problem, one is given a graph with nonnegative weights on edges and with a subset of the vertices marked as terminals. The objective is to find a minimum-weight connecting all terminals.

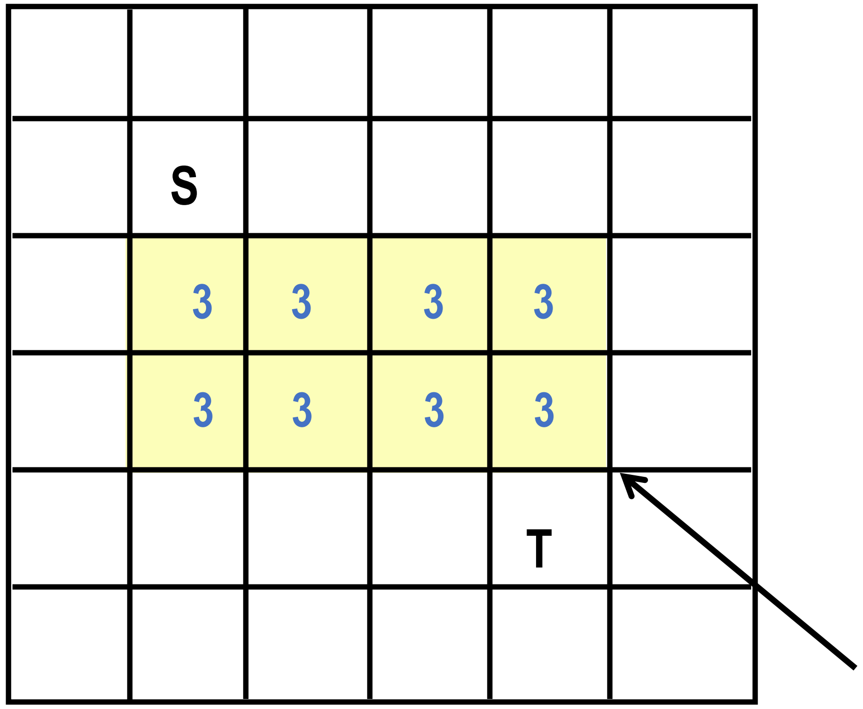
- **SteinLib**: A collection of instances for the Steiner tree problem in graphs.
- **Vienna**: Instances generated from real-world telecommunication networks by Ivana Ljubic's group at the University of Vienna. The file contains two subfamilies (GEO and I), each with two different levels of preprocessing. Detailed descriptions of the instances can be found on a technical report by Leitner et al. (2014) and on a dedicated webpage. *(Last updated on August 24, 2014.)*
- **Copenhagen14**: Graphs based on the IND, RC and RT instances for the Obstacle-avoiding rectilinear Steiner tree problem. For each instance, the ObSteiner software package of Huang and Young (2013) was used to generate a set of full Steiner trees (FSTs), which were then merged into a single graph. This is an optimality-preserving transformation. These graph instances were made available for the challenge by Daniel Juhl.
- **PUCN**: Unweighted versions of the code covering instances from the PUC series. Instances made available for the challenge by Ivana Ljubic.
- **GAPS**: Synthetic instances reflecting generalizations of Steiner tree LP gap examples from various sources, including Byrka et al. (2013) and Polzin (2003). Instances contributed to the challenge by Stephan Beyer.
- **EFST**: These instances, made available for the challenge by Daniel Juhl, David M. Warme, Pawel Winter, and Martin Zachariasen, are obtained by full Steiner tree generation for the classical Euclidean Steiner tree problem in the plane (as explained in their challenge presentation). By solving the graph problem, the underlying Euclidean Steiner tree problem is solved to optimality (modulo rounding errors in the transformation).

11th DIMACS Implementation Challenge in Collaboration with ICERM: Steiner Tree Problems: https://dimacs11.zib.de/home.html

How to Handle Non-unit Grids?

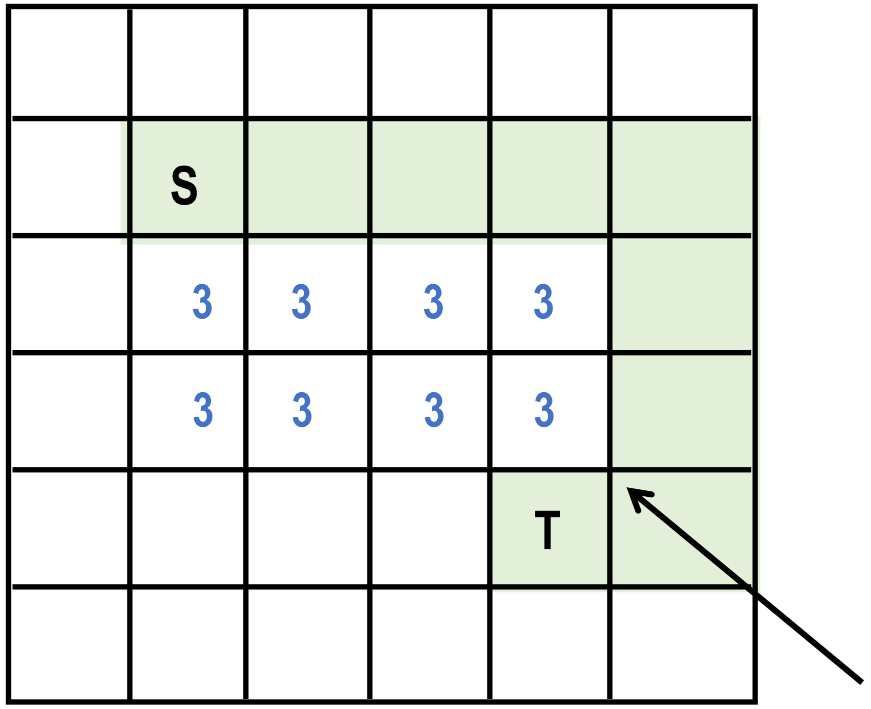
- **Practical maze routing (routing area) are non-unit**
  - Wires should avoid going through high-congested areas
  - Wires should avoid going through blocked areas
  - Wires should avoid going through constrained areas
  - …
- **Can we apply maze routing algorithm still?**

Maze Router on Non-unit Grids



- **Old problem**
  - Each cell in grid costs the **same** to cross it with a wire
  - Cost ==1, unit-cost
  - Is this ***necessary***? No!
- **Now, in practice**
  - Given grid, Source and target
  - Different weights for each cell
- **Problem formulation**
  - Find minimum cost path (i.e., shortest path) connecting source and target.

Maze Router on Non-unit Grids (cont'd)



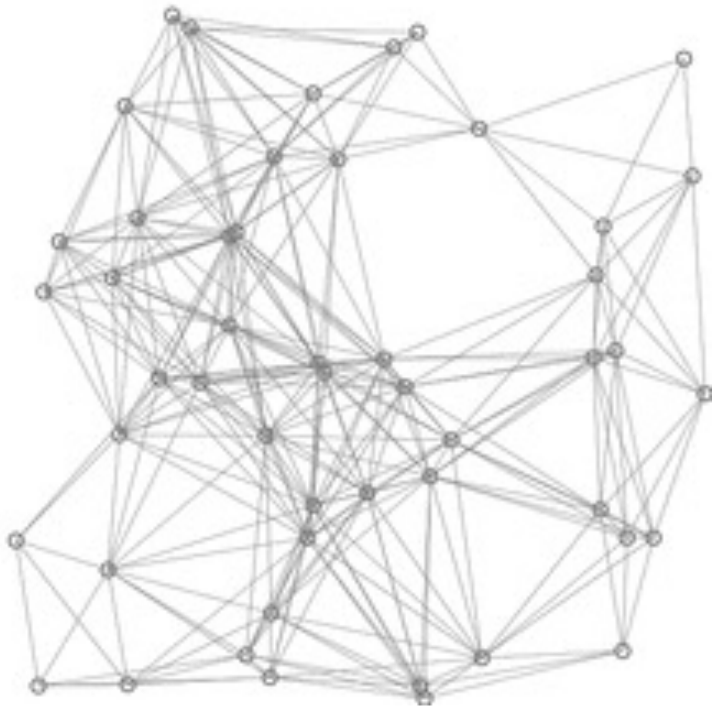
- **Old problem**
  - Each cell in grid costs the **same** to cross it with a wire
  - Cost ==1, unit-cost
  - Is this ***necessary***? No!
- **Now, in practice**
  - Given grid, Source and target
  - Different weights for each cell
- **Problem formulation**
  - Find minimum cost path (i.e., shortest path) connecting source and target.

Recap: Graph Algorithm Lecture

- **Shortest-path-faster algorithm (SPFA)**
  - Only perform relaxation from active vertices
  - Largely reduce the relaxation times

```
procedure SPFA(G, s)
for each vertex v ≠ s in V(G)
    d(v) := ∞
d(s) := 0
push s into Q
while Q is not empty do
    u := Q.pop()
    for each edge (u, v) in E(G) do
        if d(u) + w(u, v) < d(v) then
            d(v) := d(u) + w(u, v)
            if v is not in Q then
                push v into Q
```

Relaxation happens only at active vertices (in queue) – *largely reduced redundant relaxations!*



red lines are shortest path covering
blue lines are relaxations

From SPFA to Dijkstra Algorithm

```
d[s] ← 0
for each v Î V – {s}
    do d[v] ← ¥
S ← ∅
Q ← V            ⊳ Q Replacing the SPFA queue with min priority queue!
while Q ¹ ∅
    do u ← EXTRACT-MIN(Q)
       S ← S È {u}
       for each v Î Adj[u]
           do if d[v] > d[u] + w(u, v)
                 then d[v] ← d[u] + w(u, v)
                      p[v] ← u
```

Relaxation happens only at the active vertex with the minimum (shortest) value discovered so far! (i.e., vertex *u*)

Summary

- We have discussed the routing problem
- We have discussed 2-pin net maze routing algorithm
- We have discussed n-pin net maze routing algorithm
- We have discussed problems of maze routing algorithm
- We have discussed optimal “Steiner Tree” router
- We have discussed weighted maze routing algorithm