Lecture 13 –Global Routing -2

Logistics

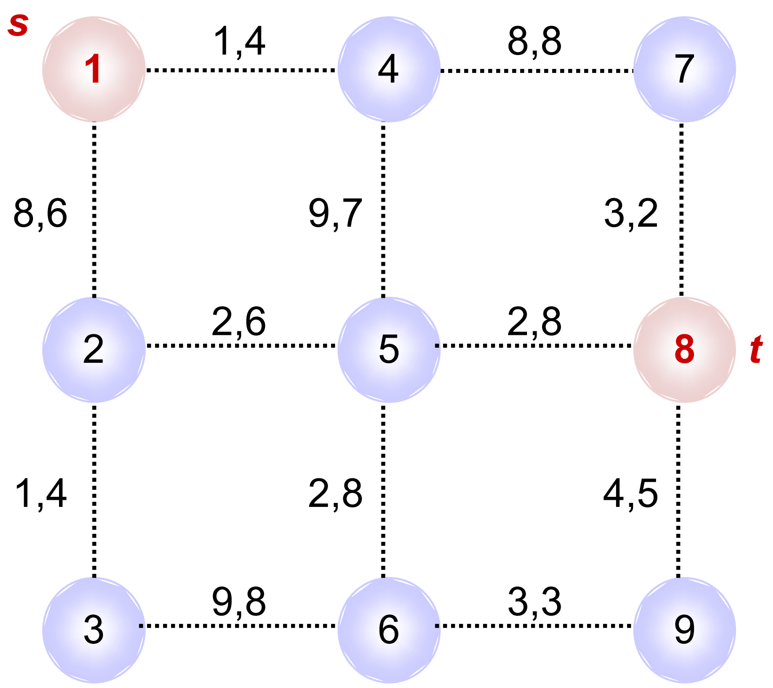
- Some of you do not have not found a project partner.
  - Please use Piazza to check if anyone is still available
  - There are odd number of students in class → at least one student will have to do it solo
  - Doing it solo is not a bad idea (you get extra credit)
- Office Hour this week postponed to Friday, 4pm (sorry, doctor appointment)

Finding Shortest Paths with Dijkstra's Algorithm

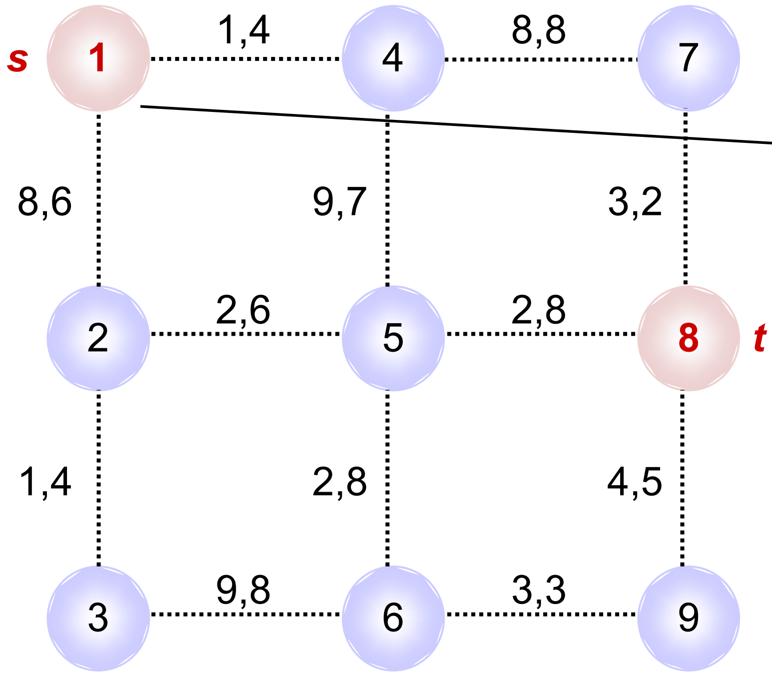
- Finds a shortest path between two specific nodes in the routing graph
- Input
  - graph $G(V,E)$ with non-negative *edge weights* W,
  - *source* (starting) node s, and
  - *target* (ending) node t
- Maintains three groups of nodes
  - Group 1 – contains the nodes that have not yet been visited
  - Group 2 – contains the nodes that have been visited but for which the shortest-path cost from the starting node has not yet been found
  - Group 3 – contains the nodes that have been visited and for which the shortest path cost from the starting node has been found
- Once t is in Group 3, the algorithm finds the shortest path by backtracing

Finding Shortest Paths with Dijkstra's Algorithm

Example



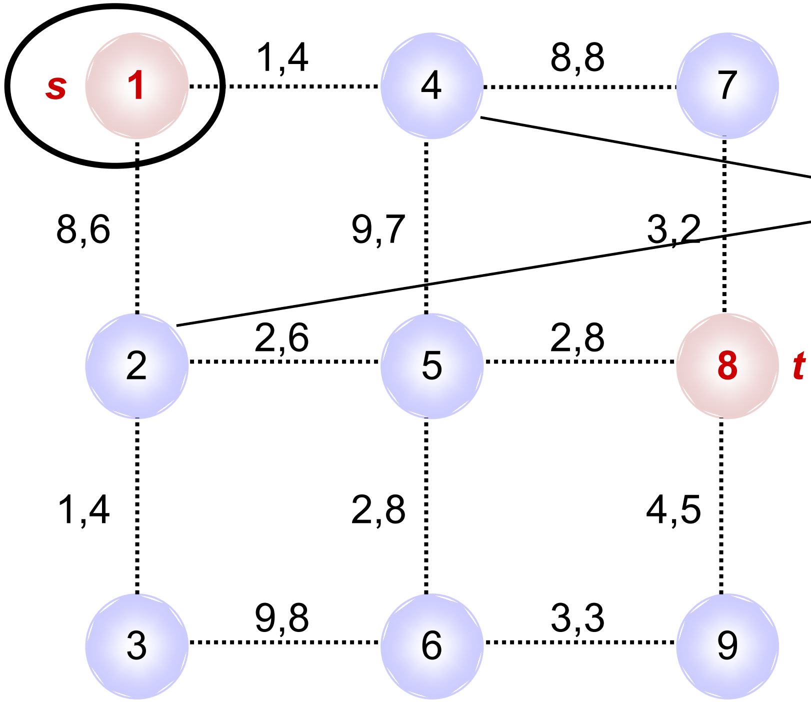
Find the shortest path from source *s* to target *t* where the path cost $\sum w_1 + \sum w_2$ is minimal



Group 2	Group 3

(1)

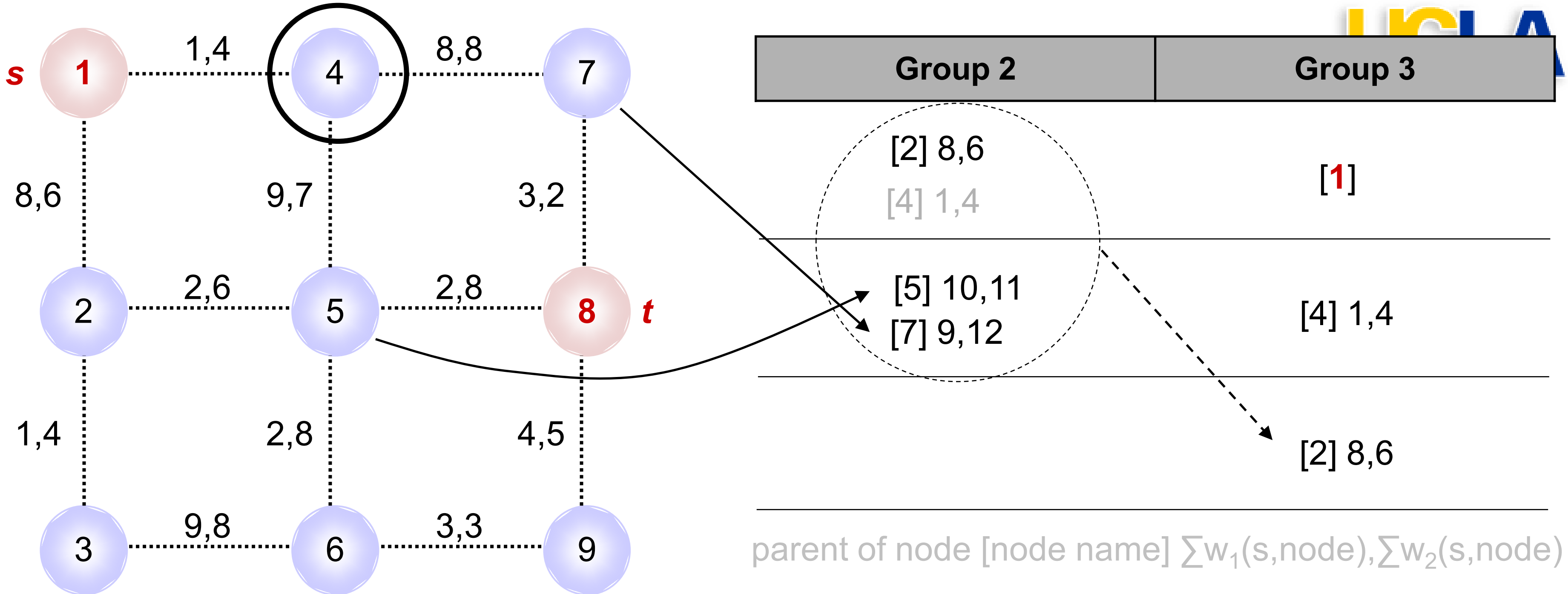
Current node: 1



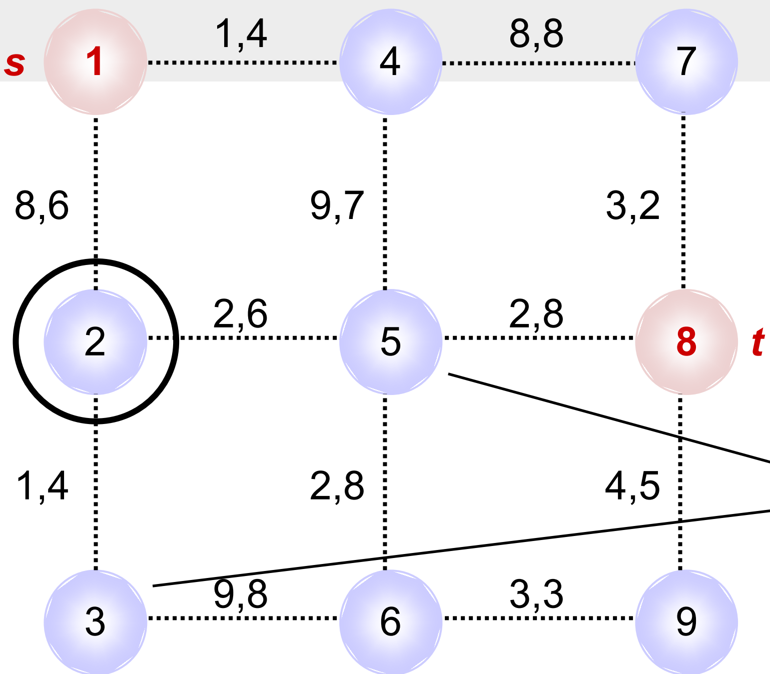
Group 2	Group 3
[2] 8,6 [4] 1,4	[1]
	[4] 1,4

parent of node [node name] $\sum w_1(s,node),\sum w_2(s,node)$

Current node: 1
Neighboring nodes: 2, 4
Minimum cost in group 2: node 4



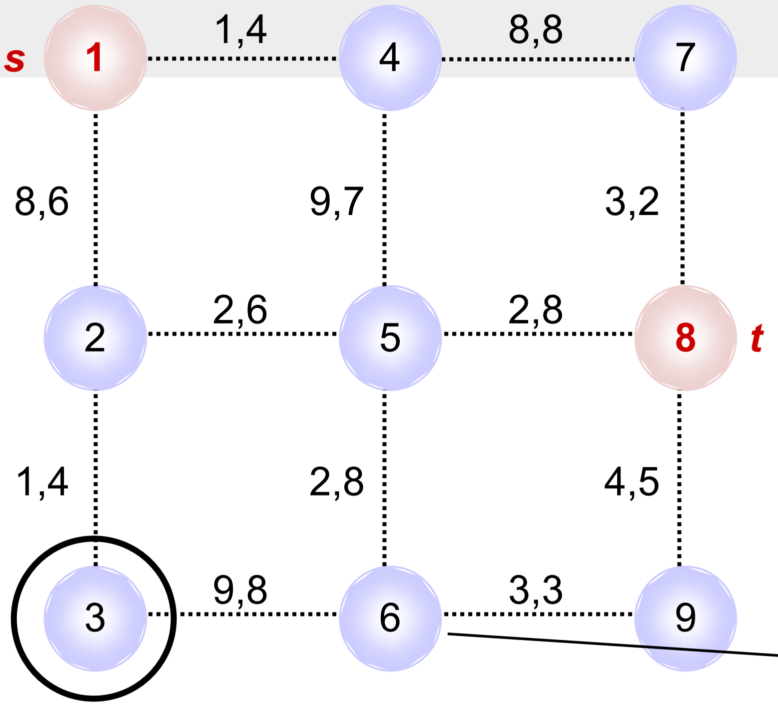
Current node: 4
Neighboring nodes: ~~1~~, 5, 7
Minimum cost in group 2: node 2



Group 2	Group 3
[2] 8,6 [4] 1,4	[1]
[5] 10,11 [7] 9,12	[4] 1,4
[3] 9,10 ~~[5] 10,12~~	[2] 8,6
	[3] 9,10

parent of node [node name] $\sum w_1(s,node),\sum w_2(s,node)$

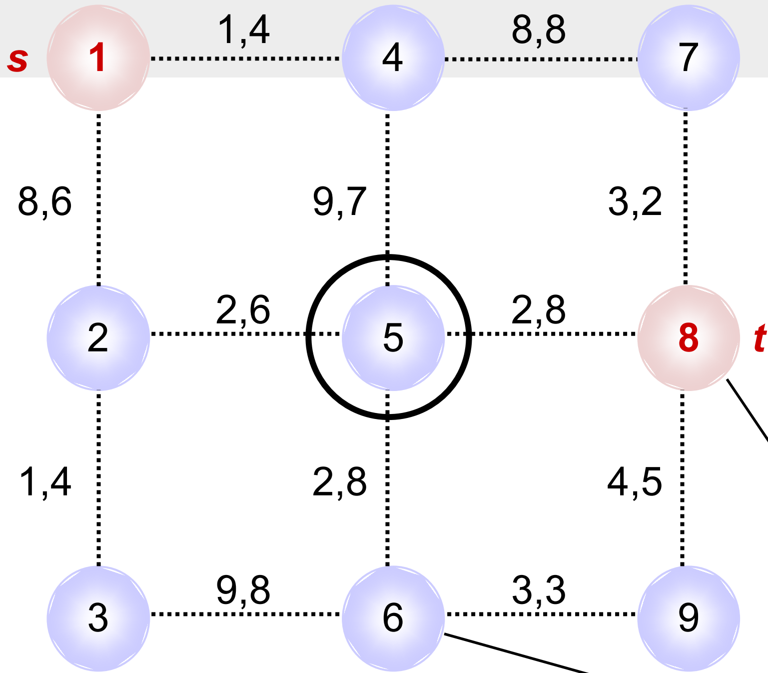
Current node: 2
Neighboring nodes: ~~1~~, 3, 5
Minimum cost in group 2: node 3



Group 2	Group 3
[2] 8,6 [4] 1,4	[1]
[5] 10,11 [7] 9,12	[4] 1,4
[3] 9,10 ~~[5] 10,12~~	[2] 8,6
[6] 18,18	[3] 9,10
	[5] 10,11

parent of node [node name] $\sum w_1(s,node),\sum w_2(s,node)$

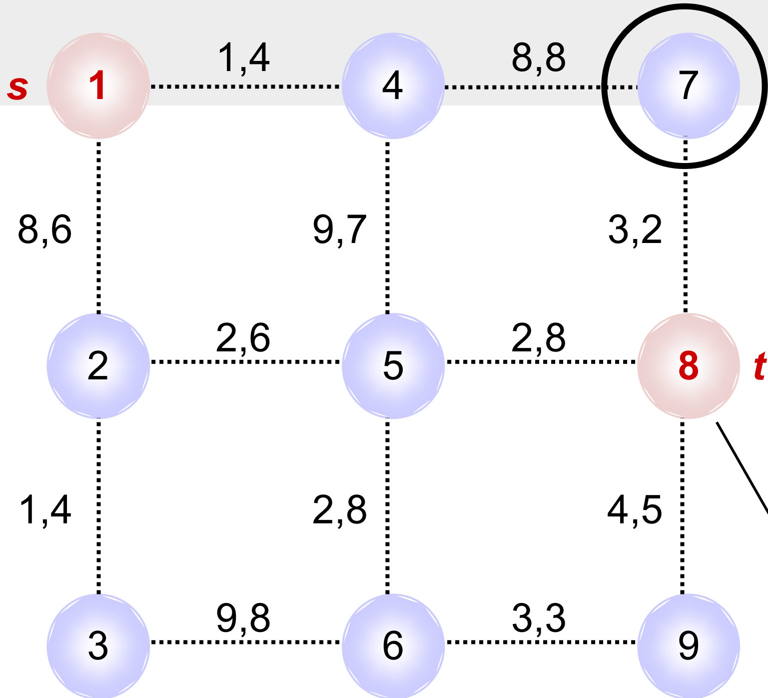
Current node: 3
Neighboring nodes: ~~2~~, 6
Minimum cost in group 2: node 5



Group 2	Group 3
[2] 8,6 [4] 1,4	[1]
[5] 10,11 [7] 9,12	[4] 1,4
[3] 9,10 ~~[5] 10,12~~	[2] 8,6
~~[6] 18,18~~	[3] 9,10
[6] 12,19 [8] 12,19	[5] 10,11
	[7] 9,12

parent of node [node name] $\sum w_1(s,node),\sum w_2(s,node)$

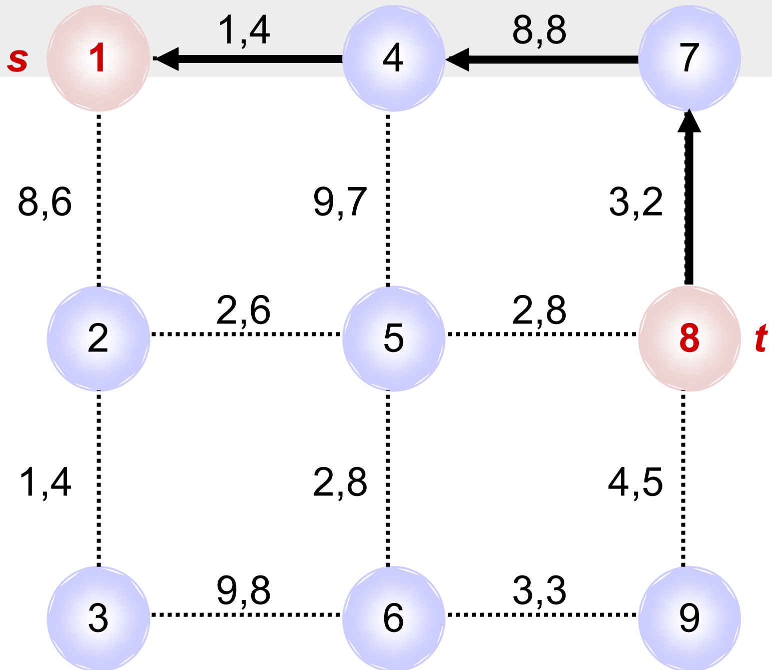
Current node: 5
Neighboring nodes: ~~2~~, ~~4~~, 6, 8
Minimum cost in group 2: node 7



Current node: 7
Neighboring nodes: ~~4~~, 8
Minimum cost in group 2: node 8

Group 2	Group 3
(2) 8,6 (4) 1,4	(1)
(5) 10,11 (7) 9,12	(4) 1,4
(3) 9,10 ~~(5) 10,12~~	(2) 8,6
(6) 18,18	(3) 9,10
(6) 12,19 ~~(8) 12,19~~	(5) 10,11
(8) 12,14	(7) 9,12
	(8) 12,14

parent of node [node name] $\sum w_1(s,node),\sum w_2(s,node)$



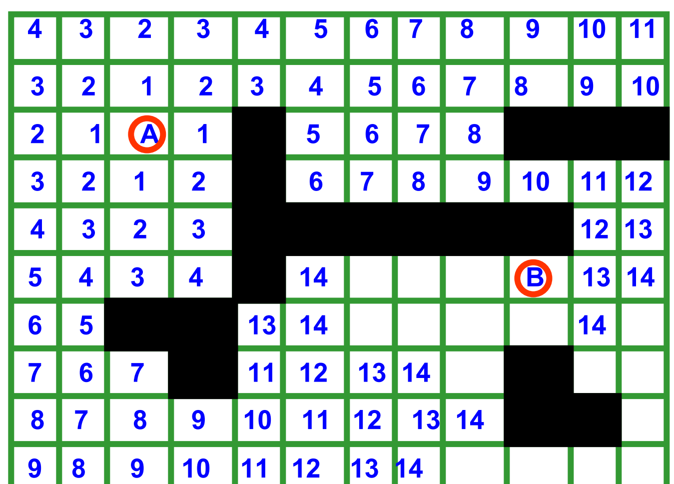
Retrace from *t* to *s*

Optimal path 1-4-7-8 from *s* to *t* with accumulated cost (12,14)

Group 2	Group 3
(2) 8,6 (4) 1,4	(1)
(5) 10,11 (7) 9,12	(4) 1,4
(3) 9,10 ~~(5) 10,12~~	(2) 8,6
(6) 18,18	(3) 9,10
(6) 12,19 ~~(8) 12,19~~	(5) 10,11
(8) 12,14	(7) 9,12
	(8) 12,14

Maze Routing

- Point to point routing of nets
- Route from source to sink
- Basic idea = wave propagation (Lee, 1961)
  - Breadth-first search + back-tracing after finding shortest path
  - Guarantees to find the shortest path
- Objective = route all nets according to some cost function that minimizes congestion, route length, coupling, etc.



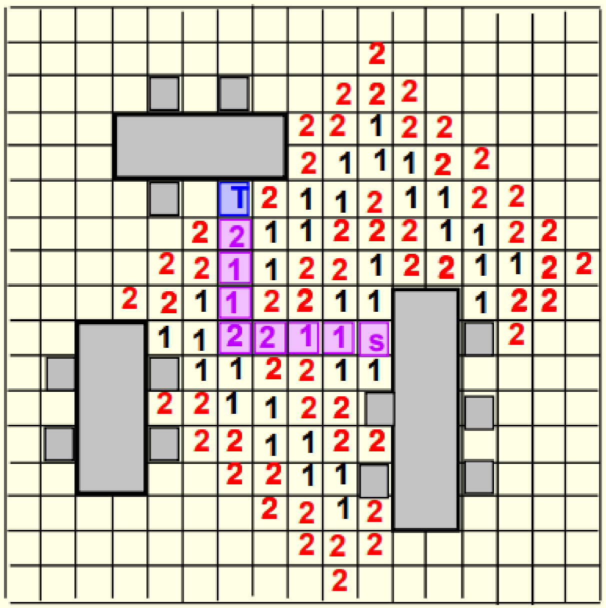
Courtesy K. Keutzer et al. UCB

Maze Routing

Slow: for each net, we have to search $M \times N$ grid.

<u>Improvements</u>

- Simple modification (Akers, 1967)
  - All one wants is previous label to be different from next label
  - use 1122 labeling
  - reduce the memory requirement per vertex (1, 2, empty, blocked → 2 bits instead of log(m+n)
  - also need to search $M \times N$ grid

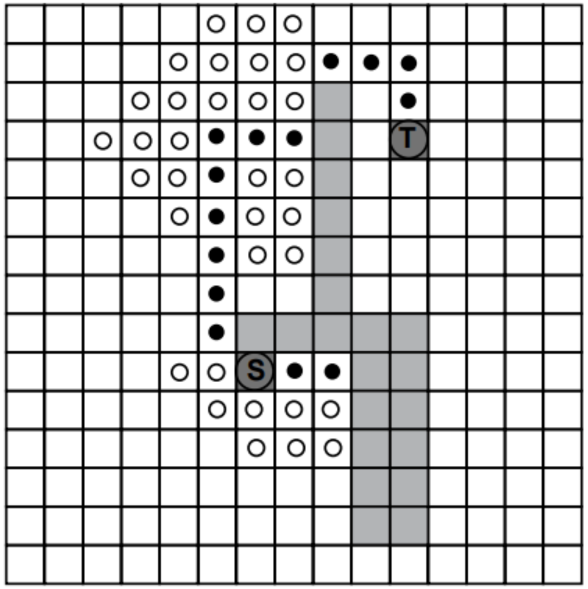


Sequence: 1, 1, 2, 2, 1, 1, 2, 2, ...

Figure courtesy Hai Zhou@Northwestern

Soukup's Algorithm

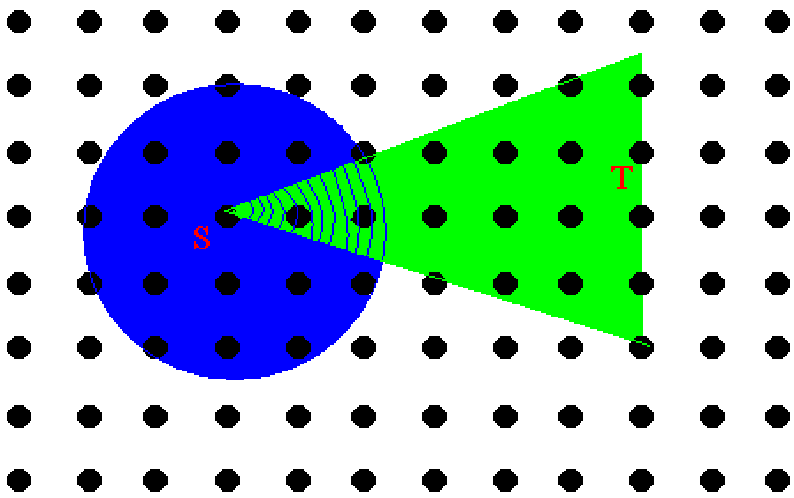
- Iterative algorithm (Soukup, 1978)
  - explore in the direction toward the target
    - Draw a line toward target in go in that direction. (DFS)
    - If stuck do Lee-style BFS till you find a grid in target direction.
    - draw a line toward target again
  - First use DFS, when get to an obstacle, use BFS to get around
  - No guarantee to find the shortest path
  - speedup Lee-maze by 10x to 50x



Puneet Gupta (puneet@ee.ucla.edu)

Directed Search

- Add <distance to sink> to cost function → directed search
  - Similar to Soukup
  - Allows maze router to explore points around direct source-sink path first
  - A* search (Hart, Nilsson and Raphael, 1968) = Best-First Search: expand from node w/min f = g +h; g = current cost, h = LB on future cost
    - If h is always a lower bound → optimal (will always find min-cost path)
  - Bidirectional A* search: nodes are expanded from both the source and target until the two expansion regions intersect



S denotes the source point
T is the sink point

Directed search limits the search space when all other cost variables are equal

Non directed search expands in a circular fashion from S

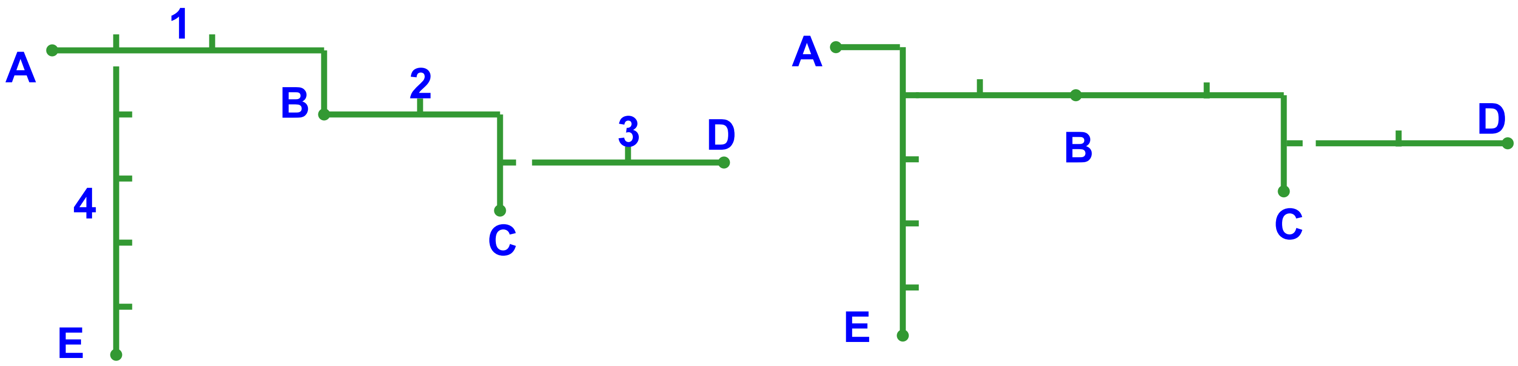
Directed search expands in a conical fashion from S to T

Connecting Multi-Terminal Nets

In general, maze routing is not well-suited to multi-terminal nets

Several attempts made to extend to multi-terminal nets

- Connect one terminal at a time
- Use the entire connected subtrees as sources or targets during expansion
- Ripup/Reroute to improve solution quality (remove a segment and re-connect)



- Results are sub-optimal
- Inherit time and memory cost of maze algorithms

5 min break

Full-Netlist Routing

- Global routers must properly match nets with routing resources, without oversubscribing resources in any part of the chip
- Signal nets are either routed
  - simultaneously, e.g., by integer linear programming, or
  - sequentially, e.g., one net at a time
- When certain nets cause resource contention or overflow for routing edges, sequential routing requires multiple iterations: rip-up and reroute

Routing by Integer Linear Programming

- A linear program (LP) consists
  - of a set of *linear* constraints and
  - an optional *linear* objective function
- Objective function is maximized or minimized
- Integer linear program (ILP): linear program where every variable can only assume integer values
  - Typically takes much longer to solve
  - In many cases, variables are only allowed values 0 and 1
- Several ways to formulate the global routing problem as an ILP, one of which is presented next

Routing by Integer Linear Programming

- Three inputs
  - $W \times H$ routing grid G,
  - Routing edge capacities, and
  - Netlist
- Two sets of variables
  - k Boolean variables $x_{net1}, x_{net2}, \ldots, x_{netk}$, each of which serves as an indicator for one of k specific paths or route options, for each net *net* in Netlist
  - k real variables $w_{net1}, w_{net2}, \ldots, w_{netk}$, each of which represents a net weight for a specific route option for *net* in Netlist
- Two types of constraints
  - Each net must select a single route (mutual exclusion)
  - Number of routes assigned to each edge (total usage) cannot exceed its capacity

Routing by Integer Linear Programming

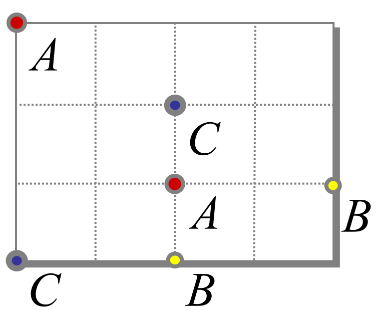
- Inputs
  - W,H: width W and height H of routing grid G
  - $G(i,j)$: grid cell at location (i,j) in routing grid G
  - $\sigma(G(i,j) \sim G(i+1,j))$: capacity of horizontal edge $G(i,j) \sim G(i+1,j)$
  - $\sigma(G(i,j) \sim G(i,j+1))$: capacity of vertical edge $G(i,j) \sim G(i,j+1)$
  - *Netlist*: netlist
- Variables
  - $x_{net1}, \ldots, x_{netk}$: k Boolean path variables for each net *net* in *Netlist*
  - $w_{net1}, \ldots, w_{netk}$: k net weights, one for each path of net *net* in *Netlist*
- Maximize

$$\sum_{net \in Netlist} w_{net_1} \cdot x_{net_1} + \ldots + w_{net_k} \cdot x_{net_k}$$

- Subject to
  - Variable ranges
  - Net constraints
  - Capacity constraints

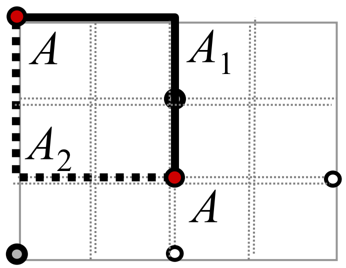
Routing by Integer Linear Programming – Example

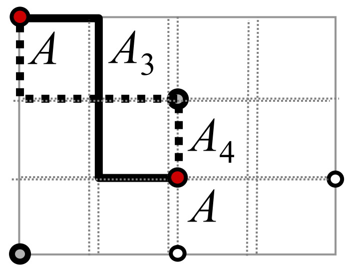
- Given
  - Nets *A*, *B*, *C*
  - $W = 5 \times H = 4$ routing grid G
  - $\sigma(e) = 1$ for all $e \in G$
  - *L*-shapes have weight 1.00 and *Z*-shapes have weight 0.99
  - The lower-left corner is (0,0).
- Task
  - Write the ILP to route the nets in the graph below



Routing by Integer Linear Programming – Example

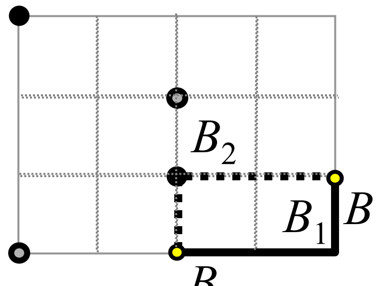
- Solution
  - For net A, the possible routes are two L-shapes (A_1,A_2) and two Z-shapes (A_3,A_4)

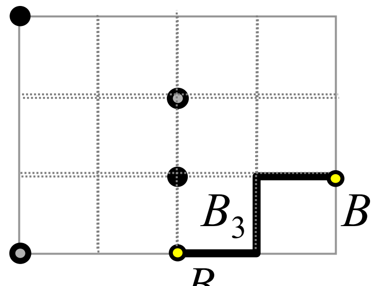




Net Constraints:
$x_{A1} + x_{A2} + x_{A3} + x_{A4} \leq 1$
Variable Constraints:
$0 \leq x_{A1} \leq 1$, $0 \leq x_{A2} \leq 1$,
$0 \leq x_{A3} \leq 1$, $0 \leq x_{A4} \leq 1$

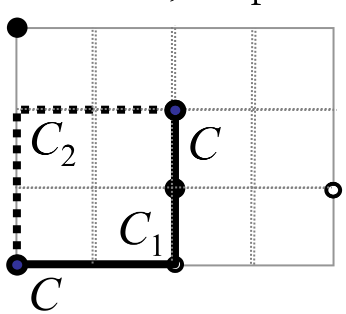
  - For net B, the possible routes are two L-shapes (B_1,B_2) and one Z-shape (B_3)

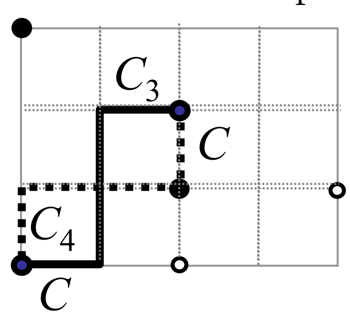




Net Constraints:
$x_{B1} + x_{B2} + x_{B3} \leq 1$
Variable Constraints:
$0 \leq x_{B1} \leq 1$, $0 \leq x_{B2} \leq 1$,
$0 \leq x_{B3} \leq 1$

  - For net C, the possible routes are two L-shapes (C_1,C_2) and two Z-shapes (C_3,C_4)





Net Constraints:
$x_{C1} + x_{C2} + x_{C3} + x_{C4} \leq 1$
Variable Constraints:
$0 \leq x_{C1} \leq 1$, $0 \leq x_{C2} \leq 1$,
$0 \leq x_{C3} \leq 1$, $0 \leq x_{C4} \leq 1$

Routing by ILP– Example

Horizontal Edge Capacity Constraints:

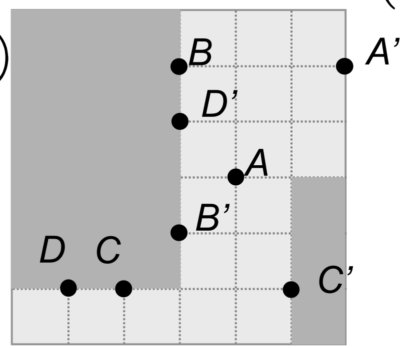
$G(0,0) \sim G(1,0)$:	$x_{C1} + x_{C3}$	$\leq$	$\sigma(G(0,0) \sim G(1,0)) = 1$
$G(1,0) \sim G(2,0)$:	x_{C1}	$\leq$	$\sigma(G(1,0) \sim G(2,0)) = 1$
$G(2,0) \sim G(3,0)$:	$x_{B1} + x_{B3}$	$\leq$	$\sigma(G(2,0) \sim G(3,0)) = 1$
$G(3,0) \sim G(4,0)$:	x_{B1}	$\leq$	$\sigma(G(3,0) \sim G(4,0)) = 1$
$G(0,1) \sim G(1,1)$:	$x_{A2} + x_{C4}$	$\leq$	$\sigma(G(0,1) \sim G(1,1)) = 1$
$G(1,1) \sim G(2,1)$:	$x_{A2} + x_{A3} + x_{C4}$	$\leq$	$\sigma(G(1,1) \sim G(2,1)) = 1$
$G(2,1) \sim G(3,1)$:	x_{B2}	$\leq$	$\sigma(G(2,1) \sim G(3,1)) = 1$
$G(3,1) \sim G(4,1)$:	$x_{B2} + x_{B3}$	$\leq$	$\sigma(G(3,1) \sim G(4,1)) = 1$
$G(0,2) \sim G(1,2)$:	$x_{A4} + x_{C2}$	$\leq$	$\sigma(G(0,2) \sim G(1,2)) = 1$
$G(1,2) \sim G(2,2)$:	$x_{A4} + x_{C2} + x_{C3}$	$\leq$	$\sigma(G(1,2) \sim G(2,2)) = 1$
$G(0,3) \sim G(1,3)$:	$x_{A1} + x_{A3}$	$\leq$	$\sigma(G(0,3) \sim G(1,3)) = 1$
$G(1,3) \sim G(2,3)$:	x_{A1}	$\leq$	$\sigma(G(1,3) \sim G(2,3)) = 1$

Vertical Edge Capacity Constraints:

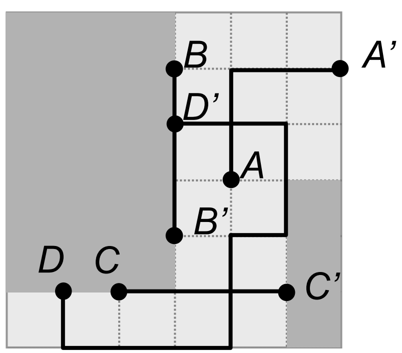
$G(0,0) \sim G(0,1)$:	$x_{C2} + x_{C4}$	$\leq$	$\sigma(G(0,0) \sim G(0,1)) = 1$
$G(1,0) \sim G(1,1)$:	x_{C3}	$\leq$	$\sigma(G(1,0) \sim G(1,1)) = 1$
$G(2,0) \sim G(2,1)$:	$x_{B2} + x_{C1}$	$\leq$	$\sigma(G(2,0) \sim G(2,1)) = 1$
$G(3,0) \sim G(3,1)$:	x_{B3}	$\leq$	$\sigma(G(3,0) \sim G(3,1)) = 1$
$G(4,0) \sim G(4,1)$:	x_{B1}	$\leq$	$\sigma(G(4,0) \sim G(4,1)) = 1$
$G(0,1) \sim G(0,2)$:	$x_{A2} + x_{C2}$	$\leq$	$\sigma(G(0,1) \sim G(0,2)) = 1$
$G(1,1) \sim G(1,2)$:	$x_{A3} + x_{C3}$	$\leq$	$\sigma(G(1,1) \sim G(1,2)) = 1$
$G(2,1) \sim G(2,2)$:	$x_{A1} + x_{A4} + x_{C1} + x_{C4}$	$\leq$	$\sigma(G(2,1) \sim G(2,2)) = 1$
$G(0,2) \sim G(0,3)$:	$x_{A2} + x_{A4}$	$\leq$	$\sigma(G(0,2) \sim G(0,3)) = 1$
$G(1,2) \sim G(1,3)$:	x_{A3}	$\leq$	$\sigma(G(1,2) \sim G(1,3)) = 1$
$G(2,2) \sim G(2,3)$:	x_{A1}	$\leq$	$\sigma(G(2,2) \sim G(2,3)) = 1$

Rip-Up and Reroute (RRR)

- Rip-up and reroute (RRR) framework: focuses on hard-to-route nets
- Idea: allow temporary violations, so that all nets are routed, but then iteratively remove some nets (rip-up), and route them differently (reroute)

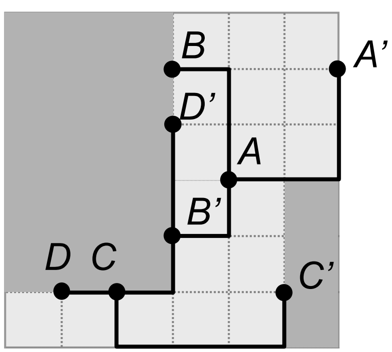


Routing without allowing violations



WL = 21

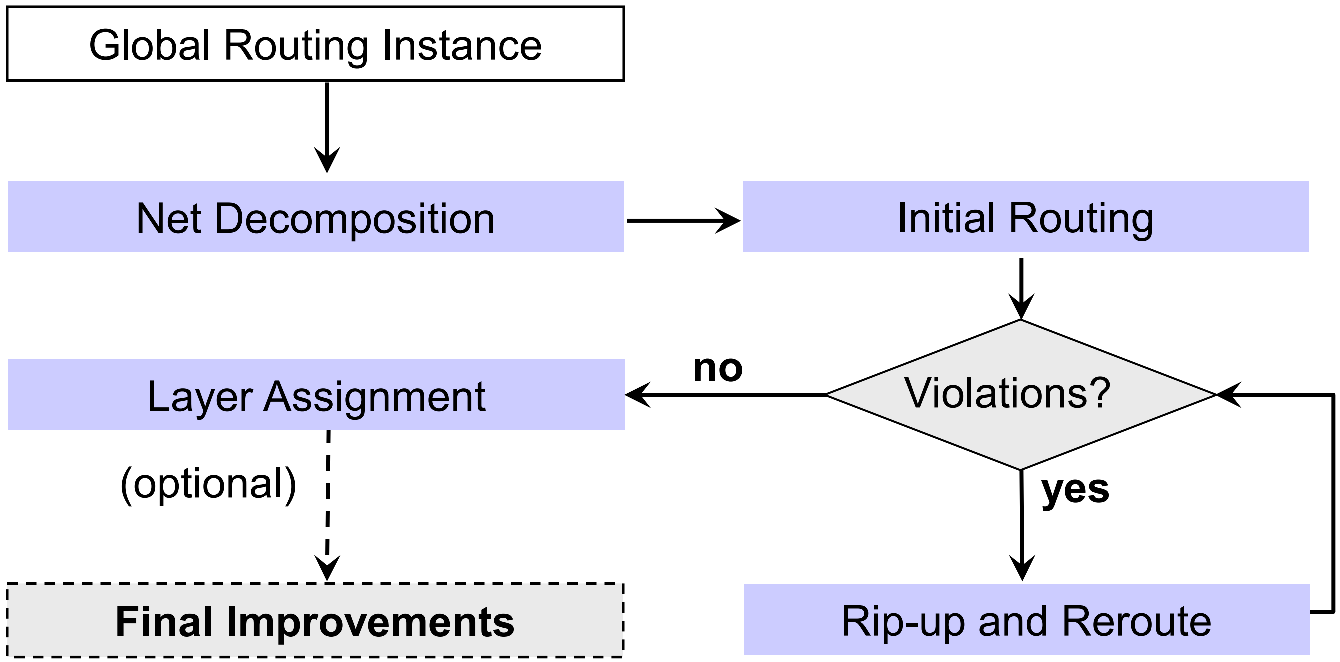
Routing with allowing violations and RRR



WL = 19

Modern Global Routing

- General flow for modern global routers, where each router uses a unique set of optimizations:



Modern Global Routing

- Initial routes are constructed quickly by pattern routing and Steiner tree construction
  - For each net, considers only a small number of shapes (L, Z, U, T, E)
  - Very fast, but misses many opportunities
- The main part of the router is based on a variant of rip-up reroute called Negotiated-Congestion Routing (NCR)
  - NCR maintains "history" in terms of which regions attracted too many nets
  - NCR increases routing cost according to the historical popularity of the regions
    - The nets with alternative routes are forced to take those routes
    - The nets that do not have good alternatives remain unchanged

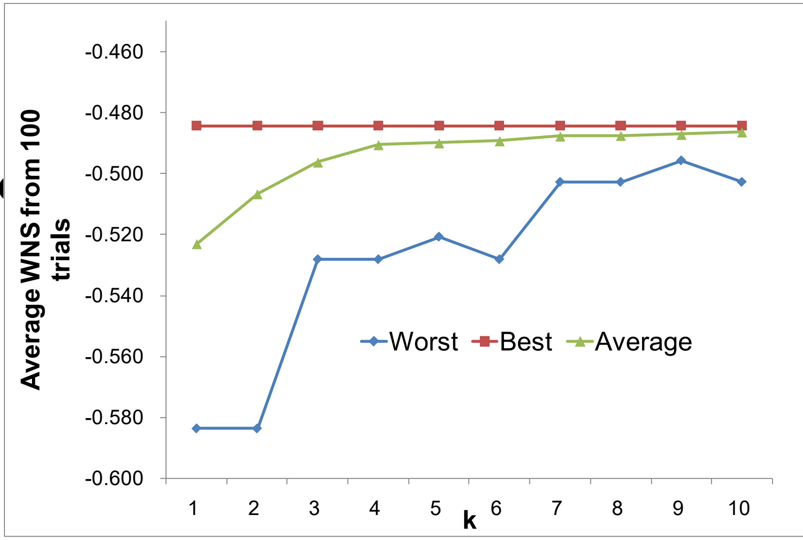
Some Points about EDA Tools

Q: Do tools give different answers when you run them multiple times?

A: Maybe, but why would that be bad?

Noise in Production Design Flow

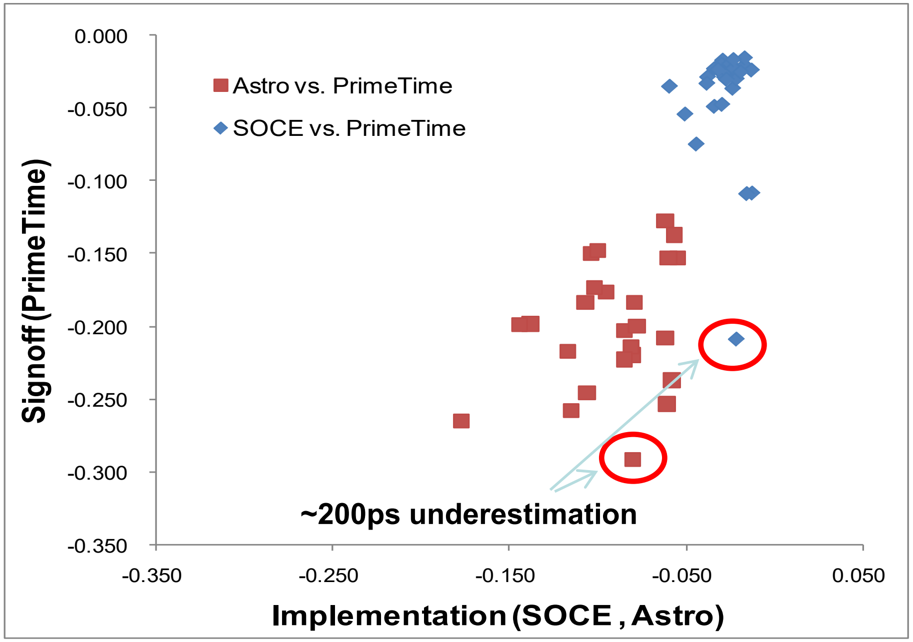
- Two major sources of noise
  - Miscorrelation in parasitics extractor and timer
  - Suboptimality of heuristic optimization engines
    - Most design optimization problems are NP-hard → Heuristic approaches have been used
    - Heuristics lead to "NOISE" that creates variability in solution quality
- Exploting noise in design flow
  - Use idle machines: we can choose
- best solution among N different soluti
  - Example: Best-of-k method



Miscorrelation: Implementation vs. Signoff

- Experiment setup
  - Testcases:
    - aes_cipher_top
    - jpeg_encoder
  - Tools
    - SOCE / Astro
- Results
  - Most cases, P&R tools underestimate timing slack; increasing TAT needed to fix violations at signoff
  - There is no clear trend – i.e., not clear what factors cause miscorrelation
- Conventional approaches
  - RC derating in implementation tools to have pessimistic delay

Worst negative slack comparison From 29 testcases



Inherent Noise: Ignorable Perturbation vs. Results

- Slight changes in design constraints can make significant difference in final timing
- Possible knobs to perturb in design constraints
  - Clock cycle time
  - Clock uncertainty
  - IO delay constrants
  - RC values
- Loose timing constraints do not always improve timing
  - 0.1ps change in constraint → > 50ps change in signoff timing
- Noise is really random! → Difficult to predict

Inherent Noise: Example Results

UCLA

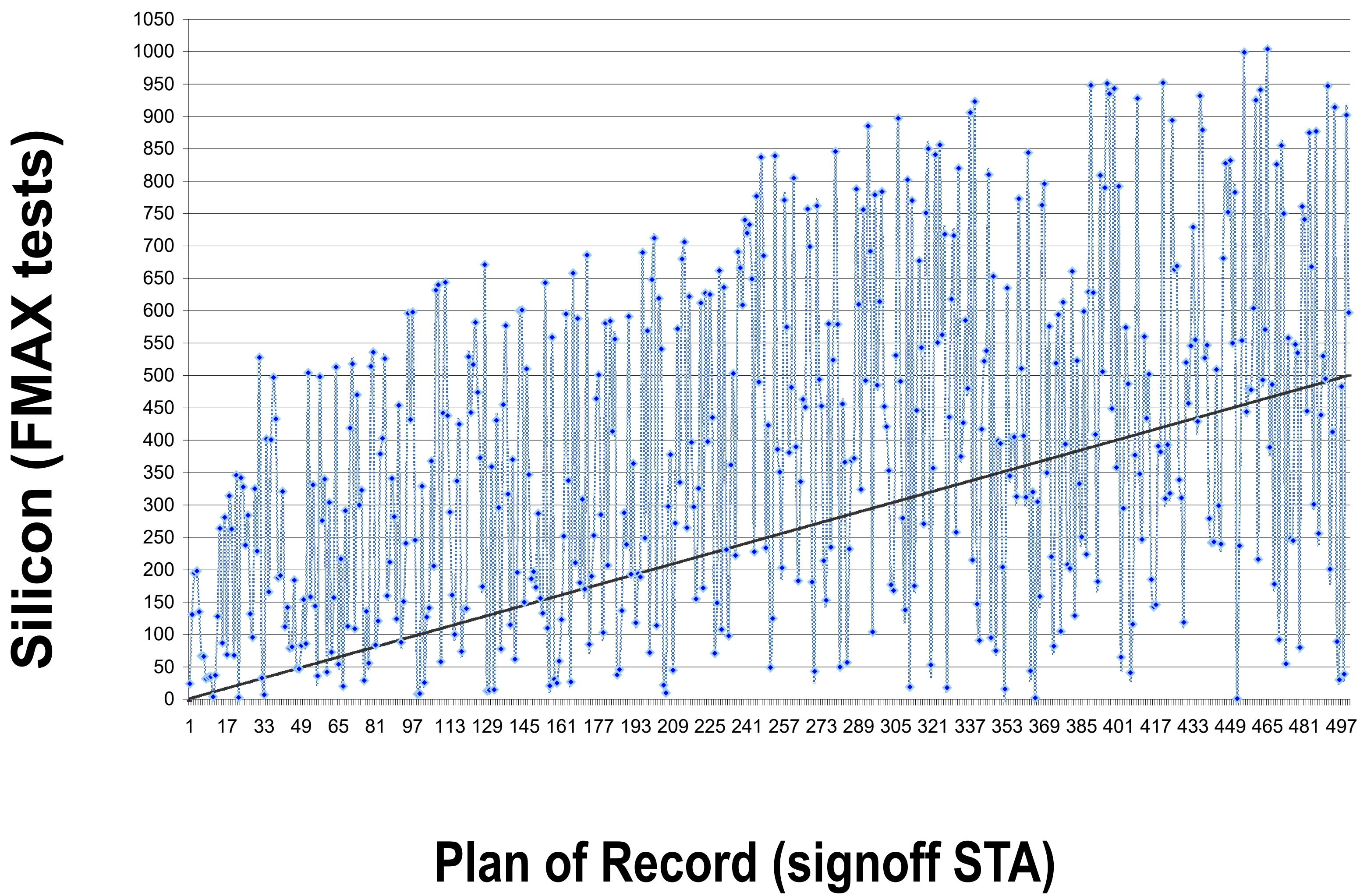
Design	Criticality	Clock (ns)	"S"			"A"			"B"		
			With original Clock			With original Clock			With original Clock		
			Setup			Setup			Setup		
			WNS (SOCE) (ns)	WNS (PT) (ns)	TNS (PT) (ns)	WNS (Astro) (ns)	WNS (PT) (ns)	TNS (PT) (ns)	WNS(BF) (ns)	WNS (PT) (ns)	TNS (PT) (ns)
AES	Tight clock (original 2.2ns)	2.1998	-0.407	-0.430	-81.124	-0.241	-0.487	-94.822	-0.077	-0.391	-60.156
		2.1999	-0.392	-0.420	-73.533	-0.218	-0.512	-89.316	-0.067	-0.397	-58.728
		2.2000	-0.399	-0.457	-85.641	-0.255	-0.569	-100.956	-0.081	-0.331	-59.985
		2.2001	-0.436	-0.439	-82.053	-0.280	-0.535	-110.341	-0.074	-0.442	-61.048
		2.2002	-0.406	-0.441	-82.576	-0.246	-0.490	-92.196	-0.067	-0.384	-51.980
	Loose clock (original 3.0ns)	2.9998	-0.026	-0.119	-1.965	0.040	-0.280	-35.482	0.000	-0.342	-44.778
		2.9999	-0.091	-0.095	-2.137	0.064	-0.325	-34.699	0.001	-0.469	-46.154
		3.0000	-0.046	-0.096	-3.499	0.049	-0.346	-36.565	-0.001	-0.448	-48.369
		3.0001	-0.049	-0.112	-1.972	0.083	-0.239	-23.040	-0.008	-0.373	-44.683
		3.0002	-0.061	-0.078	-1.718	0.057	-0.287	-31.985	0.000	-0.421	-48.042
JPEG	Tight clock (original 1.3ns)	1.2998	-0.294	-0.315	-625.434	-0.265	-0.352	-744.637	-0.228	-0.324	-501.295
		1.2999	-0.263	-0.281	-566.317	-0.240	-0.418	-701.361	-0.166	-0.266	-410.594
		1.3000	-0.257	-0.258	-537.580	-0.256	-0.395	-733.841	-0.244	-0.338	-567.228
		1.3001	-0.249	-0.303	-561.013	-0.239	-0.321	-719.196	-0.202	-0.304	-475.253
		1.3002	-0.298	-0.514	-757.272	-0.229	-0.346	-731.566	-0.197	-0.277	-471.392
	Loose clock (original 2.0ns)	1.9998	-0.005	-0.011	-0.011	0.101	-0.140	-0.520	0.000	-0.216	-11.407
		1.9999	0.008	-0.068	-0.068	0.101	-0.140	-0.520	0.000	-0.167	-12.021
		2.0000	-0.007	-0.093	-0.137	0.101	-0.131	-1.240	-0.002	-0.196	-15.189
		2.0001	-0.001	-0.010	-0.010	0.096	-0.098	-0.449	0.001	-0.181	-16.782
		2.0002	0.008	-0.004	-0.006	0.099	-0.066	-0.279	-0.006	-0.178	-12.220

Q: Why do chips pass timing signoff in design, but then fail to yield in the fab?

A: Signoff criteria are different from manufacturing criteria.

Problem: Path Index Migration/Miscorrelation

Top 500-Ranked Critical Paths At Signoff, vs. Rank In Silicon



No Two Chips are Identical

- Manufacturing variation is the reality.
  - "Corners" (FF, SS, typical) try to approximate them
- Do NOT expect models to be "accurate"
  - Manufacturing process is always a random variable with a distribution
  - Corollary: wasting time on 1ps improvement is useless
  - Corollary: wasting time on 1nm dimension change is useless and often not permitted by design rules

Q: Can I do better than the tool?

A: Very unlikely.

EDA tools are (usually) well optimized

- Experience
  - You: 0-10 designs
  - EDA tool: 1000s of designs over years/decades
- Correctness
  - You: prone to making mistakes
  - EDA tool: any bugs ironed out over experience
- Quality of results
  - You: may be can do a better job with 100 gate designs
  - EDA tool: Optimized algorithms to deal with millions of gates
- Cost and Effort
  - You: a graduate engineer paid costing a company $200K+/year
  - EDA tool: hardware cost: $10K/year for 32 processor server which can churn million gate SP&R overnight + tool cost ($10K-$500K/year amortized over many designers)
- Weird, strange constraints or objectives
  - This is where you *may* have an edge. Tools are optimized to handle the common case and some not so common cases (through a large number of visible and hidden switches)

Q: Can I get away with no "programming" being a designer?

A: Not really, at least if you want to be an effective digital designer.

Managing Complex Designs requires Methodologies → Scripting

- You may not need to write complex C++ code but scriptware is *very* common
  - Timers, SP&R, most EDA tools: TCL is the defacto standard scripting language.
    - Industry-strength tool flows often have 1000s of lines of TCL scripts
  - Running PV (e.g., Calibre): its own SVRF scripting language
  - Managing design databases (OpenAccess, Milkyway, etc) using TCL, Python, SKILL,….
  - Parsing reports, automating tool flows, managing files: Shell scripts, Perl, Python, TCL…
- Opening tool GUIs is more of an exception than norm
  - Its preferred to launch noGUI scripts and wait for runs to complete
  - May be use the GUI or the tool shell to debug
- Unix/Linux is the near-universal standard (Windows/MAC support is minimal): Learn how to use Linux and Linux shell utilities effectively!
- Jobs are launched often on large server farms → learn how to use compute cluster tools (e.g., LSF)

Q: Can I just ask somebody if I get stuck using a tool?

A: Not always, learn to debug yourself!

Debugging yourself

- "Big" companies *may* have internal tool support and external application engineering support which *may* be sufficient
  - But no one appreciates "trivial" questions
- "Small" companies, universities get little tool support
  - Universities get near zero
- Debugging yourself
  - Google!
  - Tools have extensive documentation
    - User guides, reference manuals, man/info pages, application notes
  - Message boards on EDA company websites
  - Resist the urge to post on Piazza (or CAD support team) the first instant you see an error. Most tool errors are informative which help you debug.

Q: How do I search for prior art ?

A: Google Scholar

Literature search 101

- Very few things are truly "new"
- Best (current) way of searching literature: Google Scholar (searches books, papers, patents)
- Think of keywords on the topic and what might be one hop away
  - Remember "*i*" in Math is "*j*" in EE!
  - My TSP is your scan chain ordering!
- Everything in Google Scholar has "cites" and "cited by". This allows you to systematically trace literature.