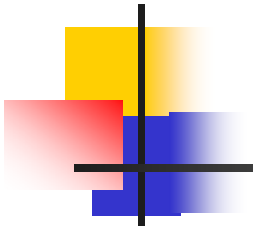
A Novel Performance-Driven Topology Design Algorithm

Min Pan and **Chris Chu**
Iowa State University

Priyadarshan Patra
Intel Corporation

Work partially supported by SRC under task ID 1206
and NSF under grant CCF-0540998

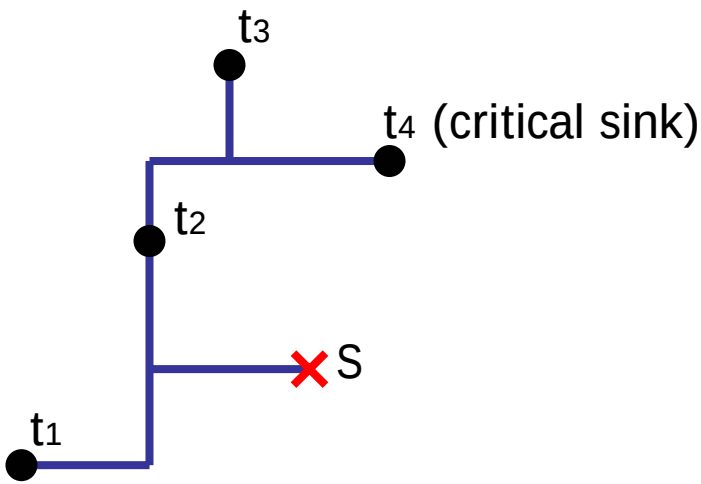


Motivation

- Global nets normally have high fanout and span large areas.
- Global nets are trouble makers
  - Use a lot of interconnect resources (30%+ total wirelength)
  - Major contributors to critical paths
- # Global nets are considerably large (10%+ nets have degree > 8).
- Efficient performance-driven topology design algorithms for these global nets are needed
- Generating good topology is hard and time consuming

Tree Topology vs. Timing

- Topology generation is very hard
  - Solution space is huge
  - Time consuming
  - Not easy to evaluate quality during generation



RSMT may not be a good choice for timing!

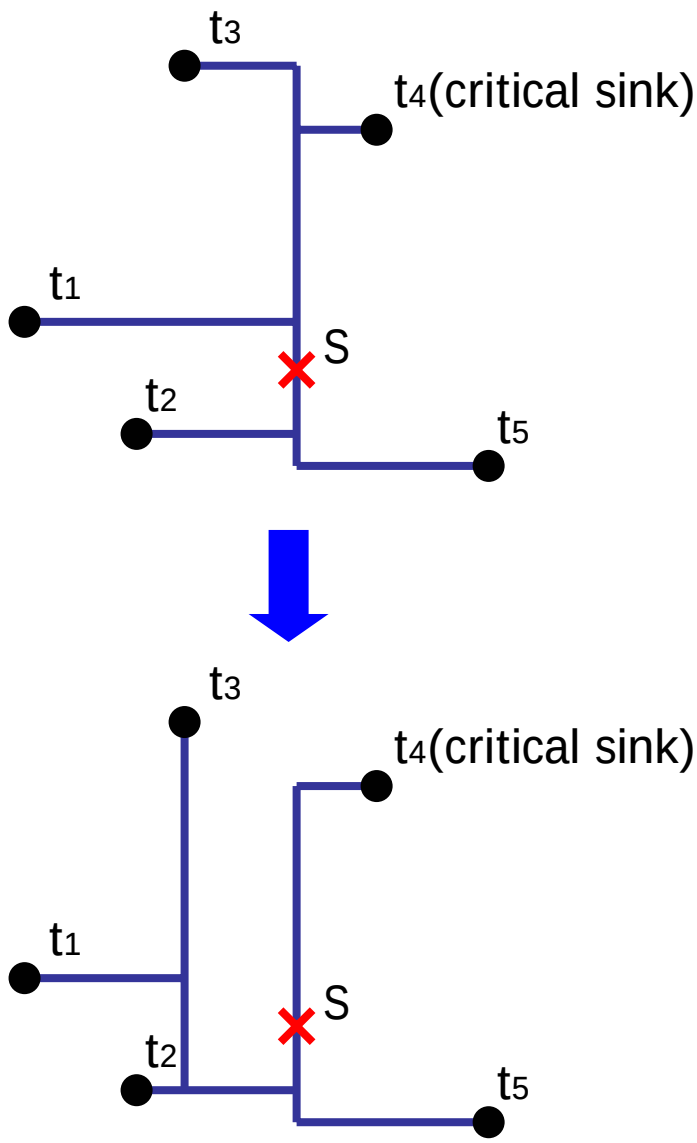
Problem Formulation

- Given:
  - A set of terminals N = {s, t_1, t_2, …, t_n}, s – source, t_i - sinks.
  - Relative required time RT = {rt_1, rt_2, …, rt_n} which are sink required time reference to source s.
  - Load capacitance for each sink CL = {cl_1, cl_2, …, cl_n} and driver resistance rd.
- Objective:
  - Find Steiner routing tree topology optimized for WNS (worst negative slack) or TNS (total negative slack).

Our Approach

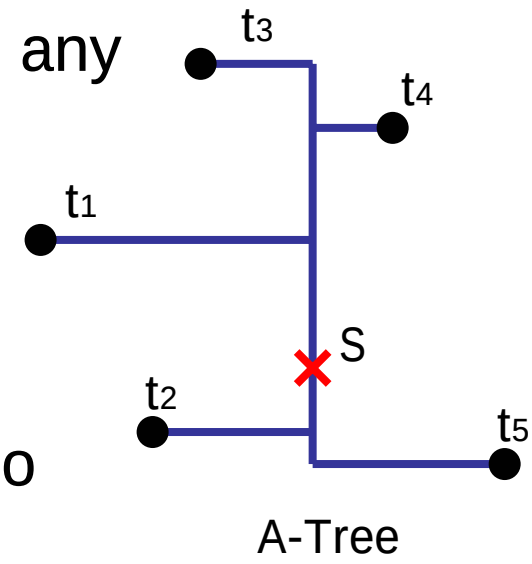
- Step 1: Construct high-quality A-trees very efficiently

- Step 2: Modify obtained A-tree structure to achieve high performance



A-Tree

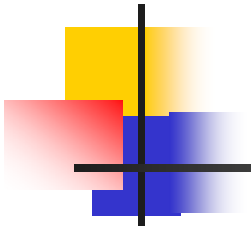
- J. Cong et. al. [DAC93]
  - A rectilinear Steiner tree is called an A-tree if every path connecting its source and any node on the tree is a shortest path.
- A-tree properties
  - Shortest path tree (SPT)
  - Minimizing total tree wirelength leads to simultaneous optimization of different components of sink delay
  - Good starting point for performance-driven routing trees



A-Tree

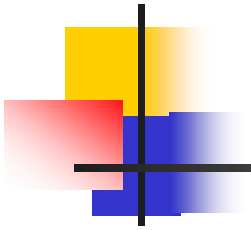
A-Tree Topology Generation

- Finding minimum wirelength A-tree is NP-complete. Want to have an efficient way to generating good A-tree topologies
- FLUTE (Fast LookUp Table Estimation) [ICCAD 04, ISPD 05]
  - An extremely fast and accurate Steiner tree algorithm
  - Table lookup technique
- Inspired by FLUTE, we propose a table lookup idea to generate A-tree topologies



A-Tree Construction

- A-Tree Lookup Table Generation
  - Table Structure
  - Boundary Compaction
  - Configuration Graph for pruning solutions
  - Abstract Topology
  - Topology Signature
- A-Tree Topology Construction

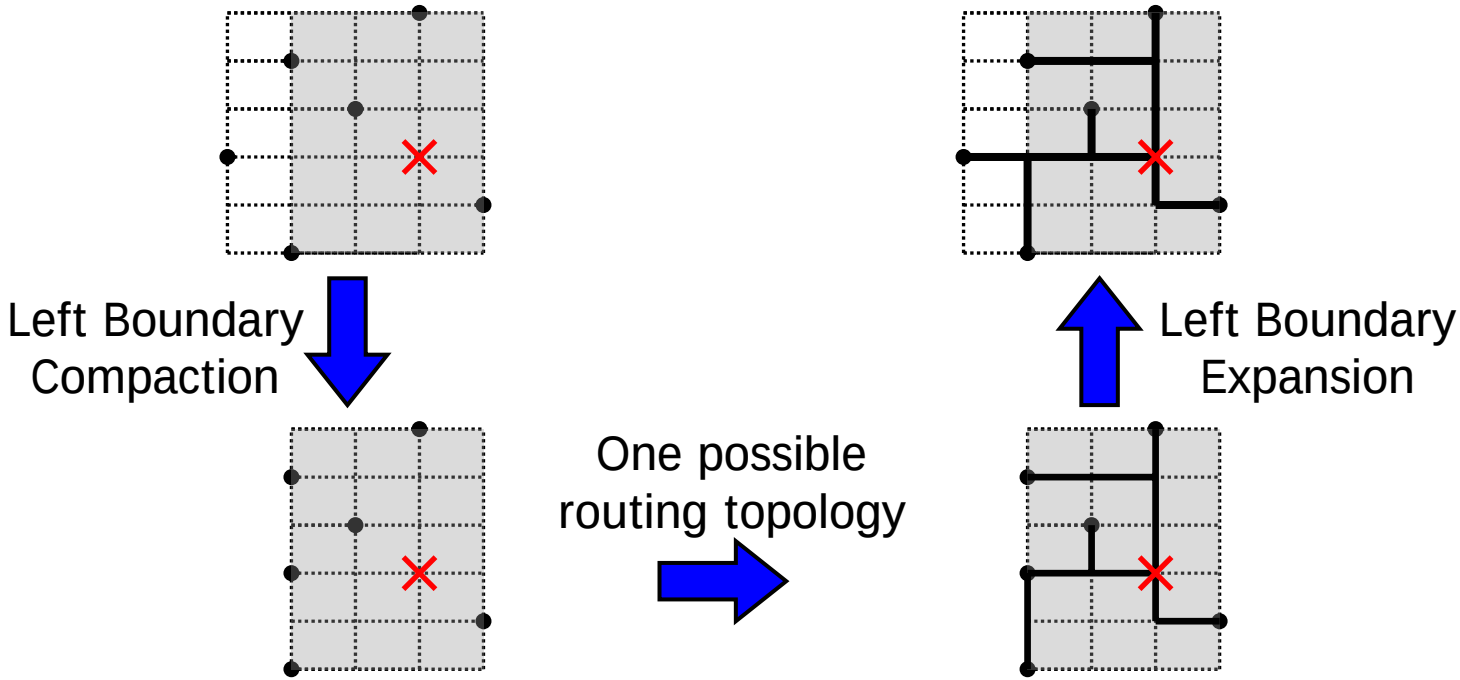


A-Tree Table Structure

- Up to degree D (nets with degree>D will be broken down until table lookup can be performed)
- Group by pin configuration and source location
  - FLUTE - only pin configuration matters
  - A-tree - both pin configuration and source location matter
- # Topologies stored for each group
  - FLUTE - only one topology for each POWV
  - A-tree – all topologies potentially give the minimum wirelength, give more flexibility for later performance-driven trees

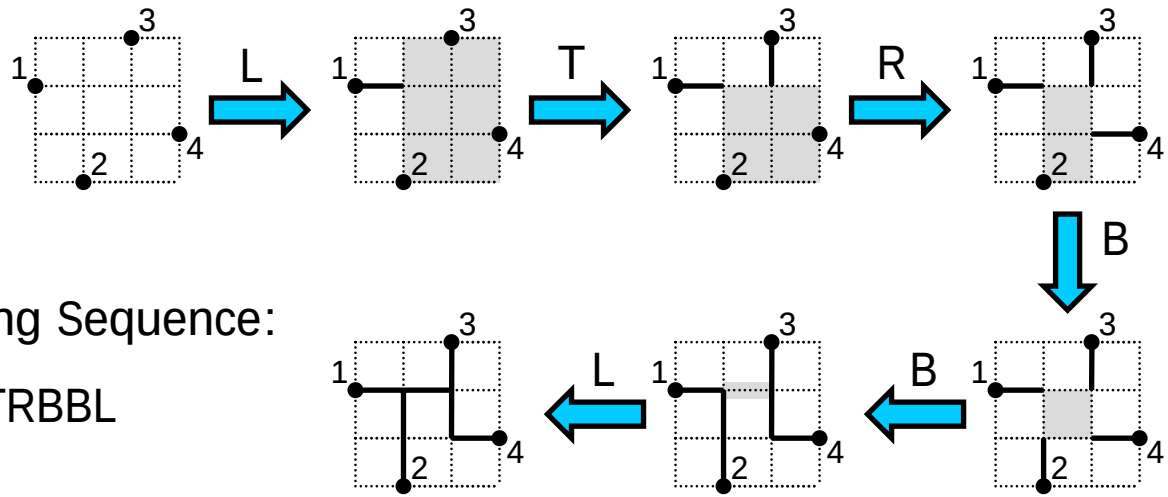
Boundary Compaction

- Observation 1: Most A-trees can be generated by boundary compaction
- Observation 2: A compacting sequence (e.g. LRTLB) corresponds to an A-tree topology



Compacting Sequence

- Compacting Sequence is a sequence of boundaries to perform boundary compaction.
- Given a compacting sequence – based on which compacting the grid into one point at the source's location, we can uniquely generate an A-tree topology.



Compacting Sequence:

LTRBBL

Configuration Graph

- Each compacting sequence gives a unique A-Tree topology
- Unfortunately the number of compacting sequences is huge and hence cannot be stored and evaluated
  - eg: For a d-pin net - # compacting sequences =

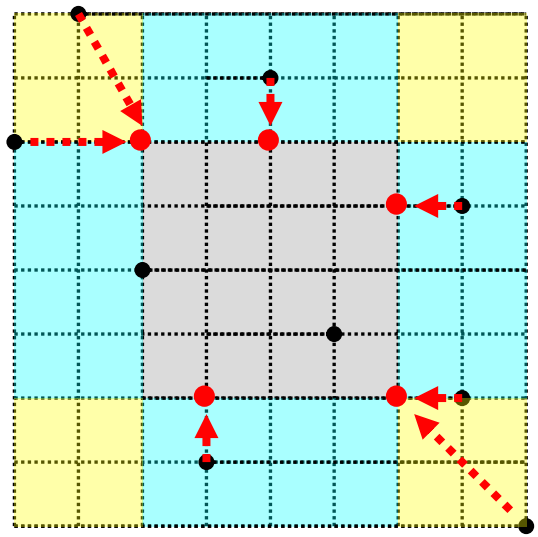
$$\binom{2(d-1)}{(d-1)} \times 2^{d-1} \times 2^{d-1} \times d!$$

- But we need to store only topologies that result in the best wirelength
- Hence, we can prune the compacting sequences

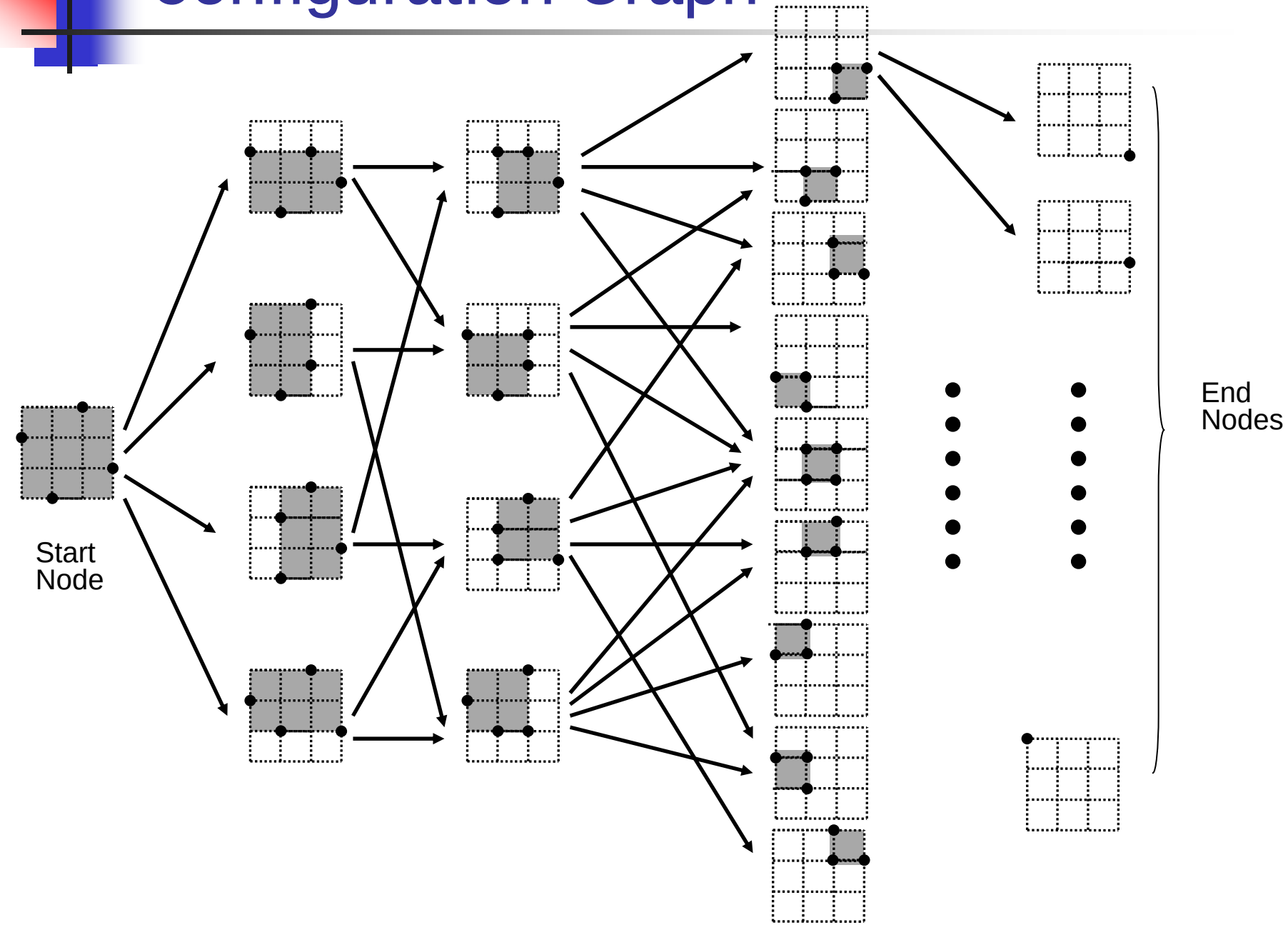
Our idea is to simultaneously prune the compacting sequences during generation – use the configuration graph for the same

Pin Configuration

- Pin Configuration (PC) - relative positions on the Hanan grid for the pins in a net
- Applying boundary compaction on a PC results in a new PC
- Lemma: The bounding box of a PC in the original Hanan grid defines the PC.

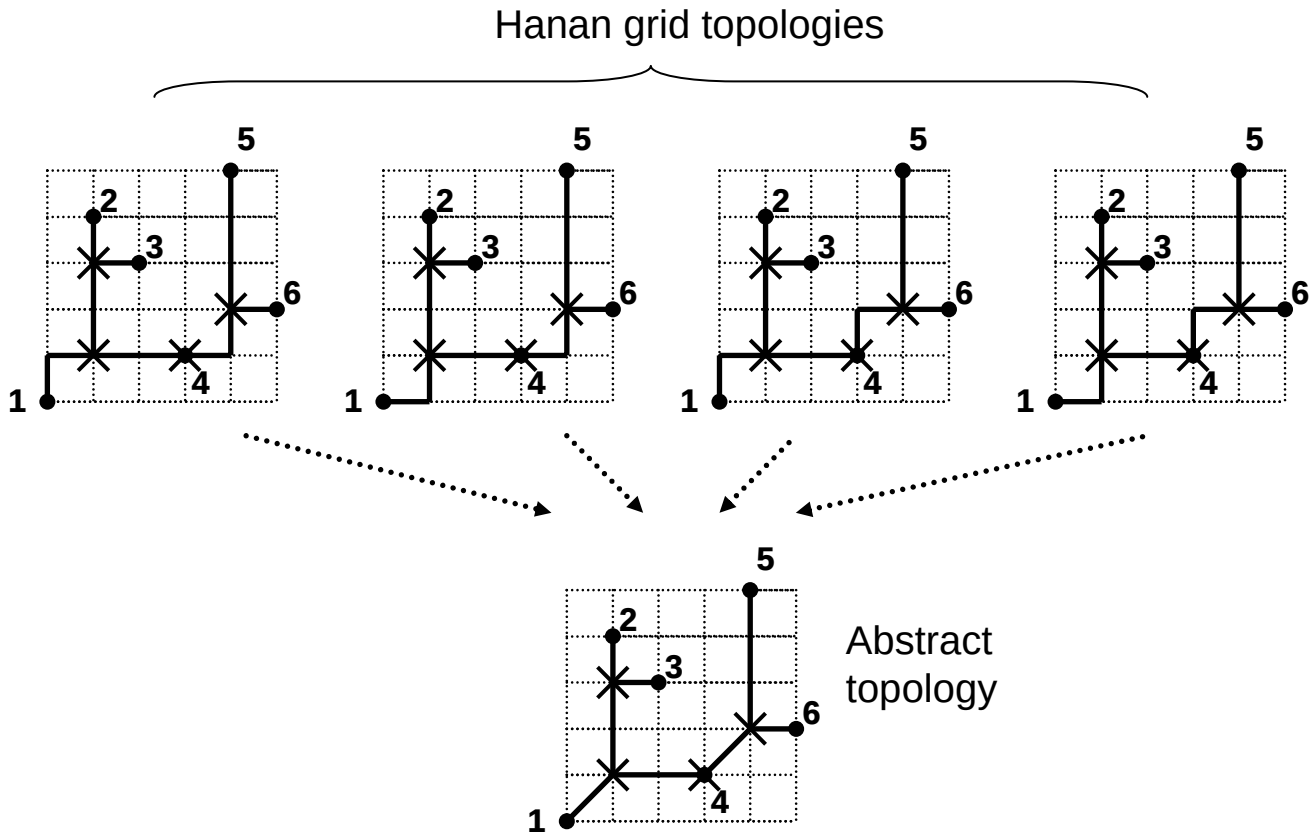


Configuration Graph



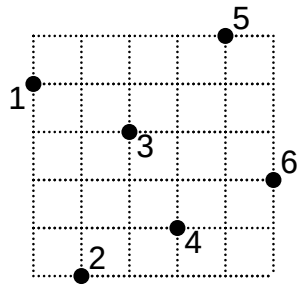
Abstract Topology

- Too many topologies to be stored in the table
- A lot of redundancy among the topologies
- Abstract Topology – fix the positions of all the nodes (pins and Steiner nodes) and the connections between nodes

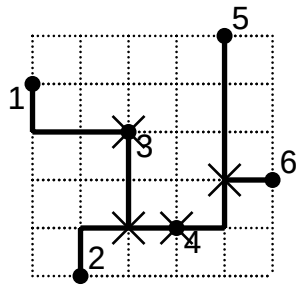


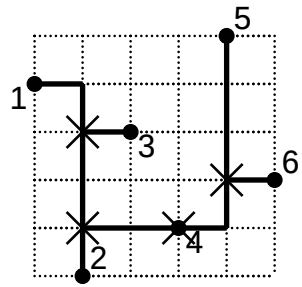
Topology Signature (1)

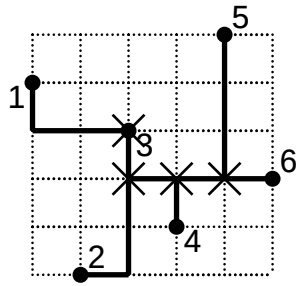
- A signature defines an Abstract Topology for a given pin configuration
- We find that the Steiner positions in a tree defines the Abstract Topology
- Make the redundancy checking much easier and faster

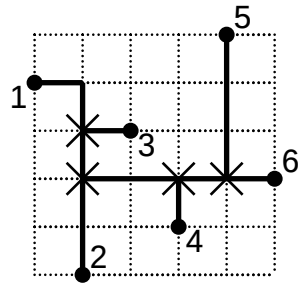


POWV:
(1,2,1,1,1;1,2,2,2,1)









Topology Signature (2)

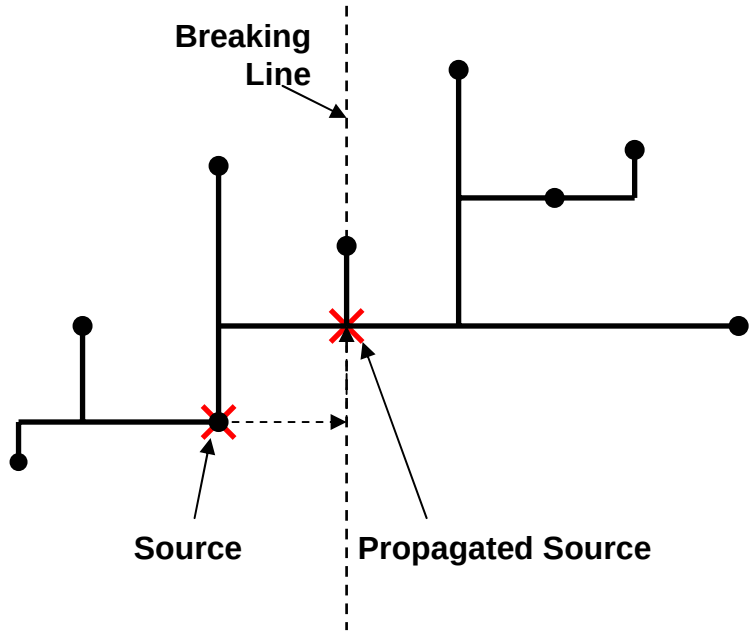
- All 9-pin nets:
  - Total # compacting sequences > 1.37 trillion
  - Total # Topology Signatures = 1087157, more than a MILLION times less!

- Statistics of Topology Signatures:

Degree (d)	# groups (d!)	Max # signatures/POWV	Total # signatures
4	24	2	5
5	120	3	41
6	720	4	354
7	5040	5	3938
8	40320	6	59652
9	362880	7	1087157

A-tree construction and Net-breaking

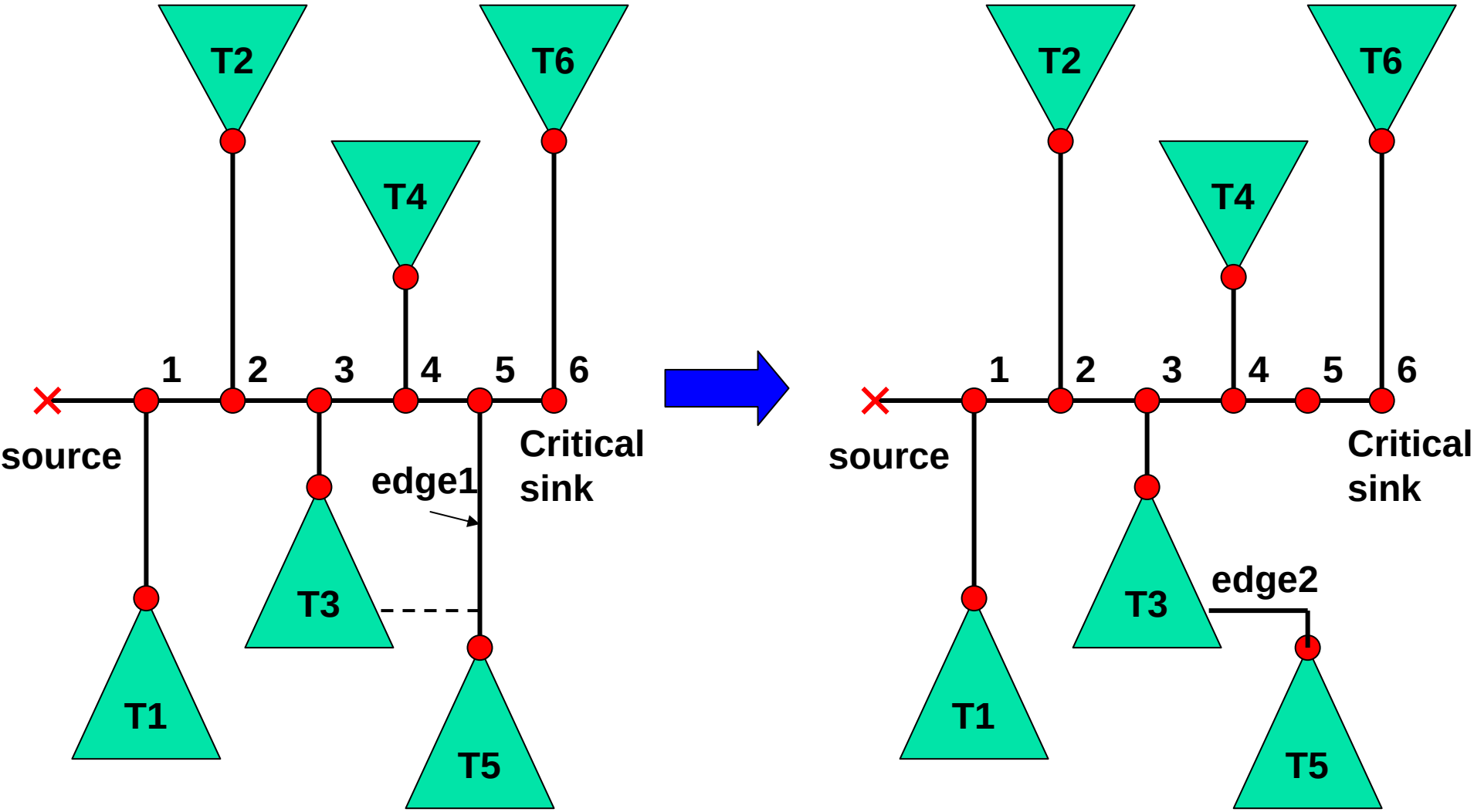
- Only the nets with degree < D can obtain their topologies directly from lookup table
- High degree nets need to be broken down until the table lookup can be applied
- When breaking the net, need to propagate the source so that each subtree has its own source for A-tree generation



Performance-driven Post-processing

- Based on obtained A-tree structure
- Not stick to A-tree any more
- Improve timing measurement (WNS or TNS)
- Branch moving heuristic
  - Effective in reducing WNS (TNS)
  - Very efficient

Branch Moving



Experimental Results (1)

- 2 sets of critical nets extracted from two industry designs
  - 12 nets (design at 65nm technology node)
  - 17 nets (design at projected 45nm technology node)
- Run on a 750MHz Sun Sparc-2 machine
- Average is over all the 29 testcases

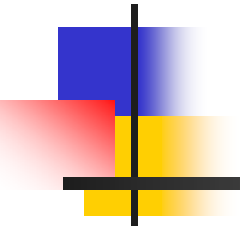
test case	deg	Tree Wirelength			WNS (ps)				TNS (ps)				Runtime (s)		
		Our	Ctree	FLUTE	Our		Ctree	FLUTE	Our		Ctree	FLUTE	Our	Ctree	FLUTE
					A-tree	Final			A-tree	Final					
t1	9	1	1.029	0.914	-0.97	-0.80	-0.97	-0.87	-0.97	-0.80	-0.97	-0.87	1	111	0.11
t2	38	1	1.112	0.936	-5.66	-5.40	-5.71	-5.55	-5.66	-5.40	-5.71	-5.55	1	191	0.57
t3	58	1	1.176	0.809	0.00	0.00	-1.98	-21.61	0.00	0.00	-1.98	-144.3	1	704	1.15
t4	21	1	0.983	0.793	-16.32	-14.33	-15.62	-20.72	-32.34	-28.52	-31.10	-41.03	1	286	0.48
t5	9	1	1.032	0.968	-4.10	-3.81	-3.91	-4.20	-7.95	-7.31	-7.52	-8.07	1	250	0.13
t6	51	1	1.145	0.782	-1.82	0.00	-2.14	-9.76	-1.82	0.00	-2.14	-26.41	1	1255	0.89
Avg	**28**	**1**	**1.095**	**0.915**	**-7.38**	**-6.09**	**-7.41**	**-10.55**	**-21.96**	**-18.76**	**-22.87**	**-75.27**	**1**	**371**	**0.487**

Experimental Results (2)

test case	deg	Tree Wirelength			WNS (ps)				TNS (ps)				Runtime (s)		
		Our	Ctree	FLUTE	Our		Ctree	FLUTE	Our		Ctree	FLUTE	Our	Ctree	FLUTE
					A-tree	Final			A-tree	Final					
n_1885	27	1	1.077	0.860	-4.56	-1.51	-3.73	-6.19	-4.56	-1.51	-3.73	-6.19	1	346	0.73
n_1898	39	1	1.052	0.907	-4.91	-2.73	-4.75	-8.85	-4.91	-2.73	-4.75	-8.85	1	304	0.87
n_2045	54	1	1.181	0.897	-22.71	-22.71	-25.29	-23.28	-126.0	-126.0	-155.4	-147.3	1	455	0.75
n_2049	45	1	1.158	0.924	-2.95	-0.62	-3.55	-5.43	-2.95	-0.62	-5.27	-7.97	1	468	0.84
n_2071	29	1	1.079	0.890	-12.99	-10.66	-14.51	-14.38	-12.99	-10.66	-14.51	-14.38	1	375	0.56
n_2072	69	1	1.180	0.845	-14.72	-12.09	-22.98	-61.55	-48.39	-37.92	-96.73	-1420	1	385	0.74
Avg	**28**	**1**	**1.095**	**0.915**	**-7.38**	**-6.09**	**-7.41**	**-10.55**	**-21.96**	**-18.76**	**-22.87**	**-75.27**	**1**	**371**	**0.487**

Conclusion and Future Direction

- New topology design algorithm
  - An efficient lookup-table based A-tree algorithm
  - A post-processing technique further improve performance
  - Achieve high-quality and fast runtime
- Future direction
  - Include buffer insertion and sizing
  - Include wire sizing



Thank You !

Questions?