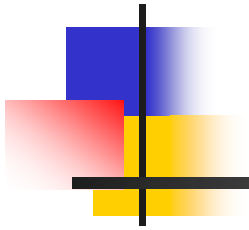


FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm

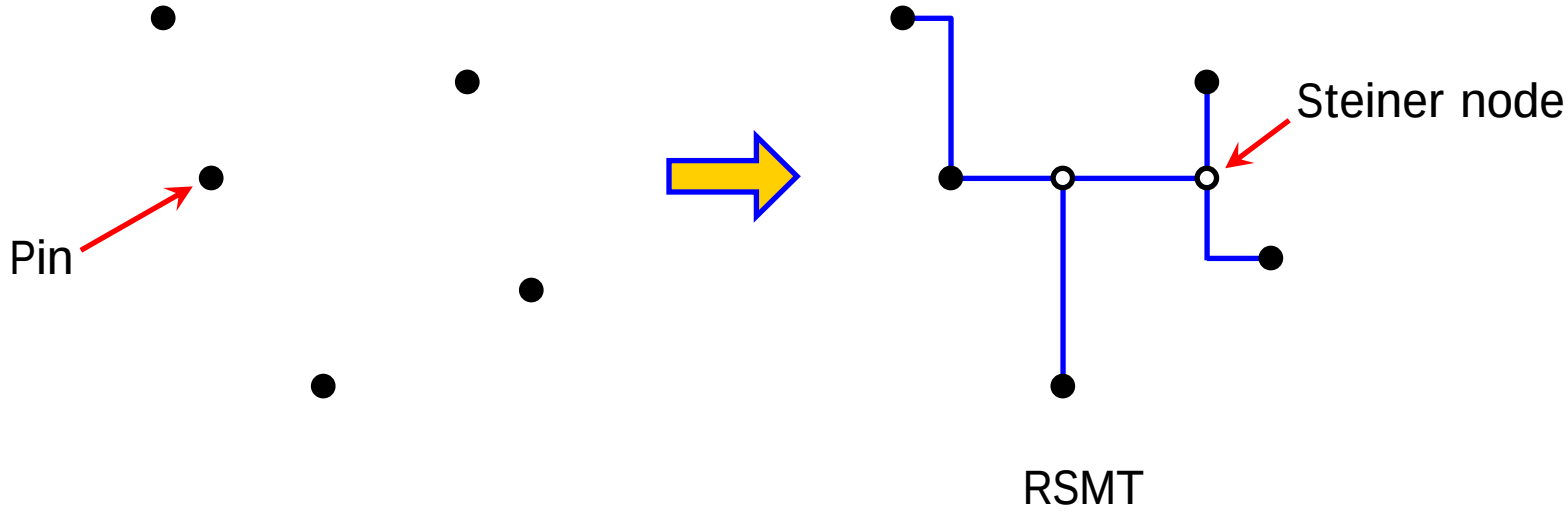
Chris Chu

Department of Electrical and Computer Engineering

Iowa State University

RSMT Problem

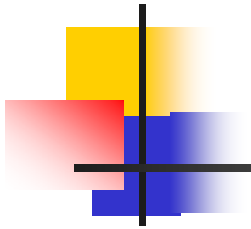
- Rectilinear Steiner minimal tree (RSMT) problem:
  - Given pin positions, find a rectilinear Steiner tree with minimum wirelength (WL)



  - Very useful in routing of VLSI circuits
  - NP-complete

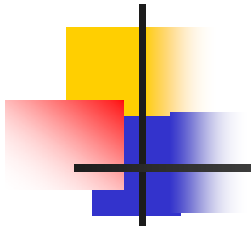
Previous RSMT Algorithms

- Optimal algorithms:
  - Hwang, Richards, Winter [ADM 92]
  - Warme, Winter, Zachariasen [AST 00] *GeoSteiner package*
- Near-optimal algorithms:
  - Griffith et al. [TCAD 94] *Batched 1-Steiner heuristic (BI1S)*
  - Mandoiu, Vazirani, Ganley [ICCAD 99]
- Low-complexity algorithms:
  - Borah, Owens, Irwin [TCAD 94] *Edge-based heuristic, O(n log n)*
  - Kahng et al. [ASPDAC 03] *Batch Greedy Algorithm, O(n $\log^2$ n)*
  - Zhou [ISPD 03] *Spanning graph based, O(n log n)*
- Algorithms targeting low-degree nets (VLSI applications):
  - Soukup [Proc. IEEE 81] *Single Trunk Steiner Tree (STST)*
  - Chen et al. [SLIP 02] *Refined Single Trunk Tree (RST-T)*



FLUTE Overview

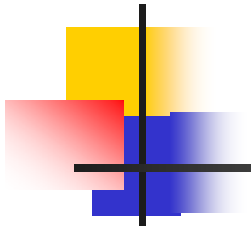
- FLUTE -- Fast LookUp Table Estimation
- Basic idea:
  - LUT to handle nets with a few pins
  - Net breaking technique to recursively break large nets
- Low degree nets are handled extremely well:
  - Optimal and extremely efficient for nets up to 9 pins
  - Still very accurate and fast for nets up to 100 pins
- So FLUTE is especially suitable for VLSI applications:
  - Over all 1.57 million nets in 18 IBM circuits [ISPD 98]
    - More accurate than Batched 1-Steiner heuristic
    - Almost as fast as minimum spanning tree construction



Practical Impact of FLUTE

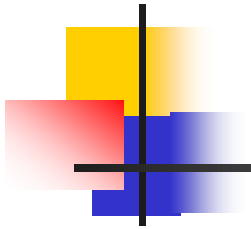
- Nine companies either have or are planning to incorporate FLUTE into their tools:
  - Intel, IBM, Magma, Calypto Design Systems, Atoptech, Dorado Design Automation, Lightspeed Semiconductor Corporation, Lizotech, Pulsic Limited

- Thirteen academic EDA tools have incorporated FLUTE:
  - Physical synthesis tools: SafeResynth
  - Placement tools: Rooster, IPR
  - Global routing tools: BoxRouter, FastRoute, DpRouter, FGR, Maizerouter, Archer, NTHU-Route, IGOR, HSR, Simple Router



FLUTE Publications

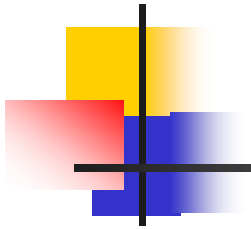
- Chris Chu, “FLUTE: Fast Lookup Table Based Wirelength Estimation Technique”, ICCAD 2004. (FLUTE 1.0)
- Chris Chu and Y.-C. Wong, “Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”, ISPD 2005. (FLUTE 2.0)
- Chris Chu and Y.-C. Wong, “FLUTE: Fast Lookup Table Based Recti-linear Steiner Minimal Tree Algorithm for VLSI Design”, TCAD 2008. (FLUTE 2.5)
- Yiu-Chung Wong and Chris Chu, “A Scalable and Accurate Rectilinear Steiner Minimal Tree Algorithm”, VLSI-DAT 2008. (FLUTE 3.0)



Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets

- Experimental results



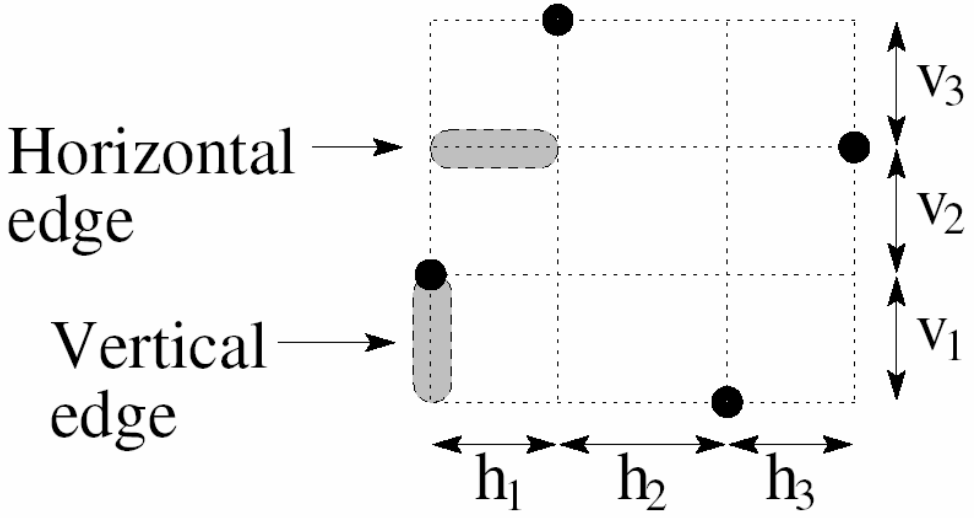
Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets

- Experimental results

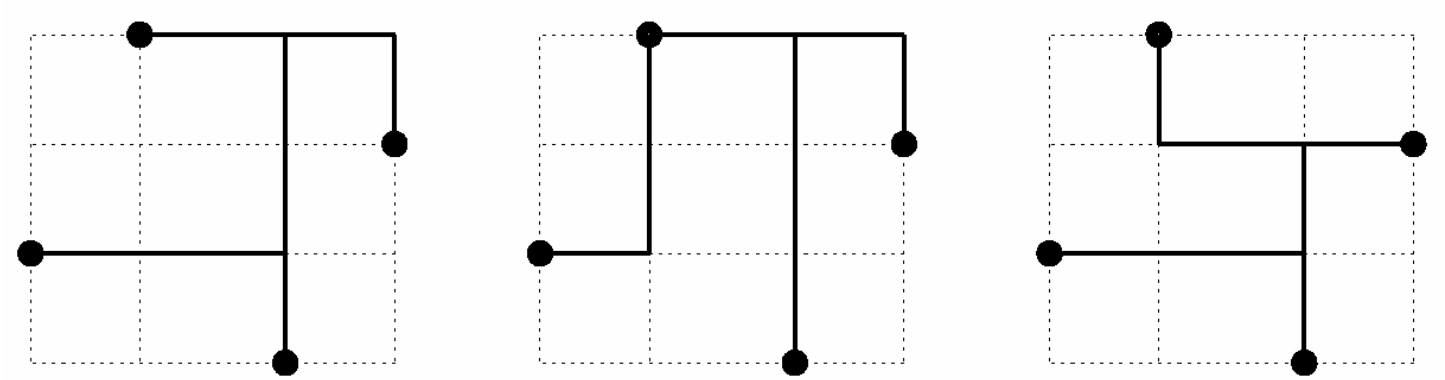
Preliminary

- A net is a set of n pins
- Degree of a net is the number of pins in it
- Consider routing along Hanan grid
  - Hanan proved an optimal RSMT always exists along the Hanan grid
- Observation: An optimal RSMT can always be broken down into a set of horizontal edges and vertical edges
- Define edge lengths h_i and v_i:



Wirelength Vector (WV)

- Observation: WL can be written as a linear combination of edge lengths with positive integral coefficients



$WL = h_1 + 2h_2 + h_3 + v_1 + v_2 + 2v_3$

$WL = h_1 + h_2 + h_3 + v_1 + 2v_2 + 3v_3$

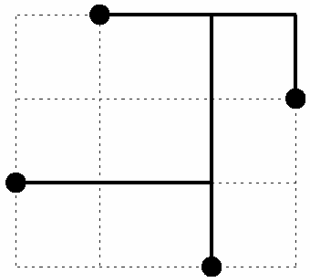
$WL = h_1 + 2h_2 + h_3 + v_1 + v_2 + v_3$

(1, 2, 1, 1, 1, 2) (1, 1, 1, 1, 2, 3) (1, 2, 1, 1, 1, 1)

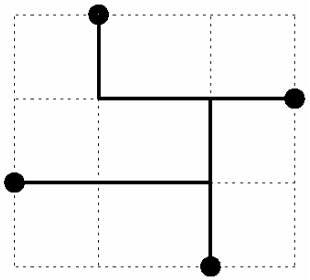
- WL can be expressed as a vector of the coefficients
- Called Wirelength Vector

Potentially Optimal WV (POWV)

- To find optimal wirelength, can enumerate all WVs
- However, most WVs can never produce optimal WL
  - (1, 2, 1, 1, 1, 2) is redundant as it always produces a larger WL than (1, 2, 1, 1, 1, 1)



(1, 2, 1, 1, 1, 2)



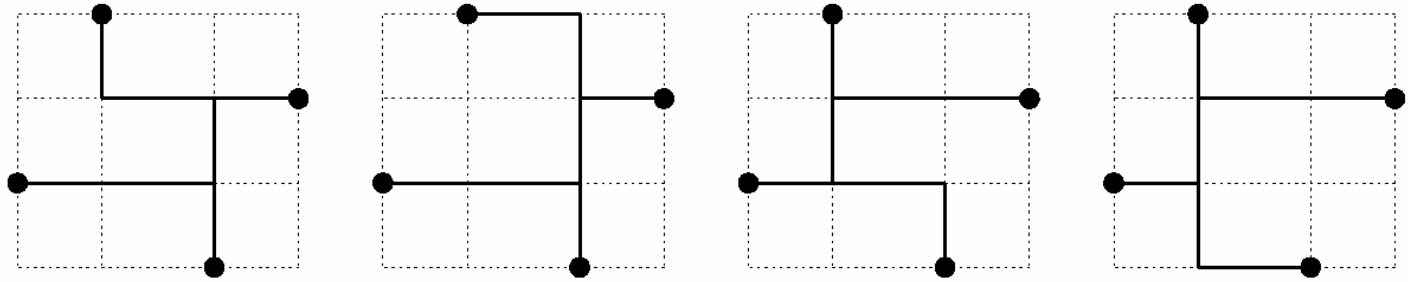
(1, 2, 1, 1, 1, 1)

- Potentially Optimal Wirelength Vector (POWV) is a WV that *may* produce the optimal wirelength

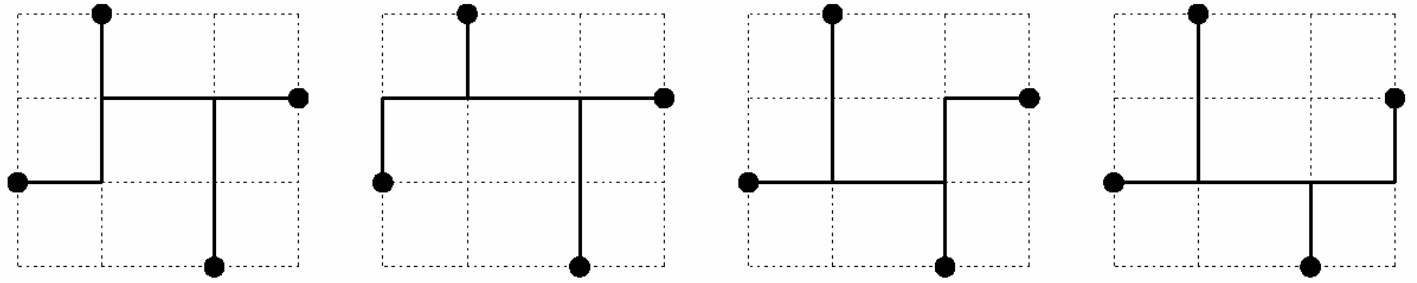
of POWVs is Very Small

- For any net,
  - # of possible routing solutions is huge
  - # of WVs is much less
  - # of POWVs is very small
- For example, only 2 POWVs for the net below:

POWV (1,2,1,1,1,1)



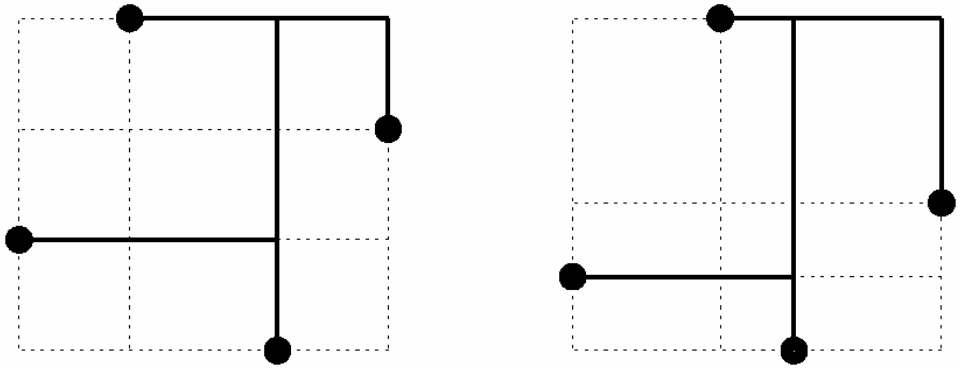
POWV (1,1,1,1,2,1)



Sharing of POWVs Among Nets

- To find optimal WL, we can pre-compute all POWVs and store them in a lookup table
- However, there are infinite number of different nets

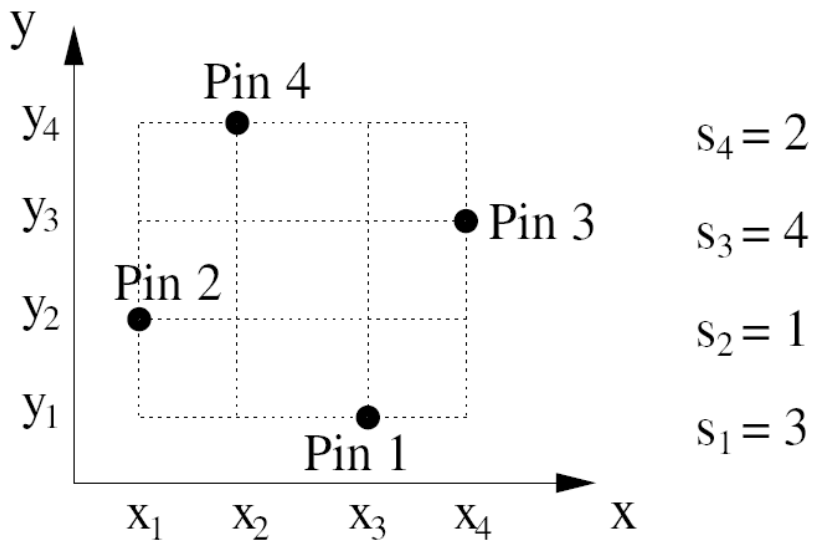
- We try to group together nets that can share the same set of POWVs
- For example, these two nets share the same set of POWVs:



Grouping by Position Sequence

- Define position sequence $s_1 s_2 \ldots s_n$ to be the list of rank of pins in x-coordinate

Position sequence
= 3142

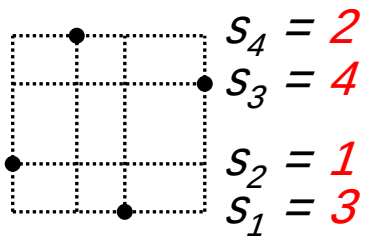


- Lemma: The set of all degree-n nets can be divided into n! groups according to the position sequence such that all nets in each group share the same set of POWVs

Steps of FLUTE for WL Estimation

- Given a net:

1. Find the position sequence

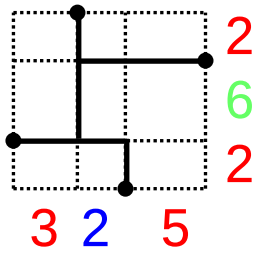


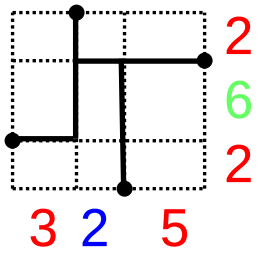
Position sequence: *3142*

2. Get the POWVs from LUT

POWVs:
(1,2,1,1,1,1) (1,1,1,1,2,1)

3. Find the edge lengths

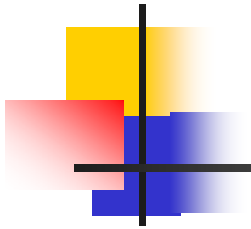




4. Find WL for each POWV and return the best

HPWL + 2 = 22 HPWL + 6 = 26

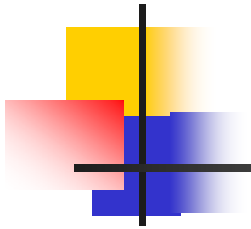
Return



Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets

- Experimental results

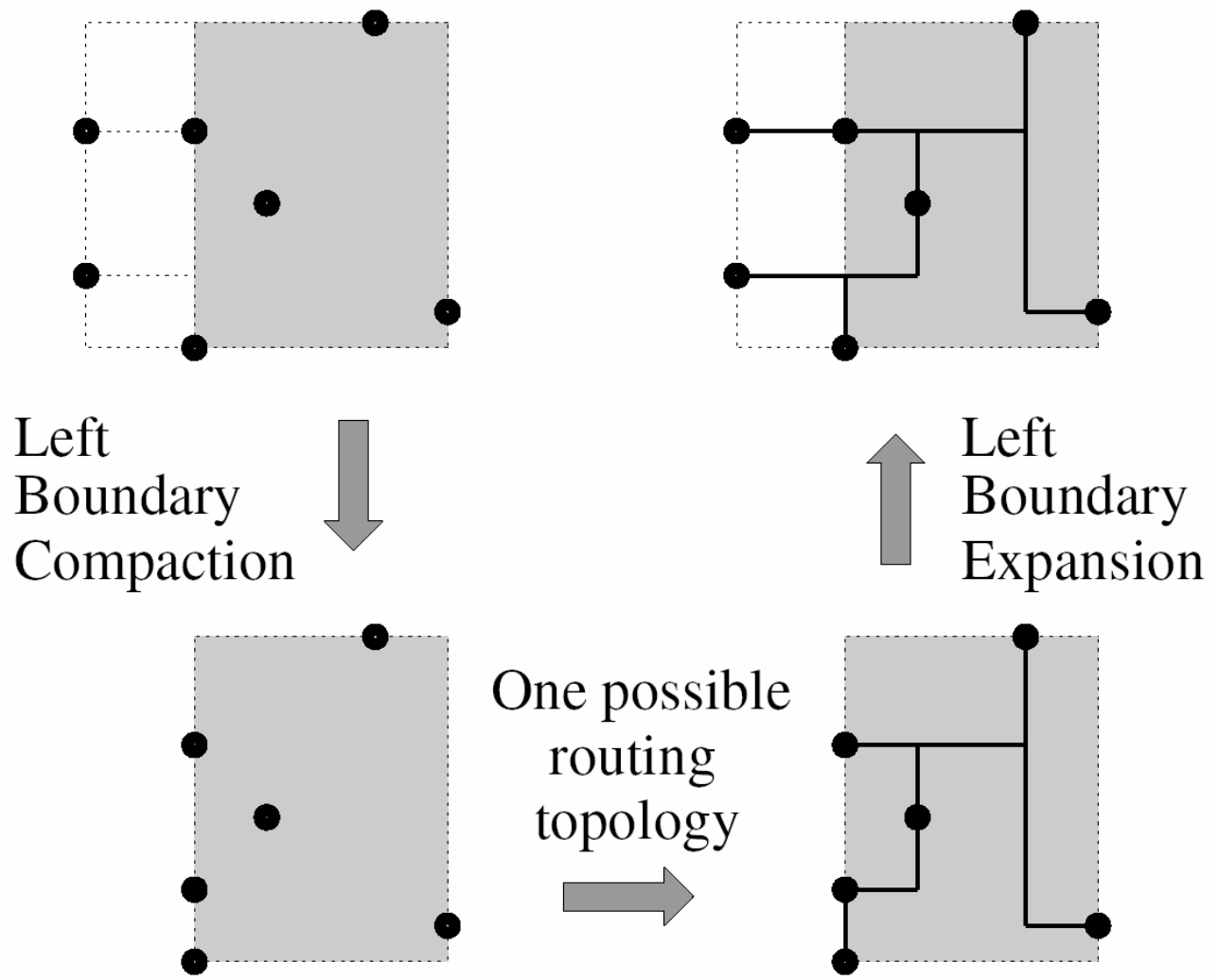


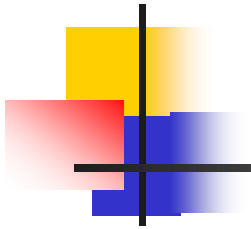
POWVs Generation -- First Attempt

- For each small net degree and for each group (i.e., position sequence),
  - generate all routing topologies
  - find the corresponding WVs
  - prune away the redundant ones.
- Extremely expensive

- A better algorithm based on boundary compaction

Boundary Compaction Technique





POWVs Generation by Boundary Compaction

- Boundary compaction can be considered as a specific way to perform routing
- Different order in compacting the 4 boundaries will generate different routing topologies
- Some routing topologies and hence some WVs may be missed

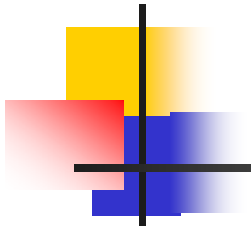
- Theorem: Boundary compaction can enumerate all POWVs for nets up to degree 6

- By including some extra topologies, we can enumerate all POWVs up to degree 9

Statistics on POWV Table

- Table size for all nets up to degree 9 is 2.75MB

Degree n	# of groups $n!$	# of POWVs in a group		
		Min.	Ave.	Max.
2	2	1	1	1
3	6	1	1	1
4	24	1	1.667	2
5	120	1	2.467	3
6	720	1	4.433	8
7	5040	1	7.932	15
8	40320	1	15.251	33
9	362880	1	30.039	79



Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets

- Experimental results

Minimum Wirelength Computation

- For a given net, we need to compute the WL corresponding to a set of POWVs
- Can compute each POWV independently
- However, most POWVs in a group are similar to one another
- Can speed up computation by exploring dependency among POWVs
- Example:

WL_1:(1,2,1,1, 1,2,3,1) WL_2:(1,2,2,1, 1,2,2,1) WL_3:(1,3,2,1, 1,1,2,1)

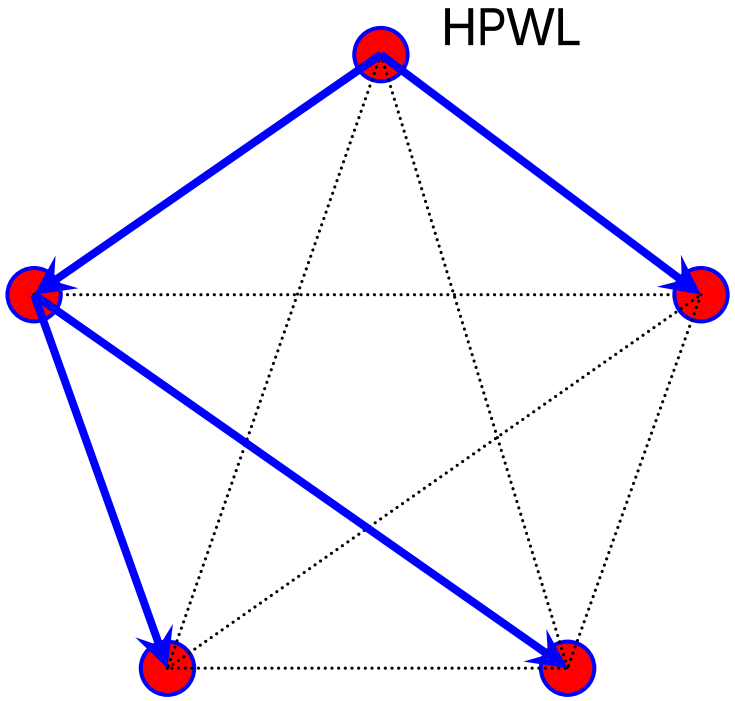
$WL_1 = HPWL + h_2 + v_2 + v_3 + v_3$

$WL_2 = WL_1 + h_3 - v_3$

$WL_3 = WL_2 + h_2 - v_2$

MST-Based Approach

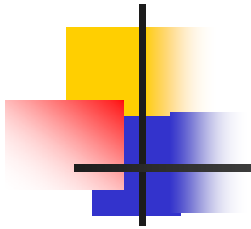
- The WL computation problem of a set of POWVs can be transformed into a MST problem
- Cost of MST = # of Add/Sub to compute all POWVs



- One node for each POWV
- Edge weight
  = # of add/sub required to convert the WL of one WV to the other

of Add/Sub for MST-Based Approach

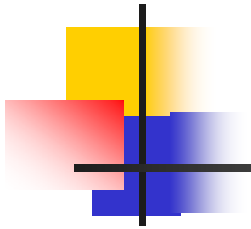
Degree n	Average # of ADD/SUB			
	per group		per POWV	
	Independent	MST	Independent	MST
2	0	0	0	0
3	0	0	0	0
4	1.333	1.333	0.8	0.8
5	4.267	4.267	1.73	1.73
6	14.422	10.333	3.253	2.331
7	39.651	20.025	4.999	2.525
8	109.136	38.561	7.156	2.528
9	288.060	74.155	9.590	2.469



Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets

- Experimental results



High-Degree Nets by Net Breaking

- Lookup table is practical only for low-degree nets
- Have a user-defined parameter D
  - D=9 in our implementation

- For nets up to degree D, use lookup table
- For nets with degree > D, recursively break net until degree <= D
  - Optimal net breaking algorithm
  - Net Breaking Heuristic #1
  - Net Breaking Heuristic #2
  - Net Breaking Heuristic #3
  - Net Breaking Heuristic #4
  - Accuracy control scheme

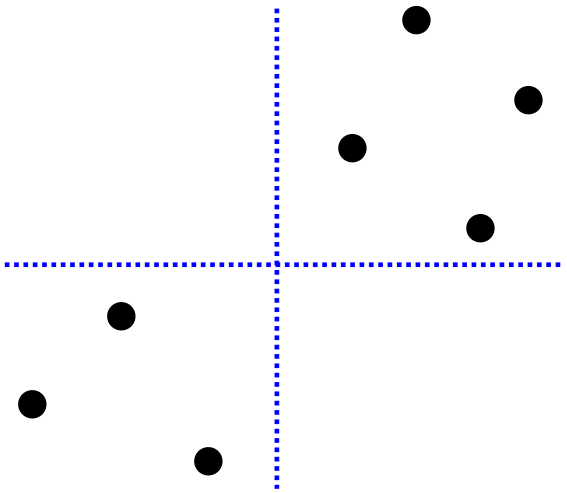
Optimal Net Breaking Algorithm

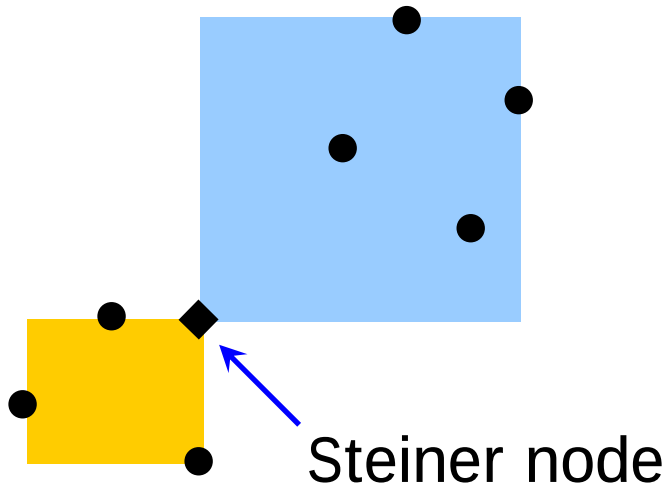
Condition:

Pins on opposite quadrants.

Theorem:

By combining the two optimal sub-trees, the Steiner tree constructed is optimal.

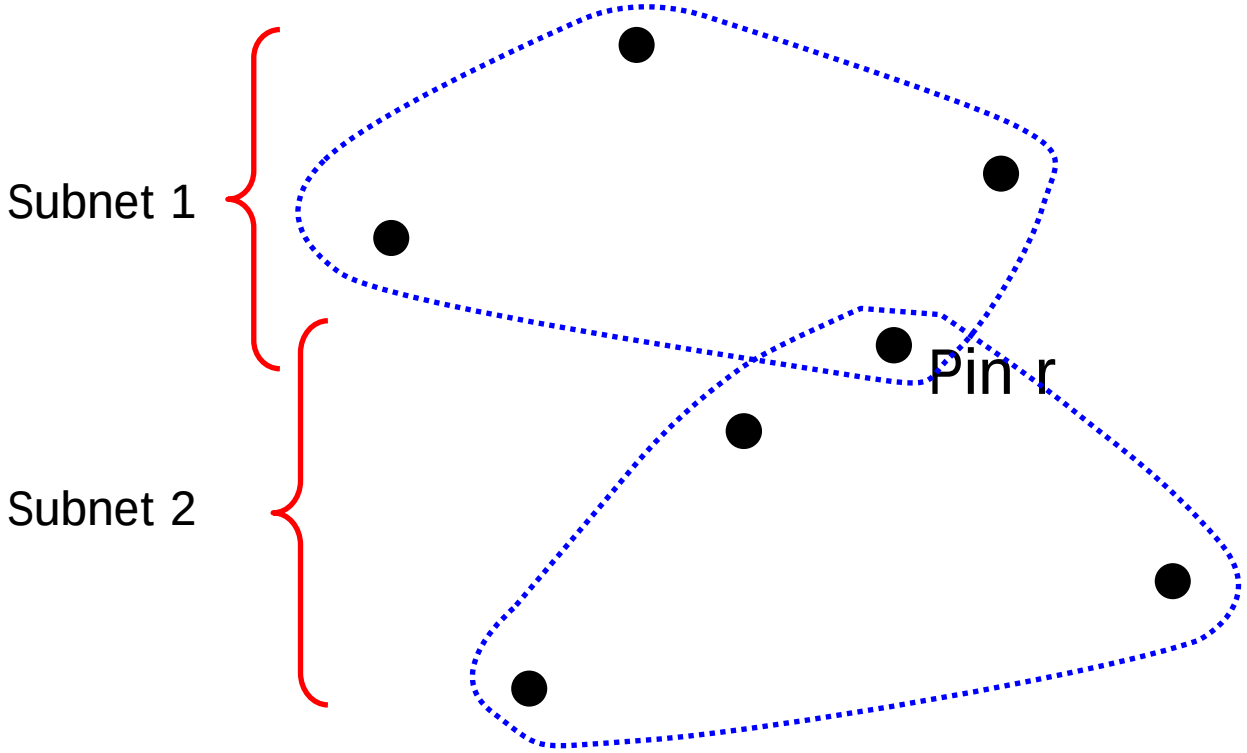




Net Breaking Heuristics

- Apply if condition for optimal net breaking is not satisfied
- A score for each direction and each pin
- Break in several ways which give the highest scores

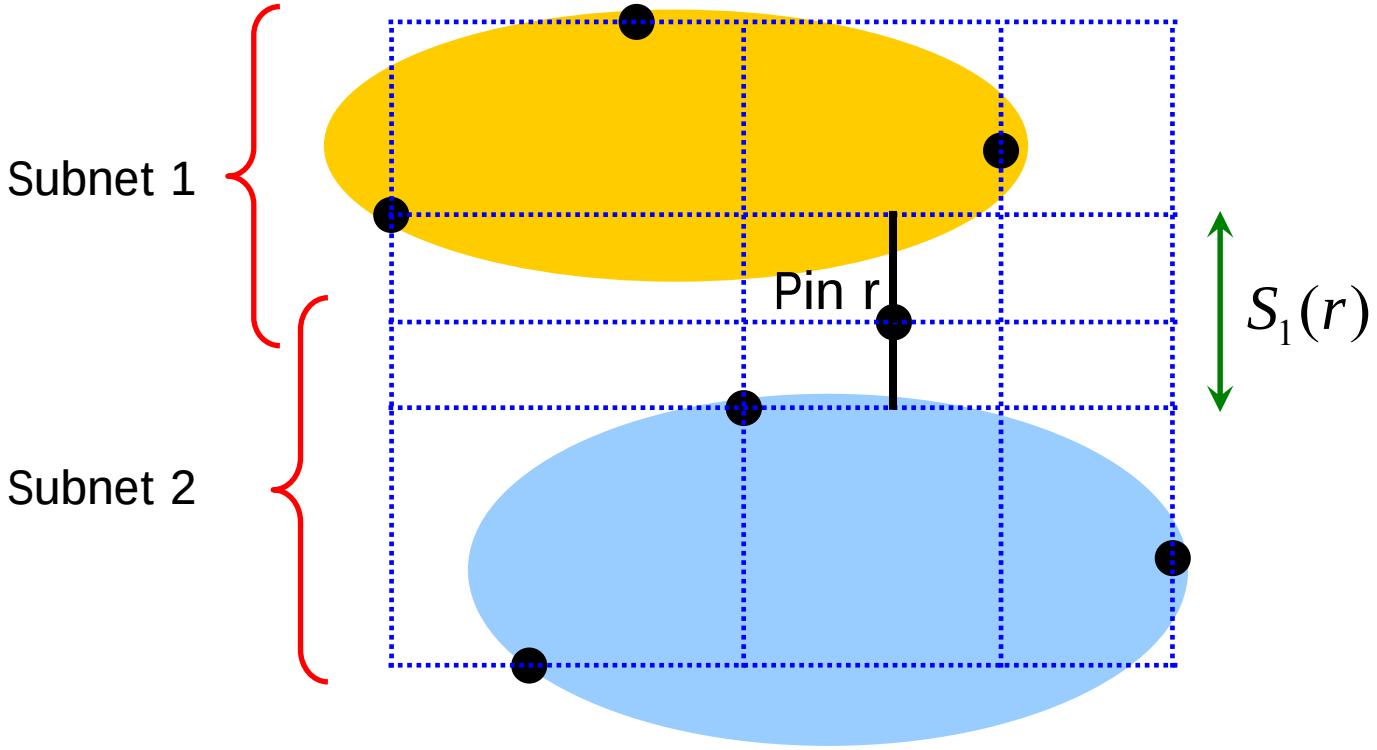
$$S_V(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r)$$



Net Breaking Heuristic #1

- A score for each direction and each pin
- Break in several ways which give the highest scores

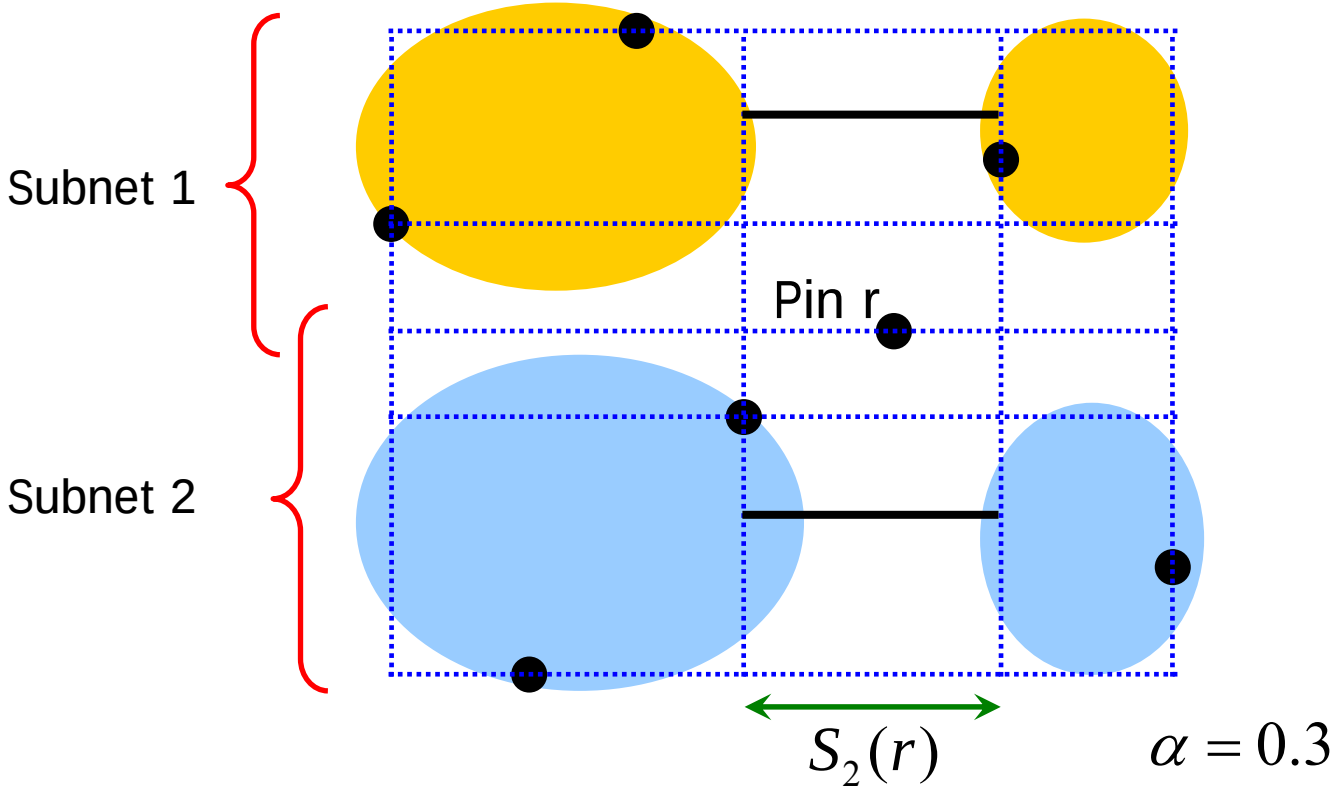
$$S_V(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r)$$



Net Breaking Heuristic #2

- A score for each direction and each pin
- Break in several ways which give the highest scores

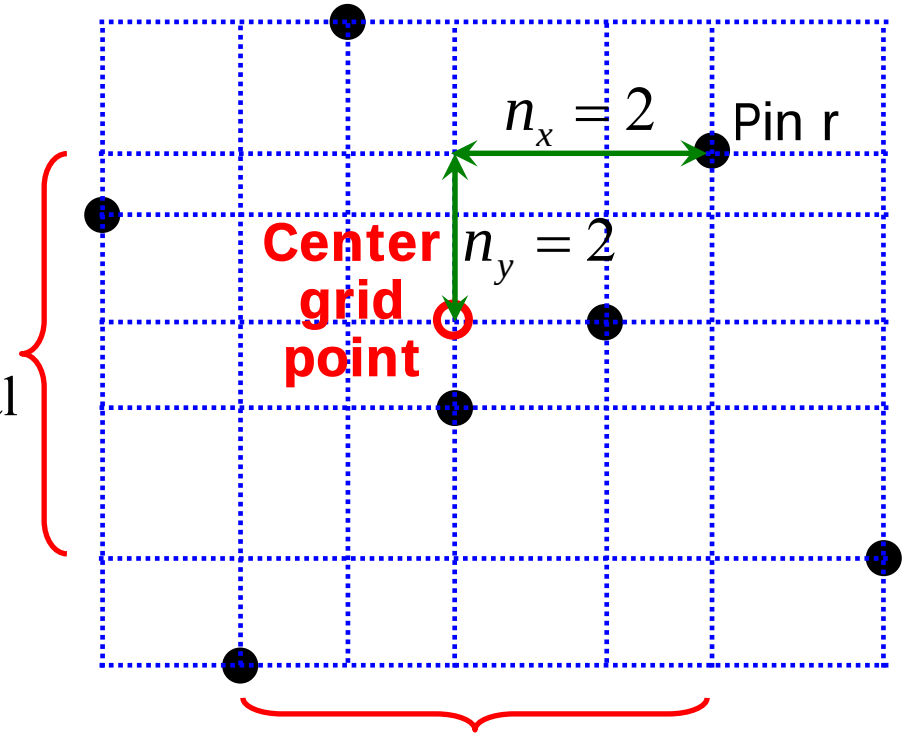
$$S_V(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r)$$



Net Breaking Heuristic #3

- A score for each direction and each pin
- Break in several ways which give the highest scores

$$S_V(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r)$$



$\bar{v}$ = Average vertical segment length

$$S_3(r) = n_x \cdot \bar{h} + n_y \cdot \bar{v}$$

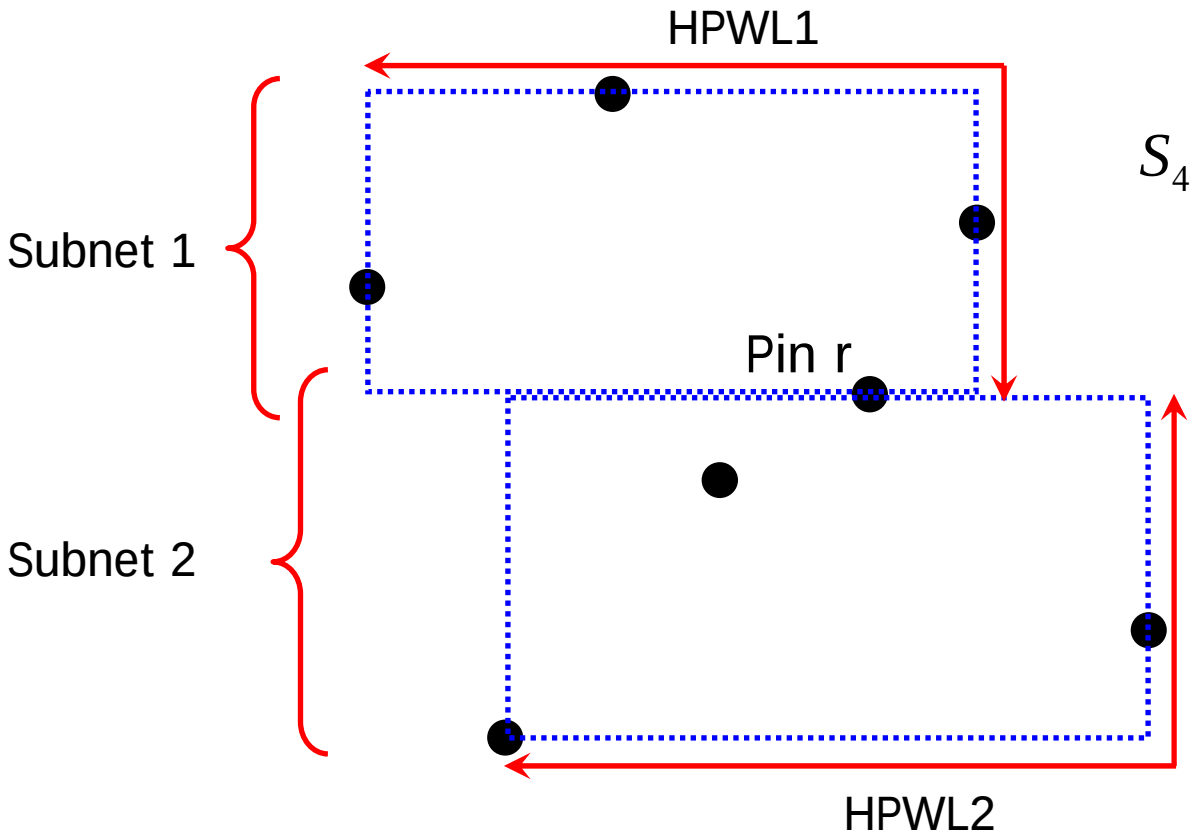
$$\beta = \frac{7.4}{n+10}$$

$\bar{h}$ = Average horizontal segment length

Net Breaking Heuristic #4

- A score for each direction and each pin
- Break in several ways which give the highest scores

$$S_V(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r)$$

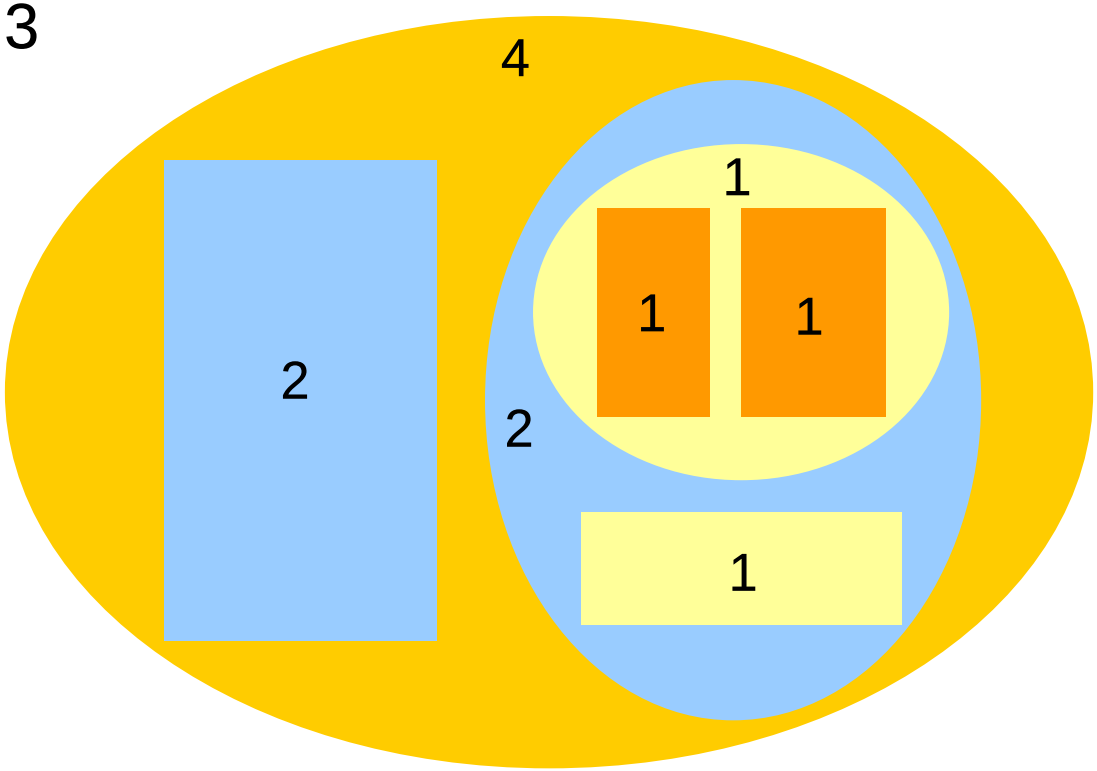


$$S_4(r) = HPWL1 + HPWL2$$

$$\gamma = \frac{4.8}{n-1}$$

Accuracy Control Scheme

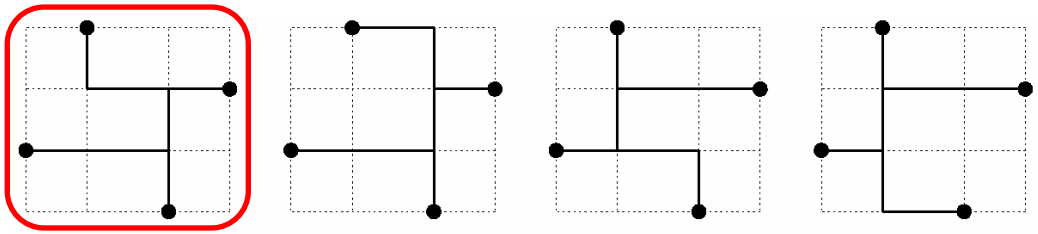
- Accuracy parameter A
- Break a net in A ways with the highest scores
- Subnets are handled with accuracy $\max\{\lfloor A/2 \rfloor, 1\}$
- Runtime complexity = $O\left(A^{\frac{\log A+1}{2}} n \log n\right)$
- Default A=3



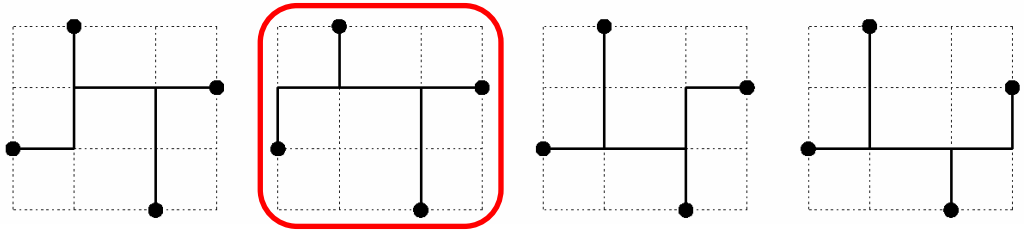
Extension for RSMT Construction

- If degree <= D, store 1 routing topology for each POWV

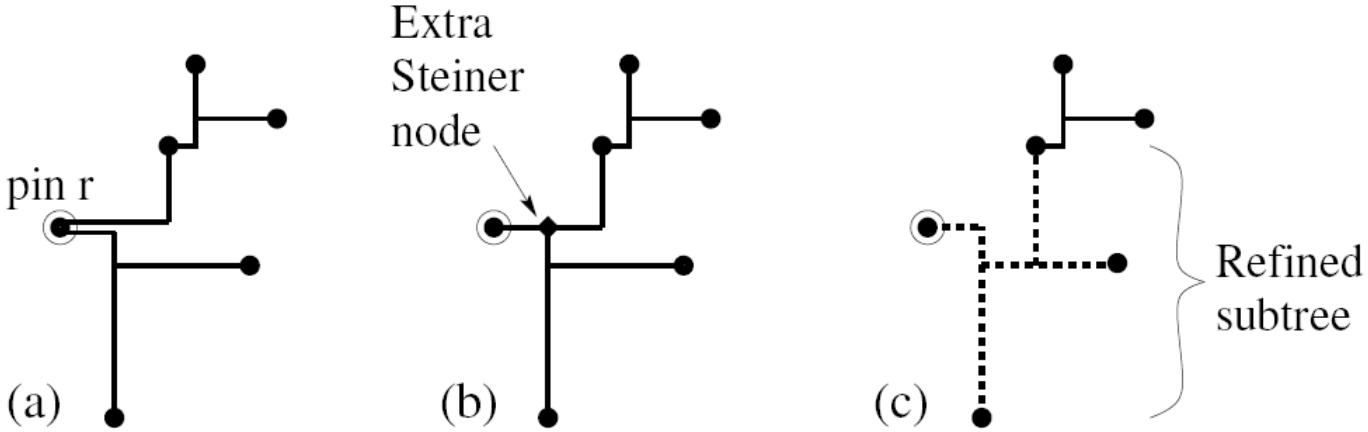
POWV
(1,2,1,1,1,1)

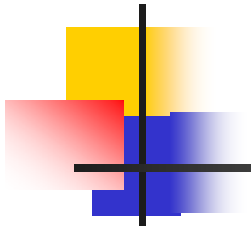


POWV
(1,1,1,1,2,1)



- If degree > D, Steiner trees of two sub-nets are combined
  - The local sub-tree around the merging pin can be refined by FLUTE

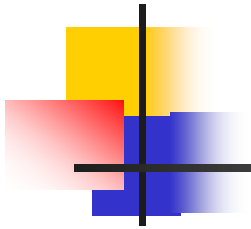




Presentation Outline

- LUT idea to handle RSMT construction
- Boundary compaction technique to generate LUT
- MST-based approach to speed up WL computation
- Net breaking technique to handle high degree nets

- Experimental results



Experimental Setup

- Comparing five techniques:
  - RMST – Prim's RMST algorithm
    - Prim [BSTJ 57]
  - RST-T – Refined Single Trunk Tree
    - Chen et al. [SLIP 02]
  - SPAN – Spanning graph based algorithm
    - Zhou [ISPD 03]
  - BGA – Batched Greedy Algorithm
    - Kahng et al. [ASPDAC 03]
  - BI1S -- Batched Iterated 1-Steiner heuristic
    - Griffith et al. [TCAD 94]
  - FLUTE (version 2.5) with D=9 and A=3
- 18 IBM circuits in the ISPD98 benchmark suite
- Placement by FastPlace [ISPD 04]
- Optimal solutions by GeoSteiner 3.1 (Warme et al.)

Benchmark Information

Circuit	# of nets	Ave. degree	Max. degree
ibm01	14111	3.58	42
ibm02	19584	4.15	134
ibm03	27401	3.41	55
ibm04	31970	3.31	46
ibm05	28446	4.44	17
ibm06	34826	3.68	35
ibm07	48117	3.65	25
ibm08	50513	4.06	75
ibm09	60902	3.65	39
ibm10	75196	3.96	41
ibm11	81454	3.45	24
ibm12	77240	4.11	28
ibm13	99666	3.58	24
ibm14	152772	3.58	33
ibm15	186608	3.84	36
ibm16	190048	4.10	40
ibm17	189581	4.54	36
ibm18	201920	4.06	66
All	1570355	3.92	134

Accuracy Comparison

Circuit	Wirelength error (%)					
	RMST	RST-T	SPAN	BGA	BI1S	FLUTE
ibm01	4.092	1.933	0.251	0.129	0.106	0.074
ibm02	5.849	3.780	0.331	0.143	0.115	0.209
ibm03	4.637	1.919	0.271	0.125	0.095	0.062
ibm04	4.048	1.255	0.203	0.084	0.060	0.051
ibm05	4.489	3.134	0.329	0.153	0.112	0.106
ibm06	5.964	2.822	0.381	0.182	0.134	0.084
ibm07	4.720	1.704	0.268	0.116	0.084	0.046
ibm08	4.784	4.445	0.328	0.162	0.123	0.261
ibm09	4.331	1.804	0.235	0.105	0.075	0.042
ibm10	4.104	1.790	0.252	0.104	0.080	0.051
ibm11	4.018	1.227	0.219	0.087	0.062	0.024
ibm12	3.783	1.908	0.248	0.106	0.077	0.054
ibm13	4.782	2.002	0.292	0.135	0.102	0.053
ibm14	3.908	1.540	0.221	0.095	0.068	0.040
ibm15	4.201	1.941	0.266	0.106	0.077	0.062
ibm16	4.231	2.421	0.279	0.124	0.090	0.068
ibm17	3.905	2.188	0.263	0.110	0.082	0.056
ibm18	4.432	3.353	0.300	0.134	0.100	0.147
All	4.232	2.261	0.269	0.117	0.086	0.075

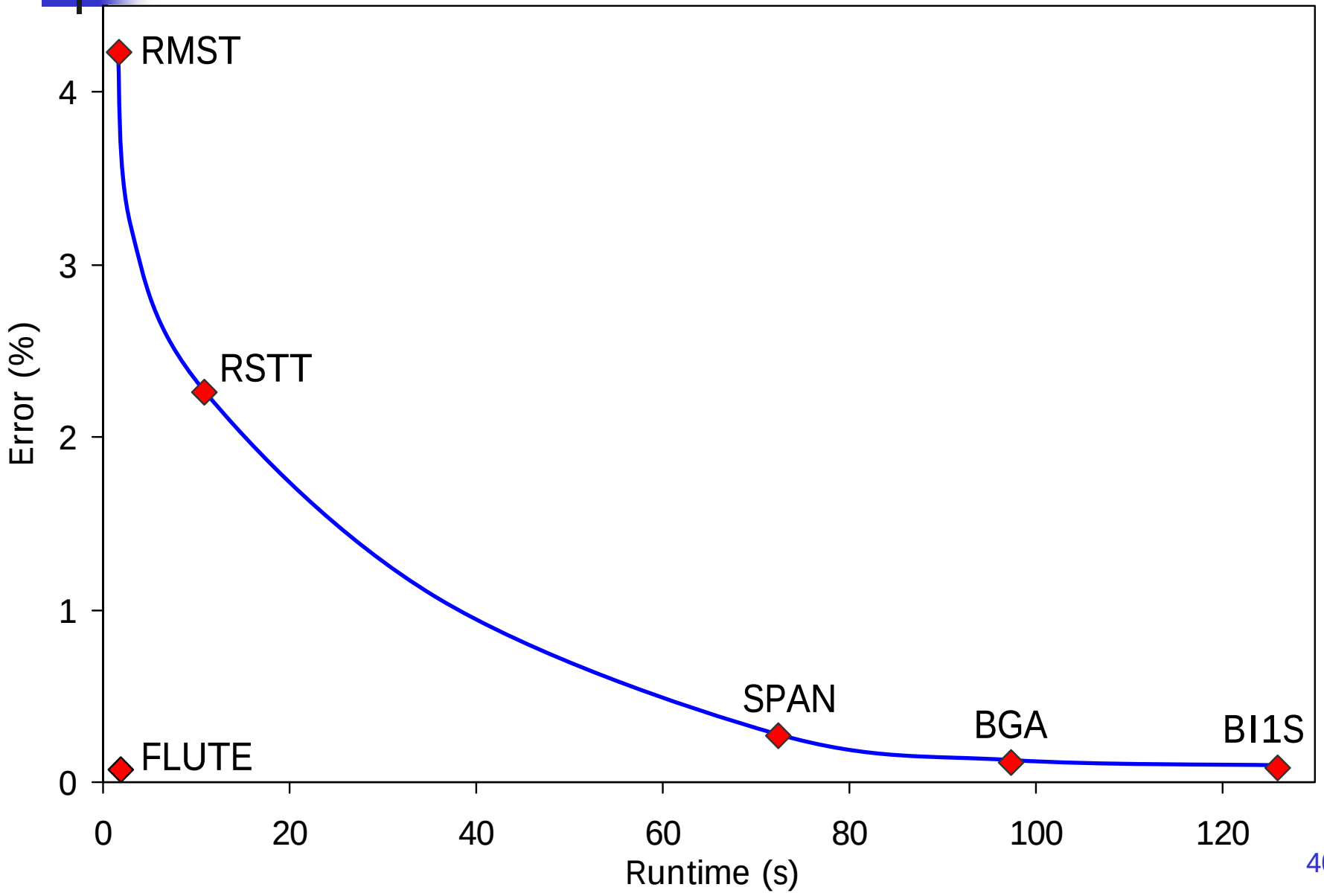
Runtime Comparison

All experiments are carried out on a 3.4-GHz Pentium 4 machine

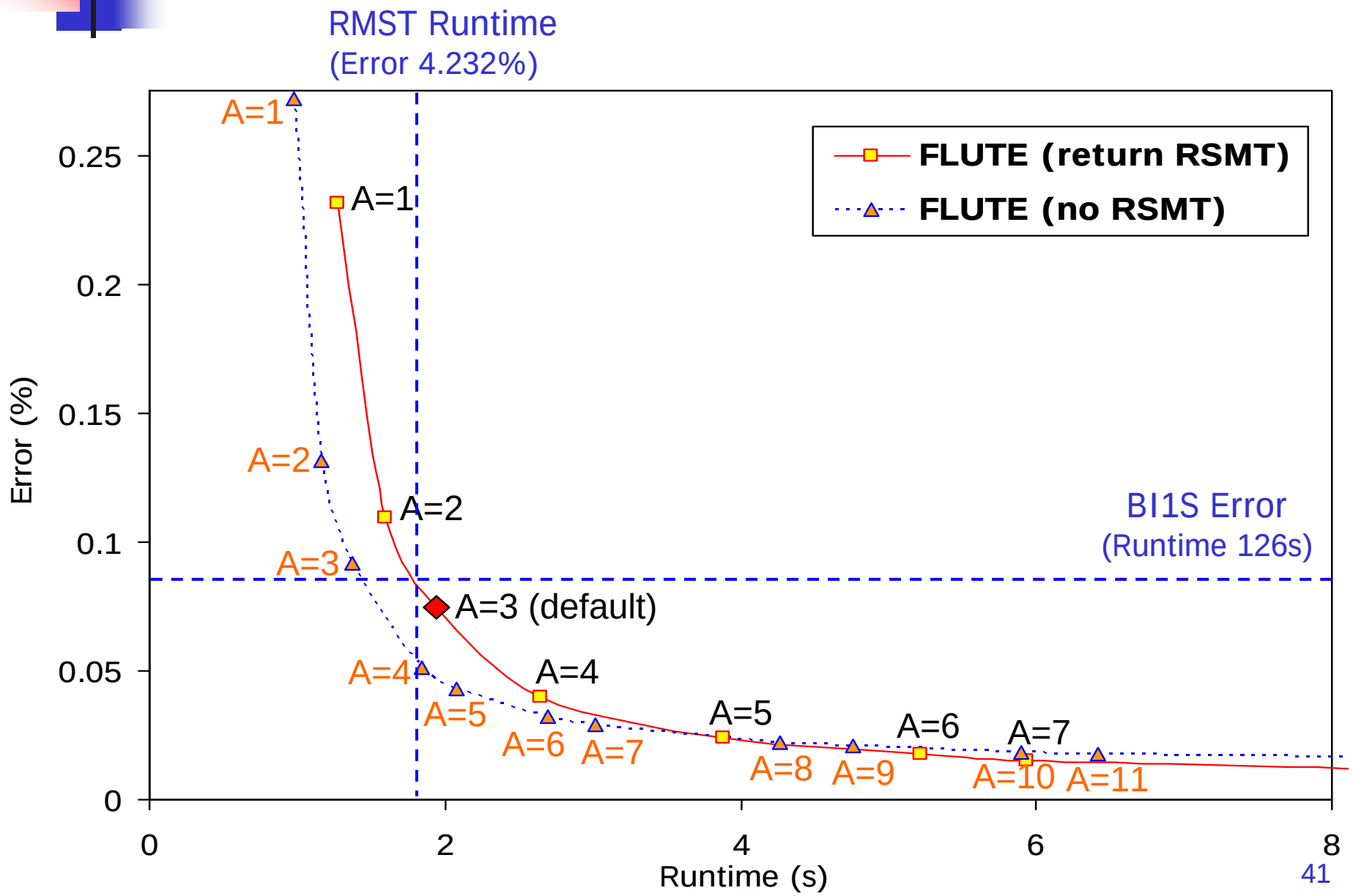
Circuit	Runtime (s)					
	RMST	RST-T	SPAN	BGA	BI1S	FLUTE
ibm01	0.02	0.09	0.55	0.75	1.01	0.02
ibm02	0.02	0.14	1.05	1.50	4.32	0.03
ibm03	0.02	0.18	1.02	1.38	1.95	0.03
ibm04	0.04	0.20	1.07	1.44	2.24	0.02
ibm05	0.03	0.20	1.71	2.40	2.69	0.05
ibm06	0.03	0.23	1.45	1.95	2.53	0.04
ibm07	0.05	0.32	1.96	2.59	3.26	0.04
ibm08	0.06	0.35	2.63	3.74	6.60	0.09
ibm09	0.07	0.40	2.42	3.19	4.13	0.06
ibm10	0.08	0.53	3.59	4.77	5.85	0.09
ibm11	0.06	0.53	2.87	3.76	5.16	0.05
ibm12	0.10	0.54	3.94	5.33	6.25	0.10
ibm13	0.10	0.66	3.89	5.18	6.68	0.09
ibm14	0.15	1.02	5.91	7.84	10.11	0.14
ibm15	0.21	1.27	8.18	10.86	13.96	0.22
ibm16	0.23	1.33	9.33	12.47	14.75	0.26
ibm17	0.28	1.39	11.06	15.06	16.63	0.31
ibm18	0.26	1.40	9.81	13.28	17.82	0.30
All	0.93	5.56	37.34	50.25	64.92	1.0

Normalized →

Accuracy vs. Runtime Tradeoff



Effect of Accuracy Control Parameter



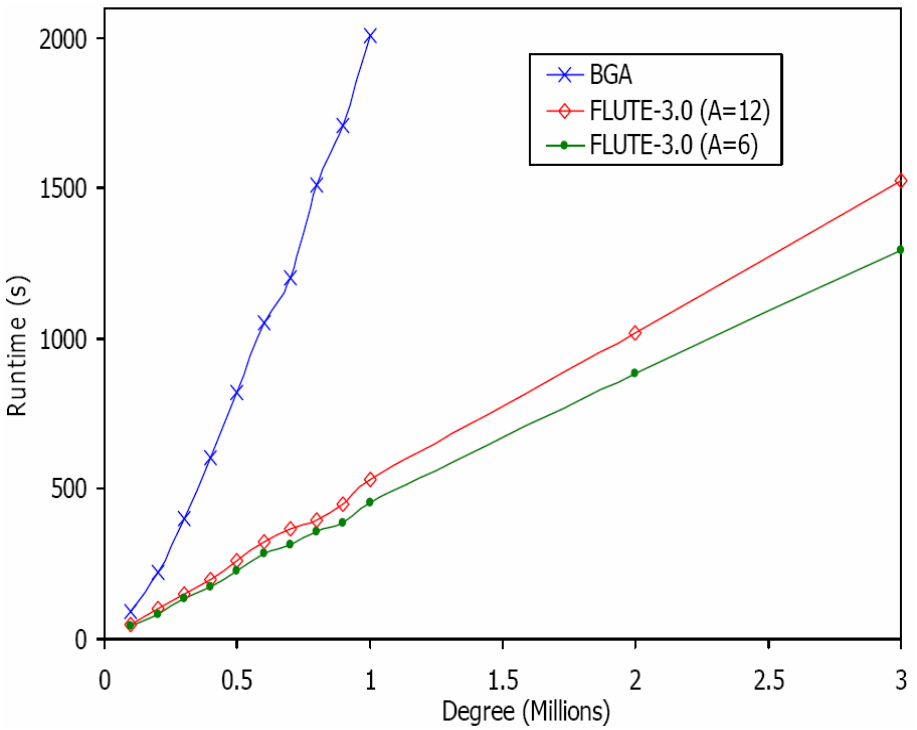
Breakdown According to Net Degree

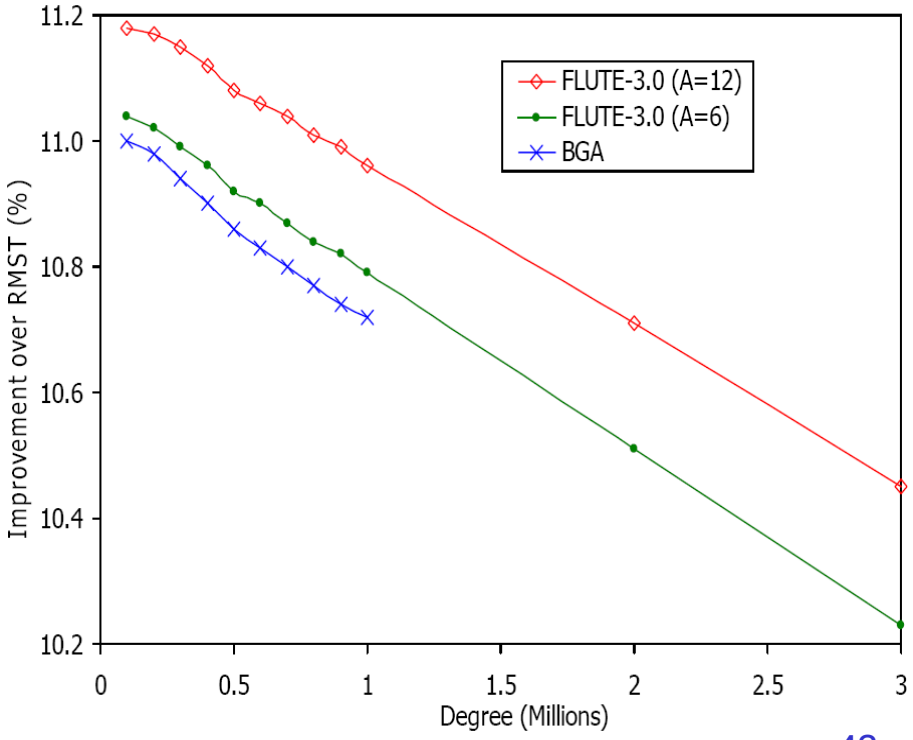
- All 1.57 million nets in 18 circuits
  - Average degree = 3.92
  - 8.13% with degree $\geq$ 10 (but 26.2% of WL)
  - 0.077% with degree $\geq$ 30
  - 0.005% with degree $\geq$ 60

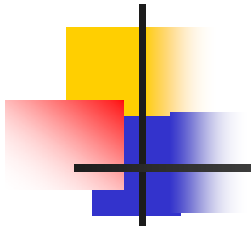
Degree	Net breakdown		Wirelength error (%)					
	#	WL	RMST	RST-T	SPAN	BGA	BI1S	FLUTE
2	54.92%	27.98%	0.00	0.00	0.00	0.00	0.00	0.00
3	14.40%	10.26%	2.50	0.00	0.03	0.00	0.00	0.00
4	7.68%	7.84%	3.89	0.00	0.11	0.00	0.00	0.00
5	5.61%	8.18%	4.74	0.00	0.21	0.07	0.05	0.00
6	3.20%	5.65%	5.40	0.49	0.29	0.12	0.07	0.00
7	2.28%	4.82%	5.91	1.02	0.37	0.13	0.09	0.00
8	1.98%	4.61%	6.25	1.91	0.42	0.16	0.12	0.00
9	1.81%	4.46%	6.79	2.65	0.48	0.21	0.15	0.00
10–17	6.98%	21.72%	7.81	6.21	0.60	0.29	0.22	0.16
$\geq$18	1.15%	4.48%	9.04	14.05	0.75	0.40	0.32	0.87

Improvement For High-Degree Nets

- Yiu-Chung Wong and Chris Chu, “A Scalable and Accurate Rectilinear Steiner Minimal Tree Algorithm”, VLSI-DAT 2008
- For high-degree nets (with tens of pins or more), net breaking according to rectilinear minimum spanning tree
  - Complicated merging techniques to achieve extraordinary accuracy







Conclusion

- FLUTE:
  - Rectilinear Steiner Minimal Tree algorithm
  - Post-placement pre-routing wirelength estimation
- Very suitable for VLSI applications:
  - Optimal and extremely fast up to degree 9
  - Very accurate and fast up to degree 100
  - Nice tradeoff between accuracy and runtime
- Key ideas:
  - Pre-computed POWVs by boundary compaction
  - Store the POWVs and corresponding RSMTs in LUT
  - MST-based approach to speed up WL computation
  - Net-breaking technique to handle large nets

Extension and Future Works

- Extension:
  - Delay-driven Steiner tree construction

    Min Pan, Chris Chu and Priyadarsan Patra, “A Novel Performance-Driven Topology Design Algorithm”, ASPDAC 2007.

- Future Works:
  - Extend FLUTE for RSMT construction with obstacles
  - Design LUT-based practical algorithms for other NP-complete problems

★Source code available in GSRC Bookshelf:

http://home.eng.iastate.edu/~cnchu/flute.html

Thank You