Seeing the Forest and the Trees: Steiner Wirelength Optimization in Placement

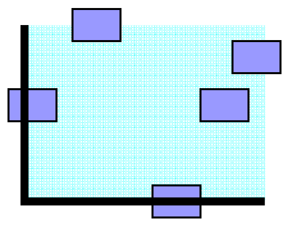
Jarrod A. Roy, James F. Lu and Igor L. Markov
University of Michigan Ann Arbor

Outline

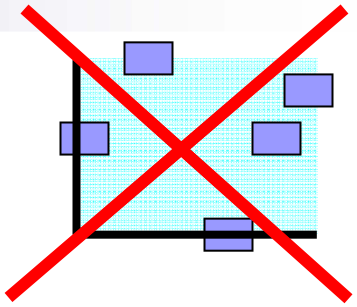
- Motivation
  - Why current placement tools are outdated
  - Analysis of placement objectives
  - A naïve attempt at optimization
- Our placement framework
- New techniques
- Empirical results
- Conclusions

Motivation (1)

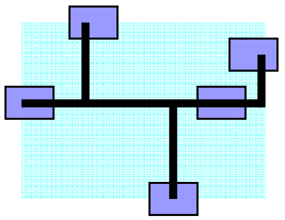
- Place-and-route
  - Pivotal step in any design flow
  - Closely related to physical synthesis
  - Is becoming harder every year
    - Greater scale, “boulders and dust”, fixed obstacles
    - Novel design techniques require P&R support
    - Heavily affected by variability
- P&R in tool flows
  - Single step for designers?
  - P&R implemented as separate point tools
  - Very little interaction/communication
  - Use different optimization objectives



Motivation (2)

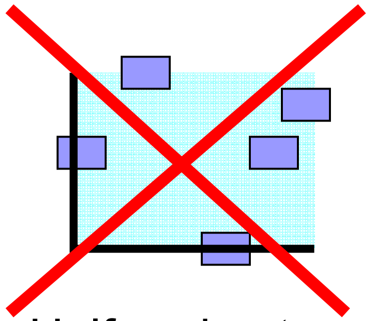


- The HPWL (half-perimeter wirelength) objective hopelessly outdated – does not account for
  - Routing demand of multi-pin nets
  - Detours around obstacles
  - Vias
  - Impact of buffers on delay (and where buffers can be inserted)

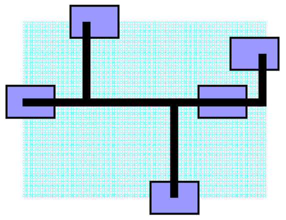


- Our goal: reduce the gap between placement and routing by **replacing the HPWL objective with realistic routes**
  - Empirical results: consistent improvement over all published P&R results
  - *Routability, routed wirelength, via counts*
  - *Compared to Silicon Ensemble (Cadence): 26% better routed WL, 3% fewer vias*

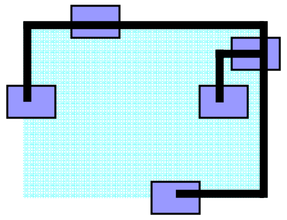
HPWL vs. Steiner Tree WL vs. MST WL



Half-perimeter wirelength



Steiner (tree) wirelength

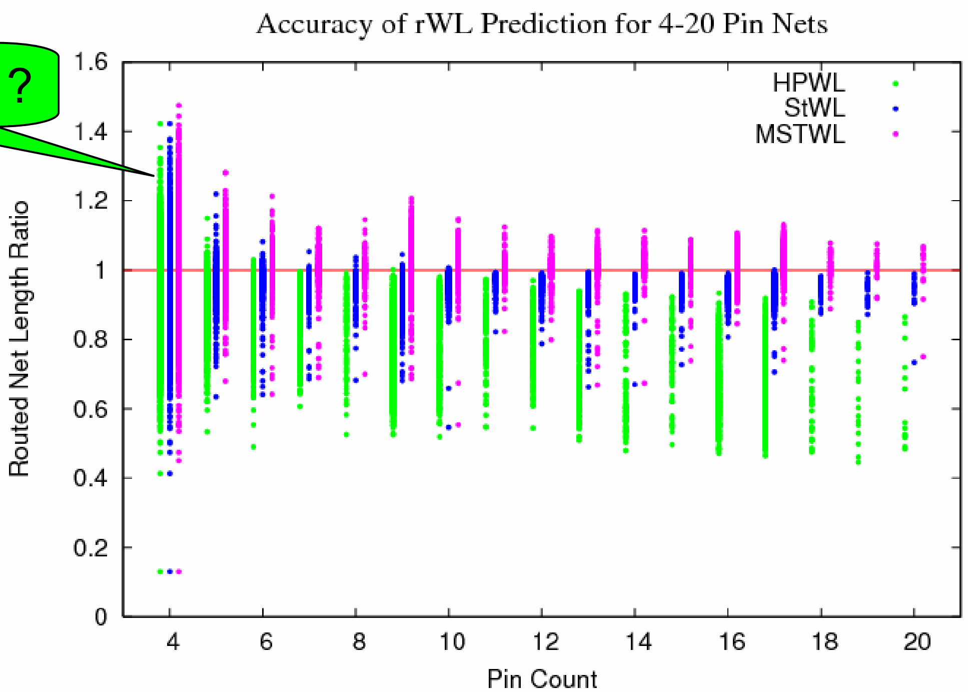


Minimum Spanning Tree (MST) wirelength

- HPWL ≤ Steiner Tree WL ≤ MST WL

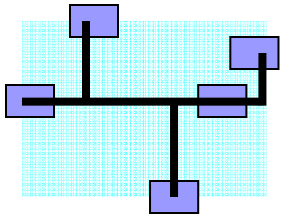
HPWL > rWL ?

Internal cell wiring not counted in rWL



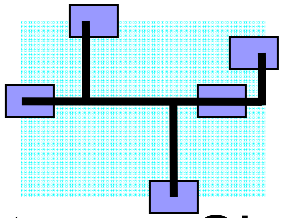
- MST WL: most accurate an average
- Steiner WL: best *fidelity*

Computing Steiner Trees



- Computing HPWL takes linear time, MST super linear *(P log P)*, but Steiner trees are NP-hard
- **Steiner Tree tools we evaluate**:
  - Batched Iterated 1-Steiner (BI1ST) [Kahng,Robins 1992]
    - Slow (n^3)
    - Very accurate, even for 20+ pins
  - FastSteiner [Kahng,Mandoiu,Zelikovsky 2003]
    - Faster but less accurate than BI1ST
  - FLUTE [Chu 2004, 2005]
    - Very fast
    - Optimal lookup tables for ≤ 9 pins
    - Less accurate for 10+ pins

Optimizing Steiner Tree Length



+

= ?

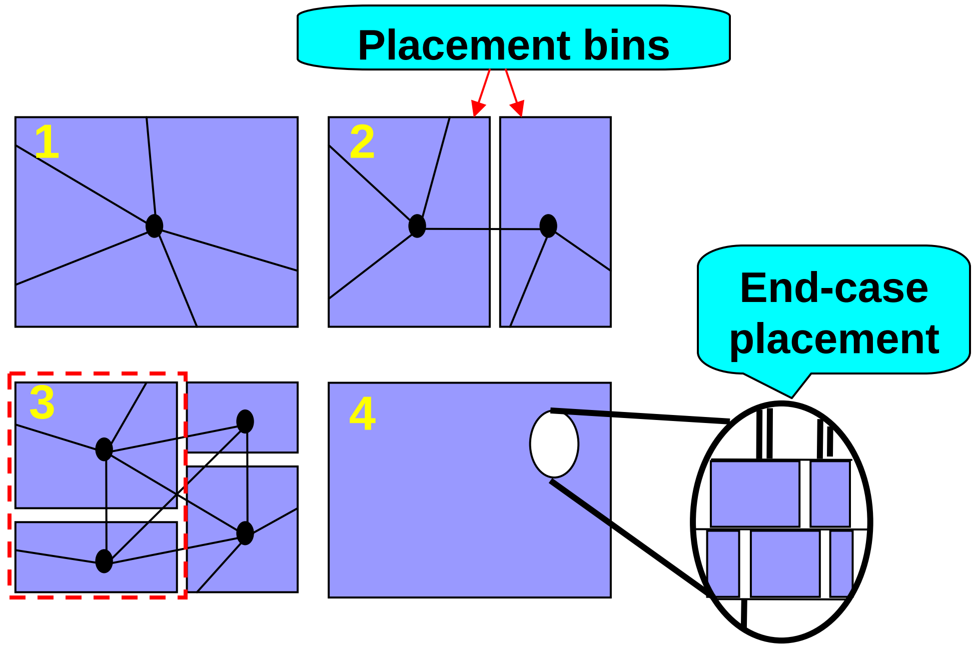
- Simple experiment
  - Take a floorplanner that uses Sim. Annealing (we used Parquet)
  - Consider the wirelength term in its objective function
  - Replace the HPWL computation with Min. Steiner-tree length (we used FLUTE)
- Empirical observations
  - Slow-down (even for 3-pin nets) – expected
  - Did not improve StWL – very surprising result !

Outline

- Motivation
  - Why current placement tools are outdated
  - Analysis of placement objectives
  - A naïve attempt at optimization
- Our placement framework
- New techniques
- Empirical results
- Conclusions

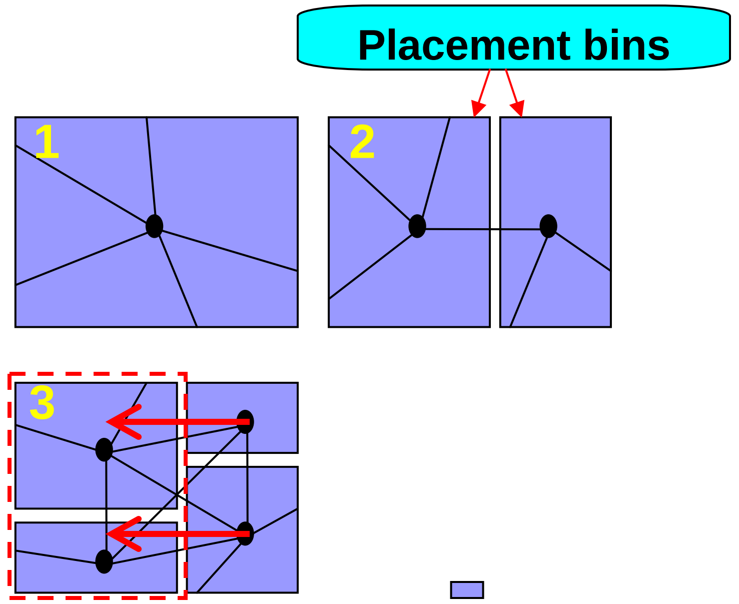
Existing Placement Framework

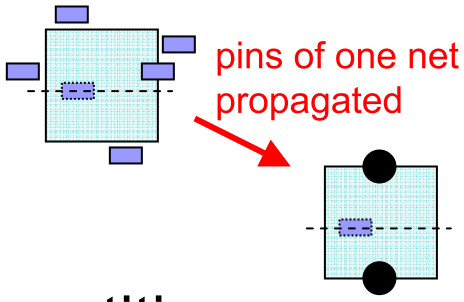
- Consider *placement bins*
- Partition them
  - Use min-cut bisection
  - Place end-cases optimally



- Traditional min-cut placement tracks HPWL

Existing Placement Framework





- ***Propagate terminals*** before partitioning
  - Terminals: fixed cells or cells outside current bin
  - Assigned to one of partitions
- Save runtime: a 20-pin may "propagate" into 3-pin net
  - "Inessential nets": fixed terminals in both partitions (can be entirely ignored)
- Traditional min-cut placement tracks HPWL

Better Modeling of HPWL by Net Weights In Min-cut

- Introduced in Theto placer [Selvakkumaran 2004]
- Refined in [Chen 2005]
    - Shown to accurately track HPWL
- Uses three net costs
    - w_{left}: HPWL when all cells on left side (a)
    - w_{right}:HPWL when all cells on the right (b)
    - w_{cut}: HPWL when cells on both sides (c)
- In min-cut partitioning, represents each net with 1 or 2 hyper-edges

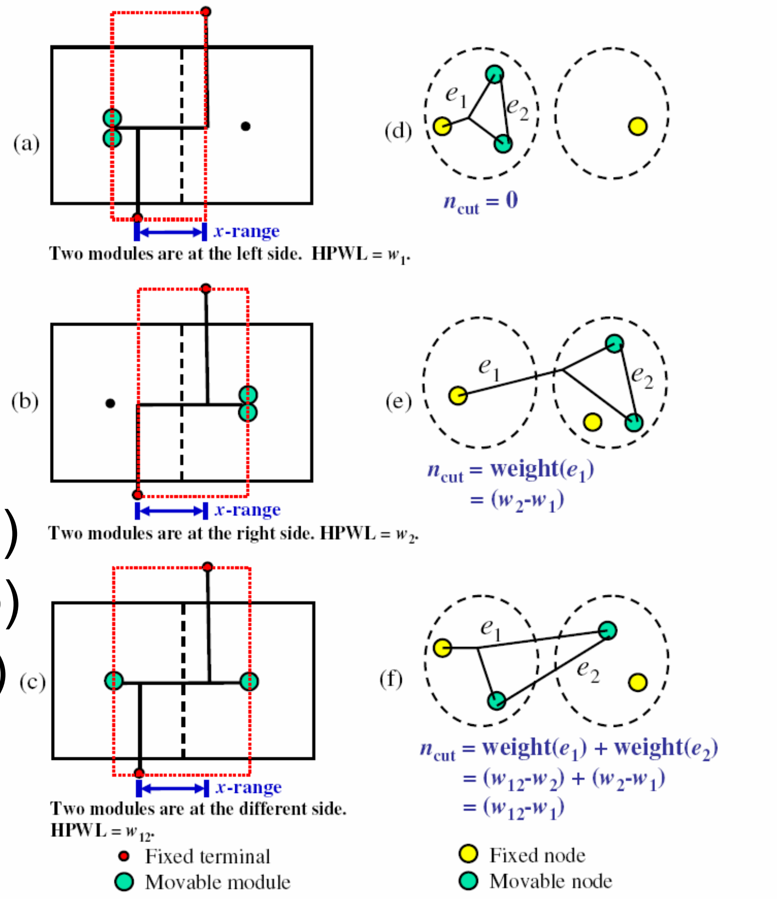
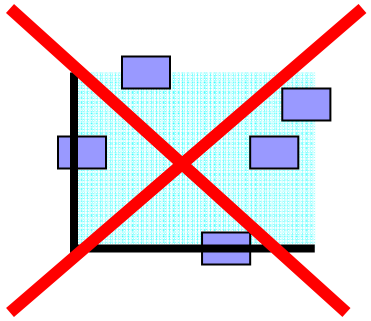

(a)
x-range
Two modules are at the left side. HPWL = w_1.
(d)
e_1
e_2
$n_{cut} = 0$
(b)
x-range
Two modules are at the right side. HPWL = w_2.
(e)
e_1
e_2
n_{cut} = weight(e_1) = (w_2-w_1)
(c)
x-range
Two modules are at the different side. HPWL = w_{12}.
(f)
e_1
e_2
n_{cut} = weight(e_1) + weight(e_2) = (w_{12}-w_2) + (w_2-w_1) = (w_{12}-w_1)
Fixed terminal
Movable module
Fixed node
Movable node

Fig. 2. An example of determining a net weight. (a), (b), and (c) are three possible partitioning results. (d), (e), and (f) are corresponding partitioning hypergraphs.

Figure from [Chen,Chang,Lin 2005]

Key Observation

- For bisection,
  cost of each net is characterized by 3 cases
  - Cost of net when cut w_{cut}
  - Cost of net when entirely in left partition: w_{left}
  - Cost of net when entirely in right partition: w_{right}
- In our work, we compute these costs using realistic routes
  - Can/should account for both X and Y components of cost
  - Real difficulty in data structures!



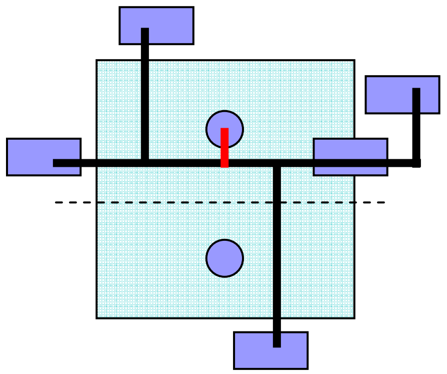
Our Contributions

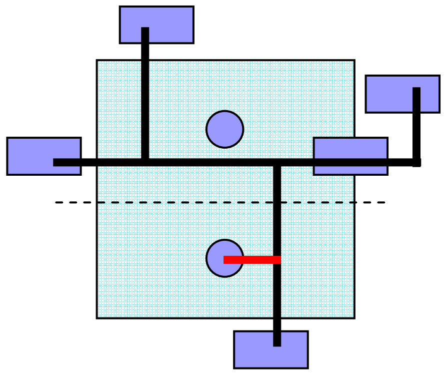
- Optimization of Steiner WL
  - In global placement (runtime penalty ~25%)
  - In detail placement
- Whitespace allocation to tame congestion
- Empirical evaluation of ROOSTER
  - No violations on 16 IBMv2 benchmarks (easy + hard)
  - Consistent improvements of published results
  - 4-10% by routed wirelength
  - 10-15% by via counts
- Vs Cadence: 26% better rWL, 3% fewer vias

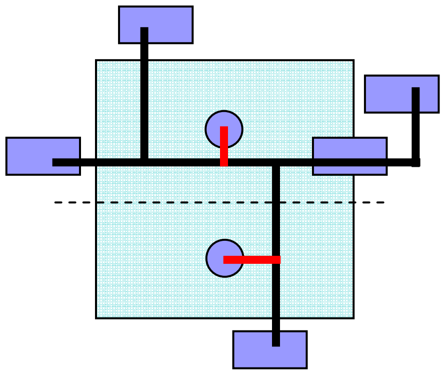
Optimizing Steiner WL During Global Placement

- Recall: each net can be modeled by 3 numbers
  - This has only been applied to HPWL optimization
- We calculate w_{top}, w_{bottom}, w_{cut} using Steiner-tree evaluator
  - For each net, <u>before partitioning starts</u>
  - The bottleneck is still in partitioning $\rightarrow$ can afford a fast Steiner-tree evaluator

Net Weights from Steiner Trees

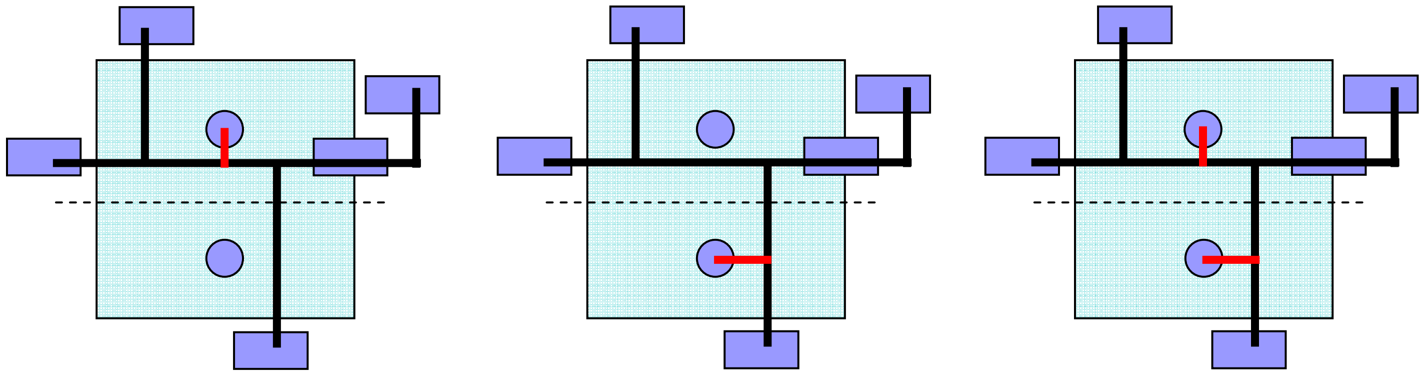

w_{top}


w_{bottom}


w_{cut}

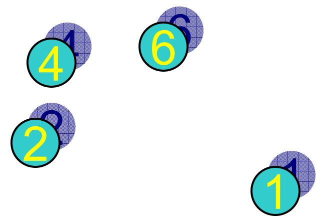
- For horizontal cutlines: w_{top}, w_{bottom}, w_{cut}
  - For vertical cutlines: w_{left}, w_{right}, w_{cut}
- Optimal tree may look very different for each cost
  - Recompute tree from scratch each time

Net Weights from Steiner Trees



- Pitfall : cannot propagate terminals !
  - Nets that were inessential are now essential
  - Must consider all pins of each net
  - **More accurate modeling, but potentially much slower**

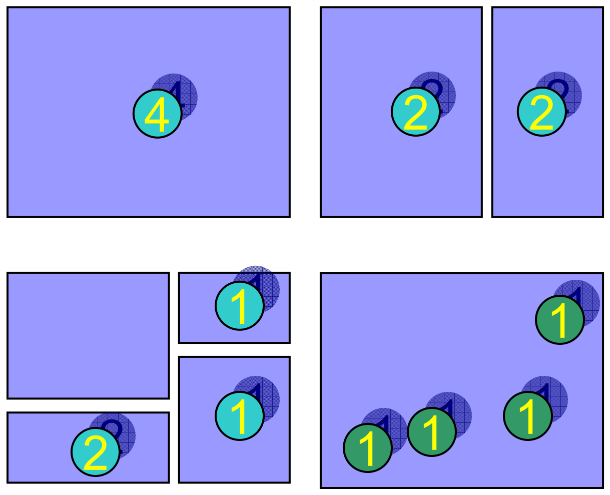
New Data Structure for Global Placement



- For each net, two ***pointsets with multiplicities***
  - Unique locations of <u>fixed</u> & <u>movable</u> pins
  - At top placement layers, very few *unique* pin positions (except for fixed I/O pins)
- Avoid repetitive/expensive re-computation
- Maintain the number of pins at each location
  - Sorted by *(x,y)* to enable batched linear-time operations
  - Easy detection of duplicates; binary search
  - Fast maintenance when pins get reassigned to partitions (or move)
- Facilitates efficient computation of the 3 costs
  - If net has large number (> 20) of unique locations, resort to HPWL

Pointsets in Action

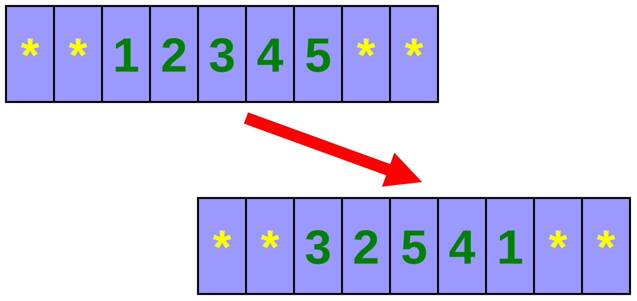
- Consider a net
  with 4 movable pins



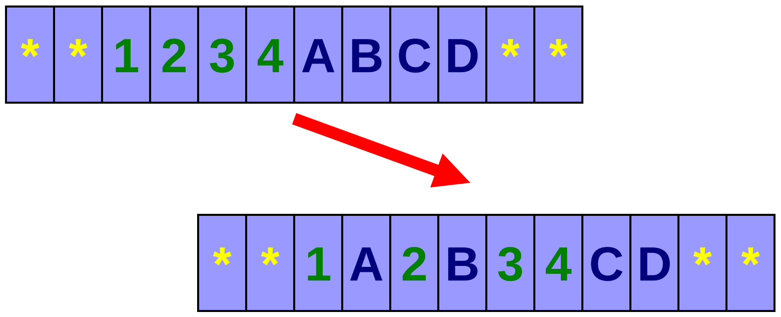
Improvement in Global Placement

- Results depend on the Steiner tree evaluator
  - ***Surprisingly***, running 2 or 3 evaluators and picking min wirelength is worse than using a single evaluator
  - Quality of Steiner-tree evaluation for 9+ pins matters
  - But for 20+ unique locations use HPWL (also tried MST)
- We choose **FastSteiner** (versus BI1ST and FLUTE)
  - Details in Appendix B of our ISPD`06 paper
- Impact of changes to global placement
  - Results consistent across IBMv2 benchmarks
  - Steiner WL ↓2.9% , HPWL ↑1.3%, runtime ↑27%

Optimizing Steiner WL in Detail Placement



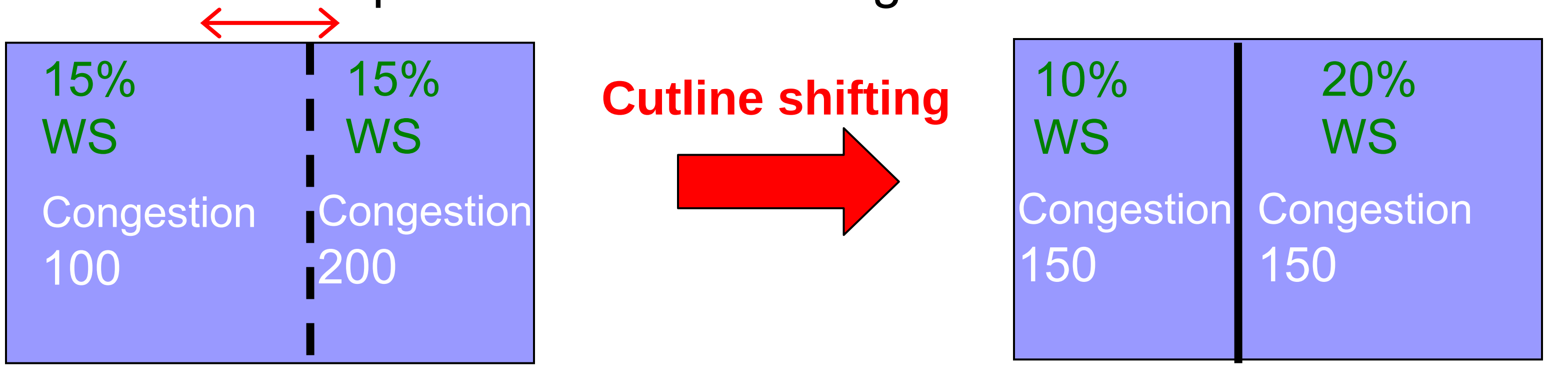
- We leverage the speed of FLUTE with two sliding-window optimizers
  - Exhaustive enumeration for 4-5 cells in a single row
  - Interleaving by dynamic programming (5-8 cells)
    - Explores an exponential solution space in polynomial time
    - Fast but not always optimal



- Steiner WL ↓0.69%, routed WL ↓1.39%
  - [global + detail] runtime ↑**11.83%**

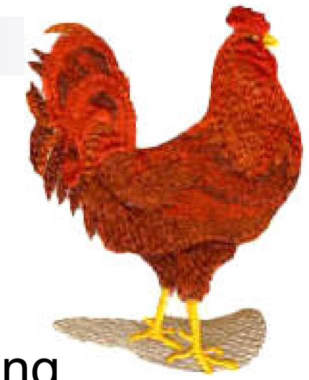
Congestion-based Cutline Shifting

- Non-uniform whitespace allocation
  - Performed during global placement
  - Uses *progressive top-down congestion estimates*
- Main idea: after each min-cut, shift the cutline to balance congestion
  - Area constraints must always be met
  - More whitespace to the more congested bin



- Compared to WSA [Li 2004], no need for legalization, reduces #vias
- Technical difficulty: maintain congestion estimates efficiently over a *slicing floorplan* (not a grid)

Empirical Results: IBMv2



ROOSTER: Rigorous Optimization Of Steiner Trees Eases Routing

Published results:

	Routed WL Ratio	Via Ratio	Routes with Violation
ROOSTER	1.000	1.000	0/16
mPL-R+WSA	1.055	1.156	0/16
APlace 1.0	1.042	1.119	1/8
Capo 9.2	1.056	Not published	0/16
Dragon 3.01	1.107	Not published	1/16
FengShui 2.6	1.093	Not published	7/16

Most recent results:

mPL-R+WSA	1.007	1.069	0/16
APlace 2.04	0.968	1.073	2/16
FengShui 5.1	1.097	1.230	10/16

ROOSTER with several detail placers: IBMv2

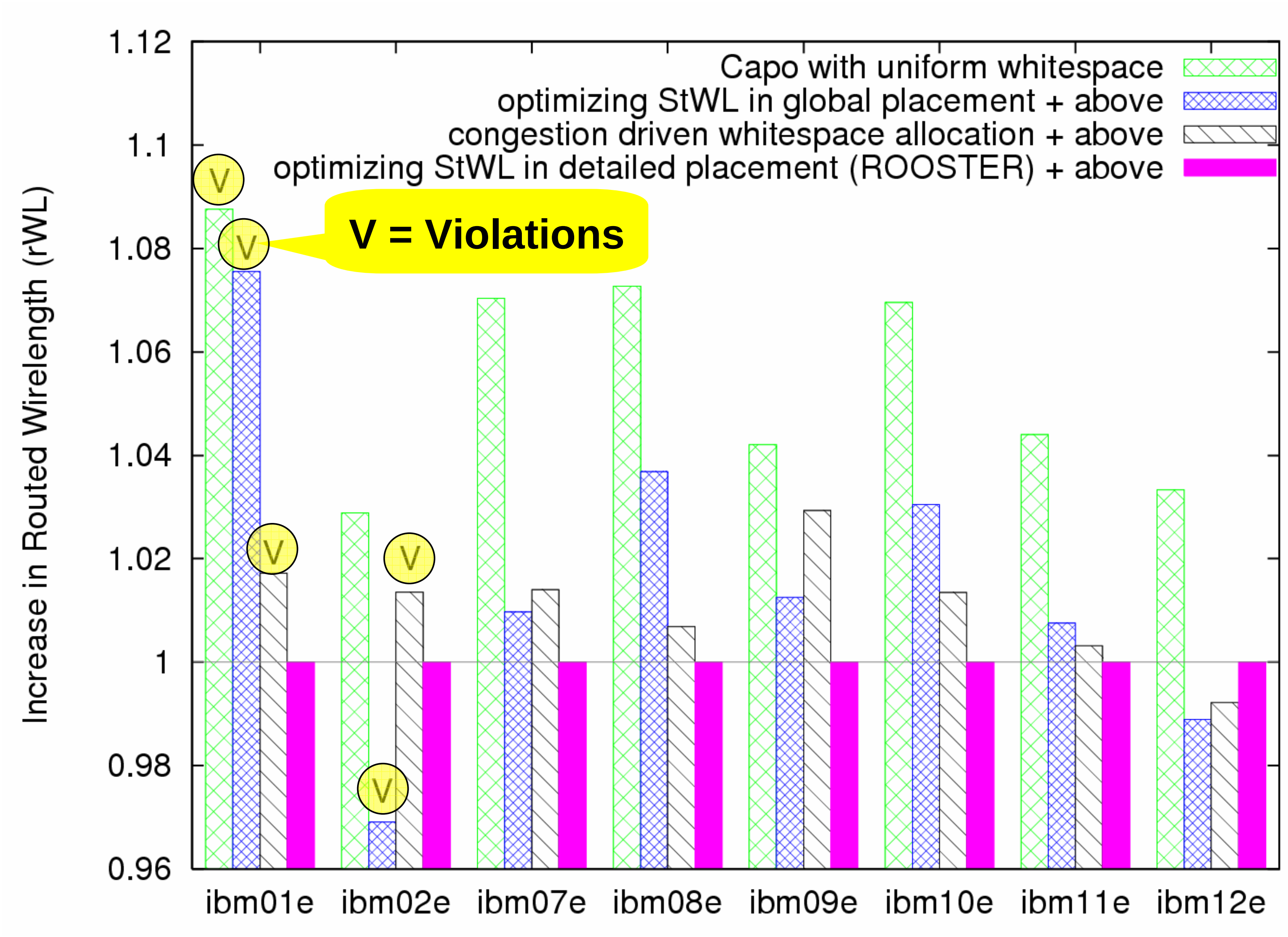
	Routed WL Ratio	Via Ratio	Routes with Violation
ROOSTER	1.000	1.000	0/16
ROOSTER+WSA	0.990	1.004	0/16
ROOSTER+ Dragon 4.0 DP	1.041	1.089	2/16
ROOSTER+ FengShui 5.1 DP	1.114	1.248	16/16

AmoebaPlace vs.

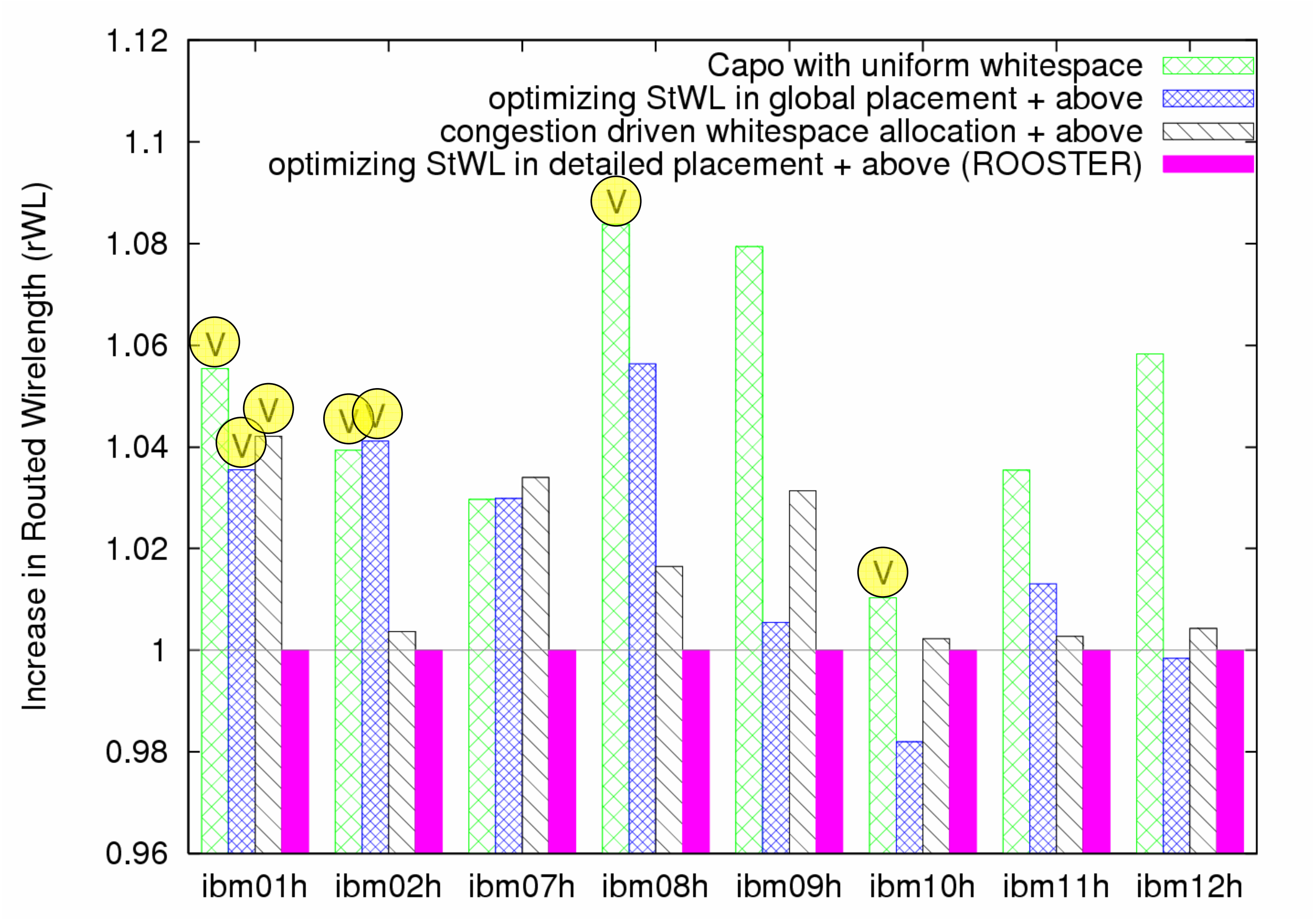
- IWLS 2005 benchmarks
  - http://iwls.org/iwls2005/benchmarks.html
- All IWLS placements routed with NanoRoute

	Rooster			AmoebaPlace		
	rWL	Vias	Viols	rWL	Vias	Viols
aes_core	**1.271**	**126645**	1	1.657	131049	1
ethernet	**6.145**	**413323**	2	7.745	471800	1
mem_ctrl	**0.890**	**89153**	0	1.224	90067	0
pci_bridge32	**1.176**	**115675**	0	1.598	117326	2
usb_funct	**0.860**	**85329**	0	1.106	85739	0
vga_lcd	**24.447**	1083504	1	25.405	**1076178**	2
Ratio	1.000	1.000		**1.265**	**1.032**	

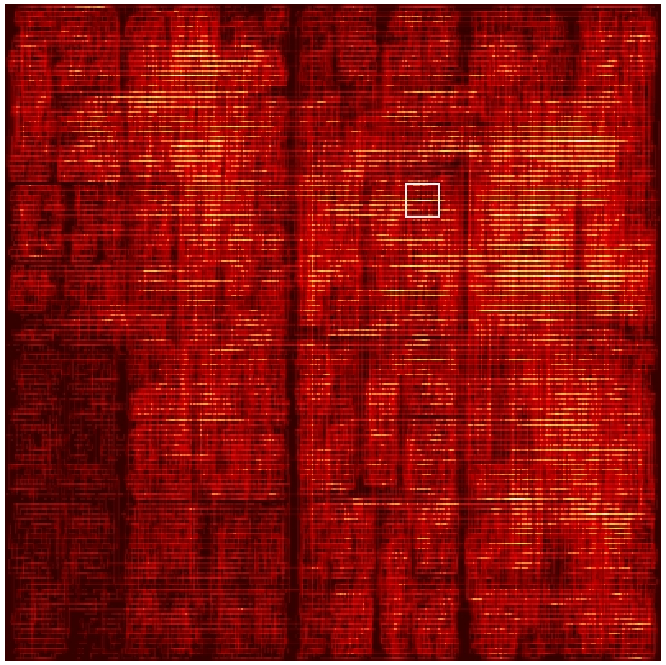
Improvement Breakdown: IBMv2 easy

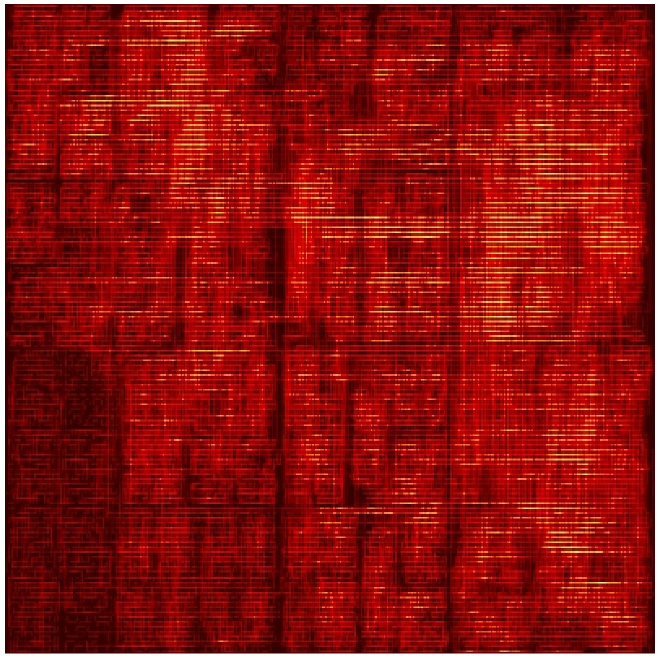


Improvement Breakdown: IBMv2 hard



Congestion with and without





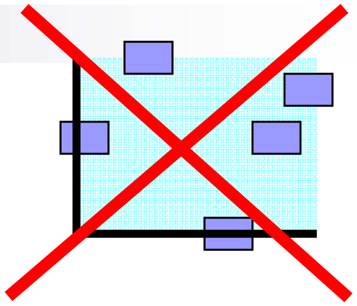
Capo -uniformWS

ROOSTER

5 hours to route; 120 violations

22 mins to route; 0 violations

Conclusions



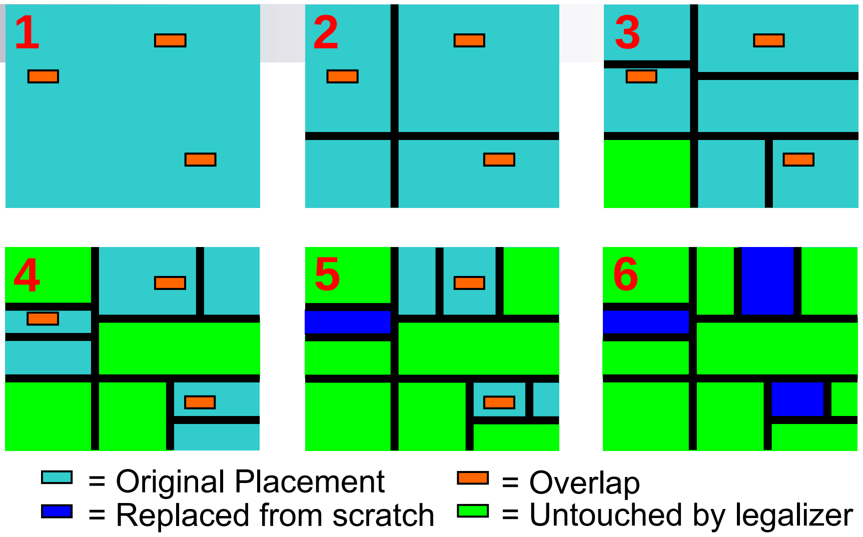
- Steiner WL should be optimized in global and detail placement
  - Improves routability and routed WL
  - 10-15% improvement in via counts (vs academic placers)
  - Better Steiner evaluators may further reduce routed WL
- Congestion-driven cutline shifting in global placement is competitive with WSA
  - Better via counts
  - May be improved if better congestion maps available
- Compared to Cadence P&R
  - 26% reduction in routed WL
  - 3% fewer vias
- ROOSTER freely available for all uses http://vlsicad.eecs.umich.edu/BK/PDtools

Ongoing Work: ECO-system

- Challenge: repair/improve an existing placement
  - A strong detail placer and legalizer (useful with analytical global placers)
  - A strong ECO placer (useful in physical synthesis)
- Complications: fixed obstacles, movable macros
- Philosophy
  - Do no harm (leave most cells where they are)
  - When a section of layout must be redone, be prepared to re-place all gates in a region

ECO-system

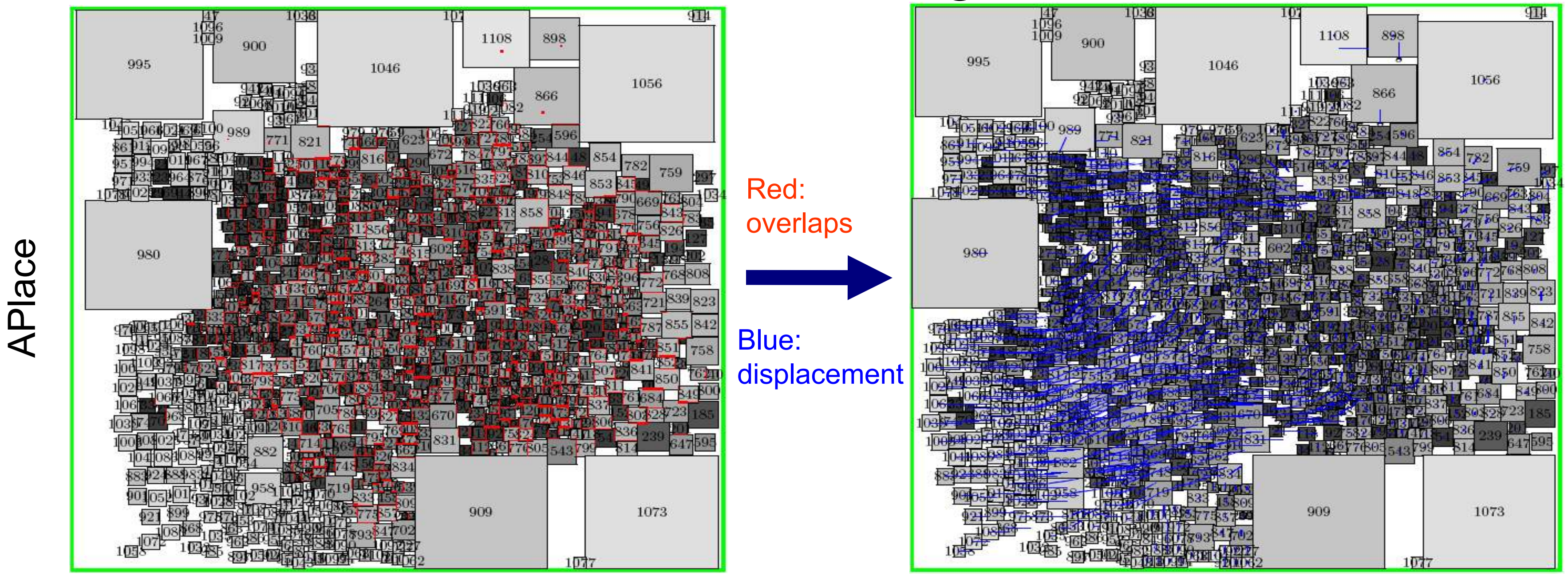
- Legalize top-down
- For each bin:
  - Quickly determine cut-line
  - Check cut-line with single FM pass
  - If cut improved significantly by FM or causes overfull child bin, replace



	APlace 2.04 Global	APlace 2.04 Legalizer			ECO-system		
	Overlap	Runtime	HPWL	WL Increase	Runtime	HPWL	WL Increase
adaptec1	34.74%	**1346**	**83.87**	3.48%	1730	84.84	4.67%
adaptec2	47.25%	2543	101.64	7.88%	**2042**	**99.47**	5.58%
adaptec3	47.12%	11495	231.17	9.49%	**4500**	**227.32**	7.67%
adaptec4	36.78%	15271	206.23	4.56%	**4132**	**203.24**	3.04%
bigblue1	28.53%	2486	**101.96**	1.44%	**1804**	105.14	4.61%
bigblue2	30.15%	14252	159.08	2.96%	**5183**	**156.63**	1.37%
bigblue3	41.06%	38873	414.29	7.50%	**13708**	**388.46**	0.79%
bigblue4	32.01%	56809	884.39	2.24%	**14910**	**881.04**	1.85%
Average				**4.91%**			**3.67%**

DAC`06: floorplan assistant (FLOORIST)

- AI-based floorplan legalizer
- Preliminary results:
  - Removes overlaps quickly, e.g., from APlace placements
  - Mostly preserves initial placement
  - Minimal increase in wirelength



DAC`06: floorplan assistant (FLOORIST)