# Blockage-Avoiding Buffered Clock-Tree Synthesis for Clock Latency-Range and Skew Minimization

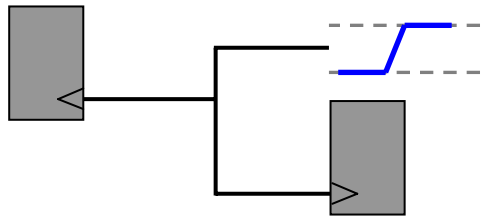
Xin-Wei Shih, Chung-Chun Cheng, Yuan-Kai Ho,
and Yao-Wen Chang

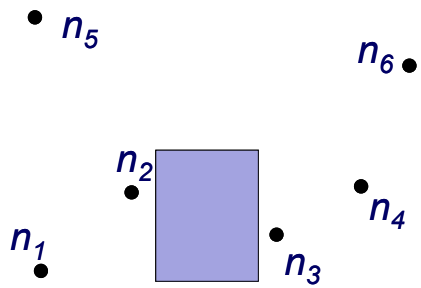
National Taiwan University, Taiwan

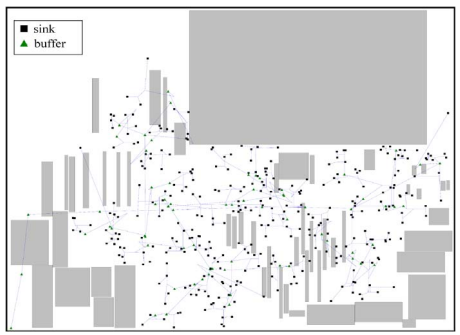
2010.01.20 -

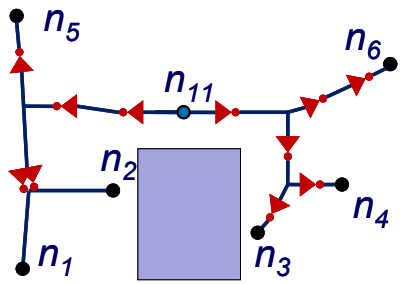
# Outline

- Introduction
- Problem Formulation
- Overall Flow and Algorithms
- Experimental Results
- Conclusions



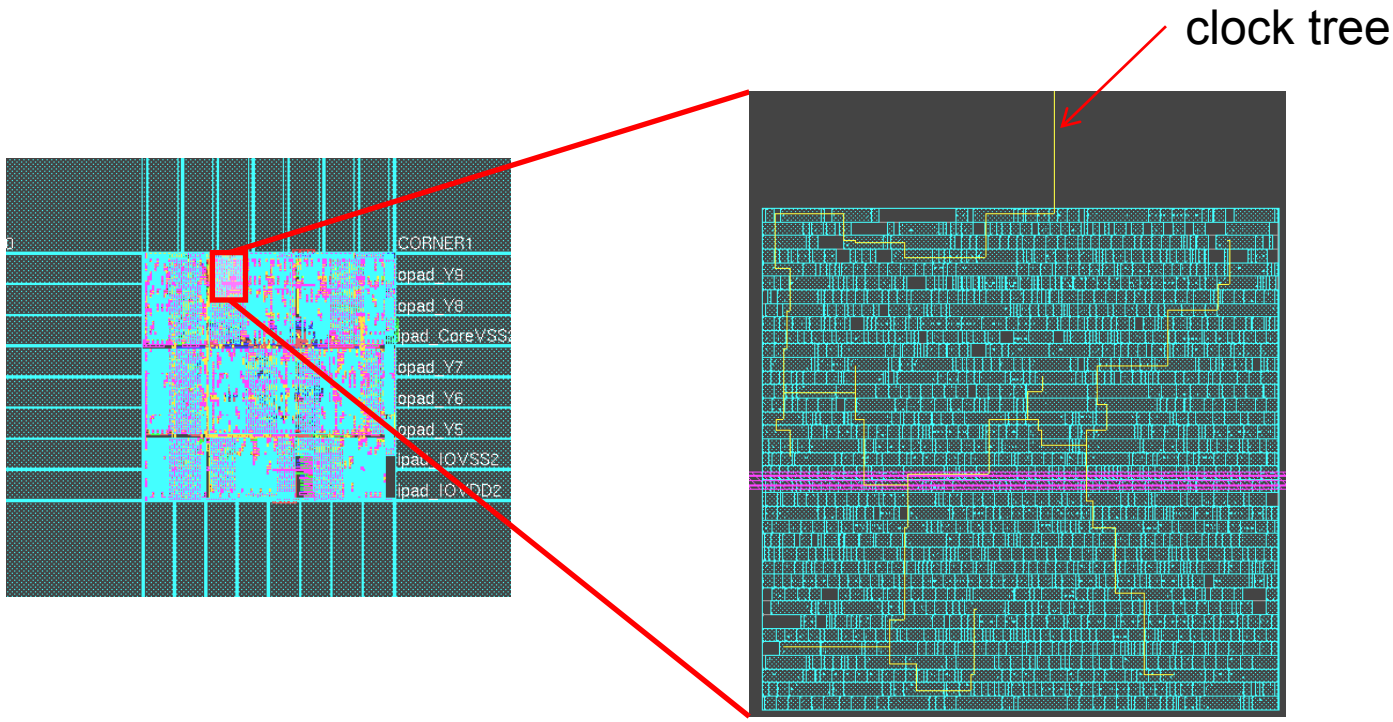


# Introduction

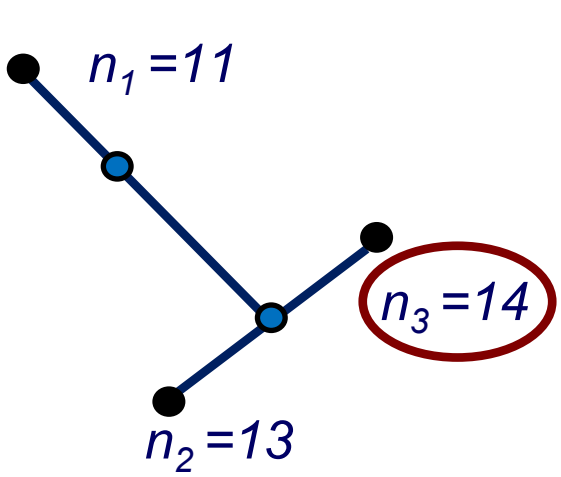
- For high-performance circuit designs, clock tree is the key to synchronize the system
- In advance nanometer technology, there are many new design challenges, such as higher clocking speed, greater design complexity, and larger process variations



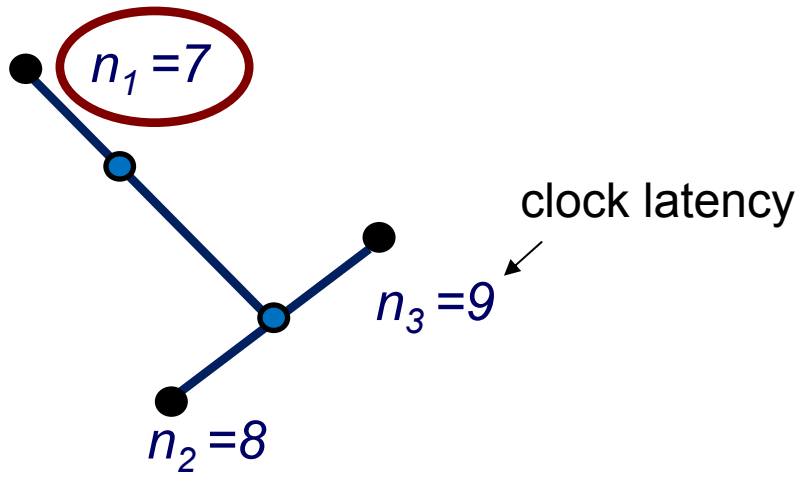

(The pictures are extracted from the Cadence tool, SOC Encounter.)

# Problem Formulation (1/2)

- Clock Latency Range
  - Introduced by the 2009 ACM ISPD Clock Network Contest
  - The latency difference under different supply voltages
  - Example: three clock sinks ($n_1$, $n_2$, $n_3$) and two available supply voltages, high and low
    - Resulting CLR equals 7 ( =max{11, 13, 14} - min{7, 8, 9} = 14 - 7 )




Low supply voltage



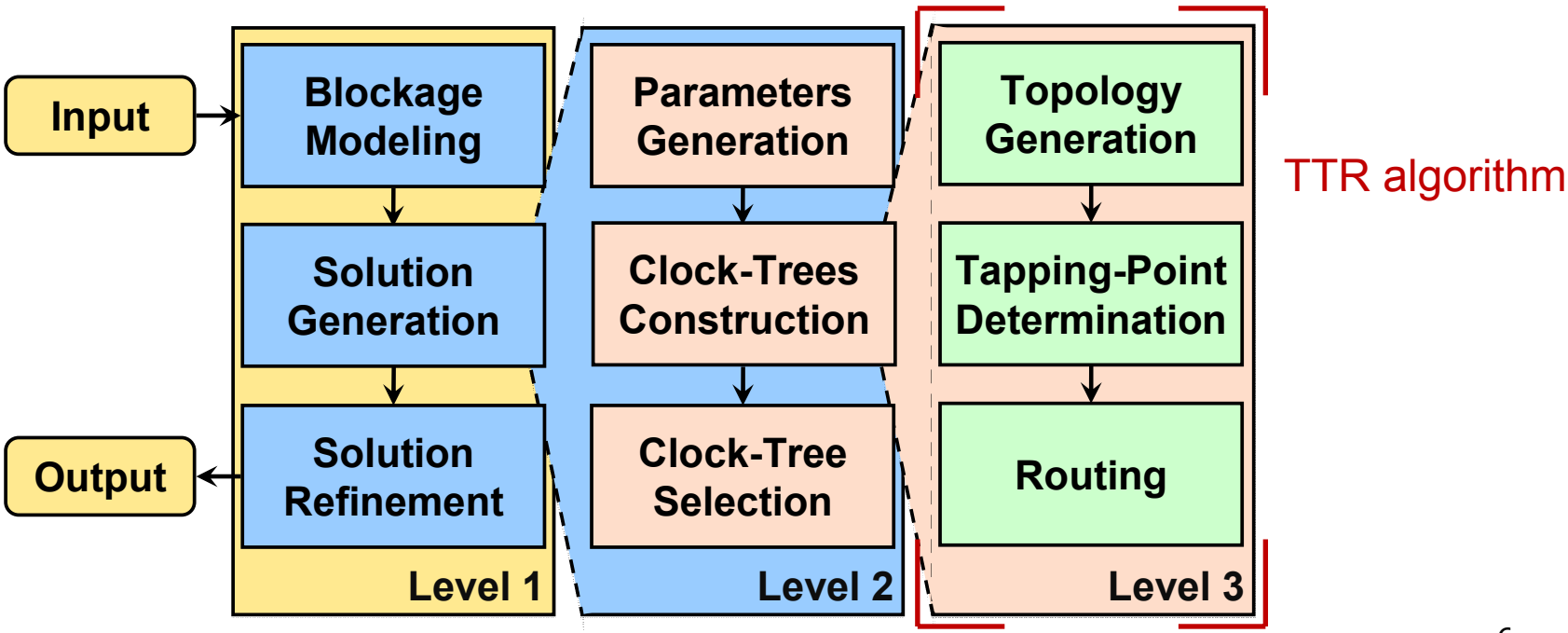

High supply voltage

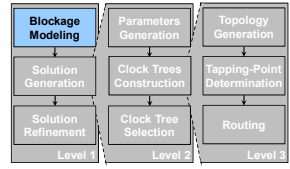
# Problem Formulation (2/2)

- Clock skew should be at least as critical as CLR, if not more important
  - A circuit will not operate at different voltages at the same time!!
- Problem
  - Blockage-Avoiding Buffered Clock-Tree Synthesis
- Instance
  - Given clock sinks, rectangular blockages, a slew-rate constraint, a capacitance limitation, and a library of buffers
- Question
  - Construct a clock tree to minimize its nominal skew (primary objective) and CLR, subject to the constraints:
    - No slew-rate violation, no inverted signal at sinks, no capacitance limitation violation and no overlap between buffers and blockages

# Overall Flow and Algorithms

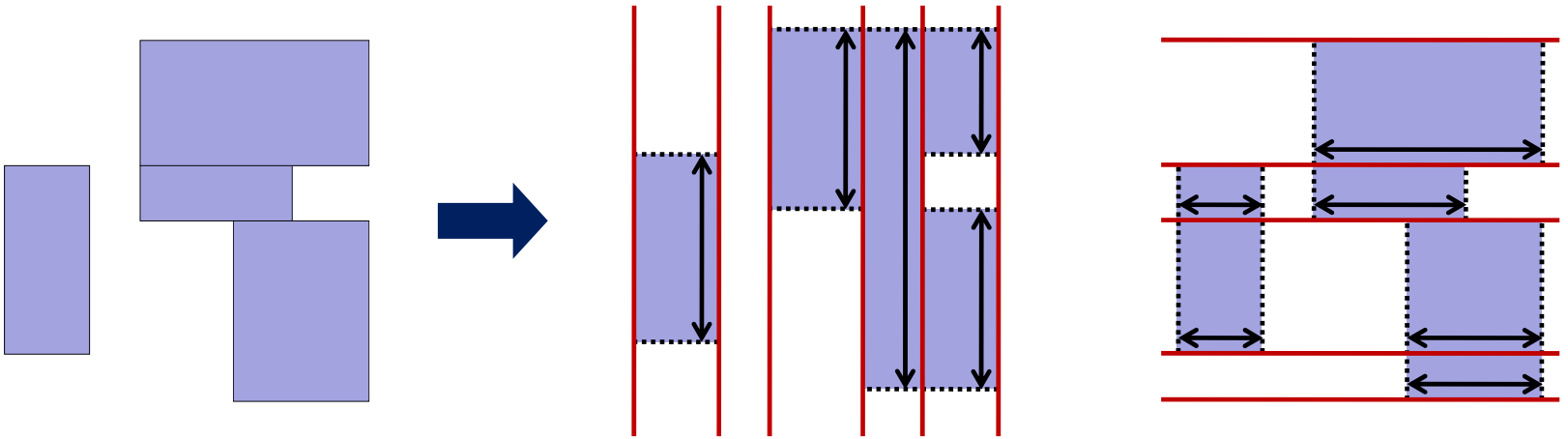
- The procedure mainly optimizes nominal skew
  - The CLR can be reduced implicitly by minimizing nominal skew (there is a positive correlation between the nominal skew and the CLR)
- A three-level framework with the three-stage TTR algorithm is proposed
- Overall flow






# Blockage Modeling

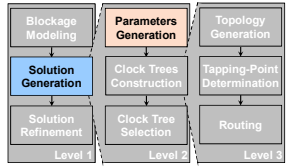
- Blockage Grid
  - Hanan-grid-like data structure
  - Dual data structures in two division orders
    - Vertical-major order
    - Horizontal-major order



Blockage distribution

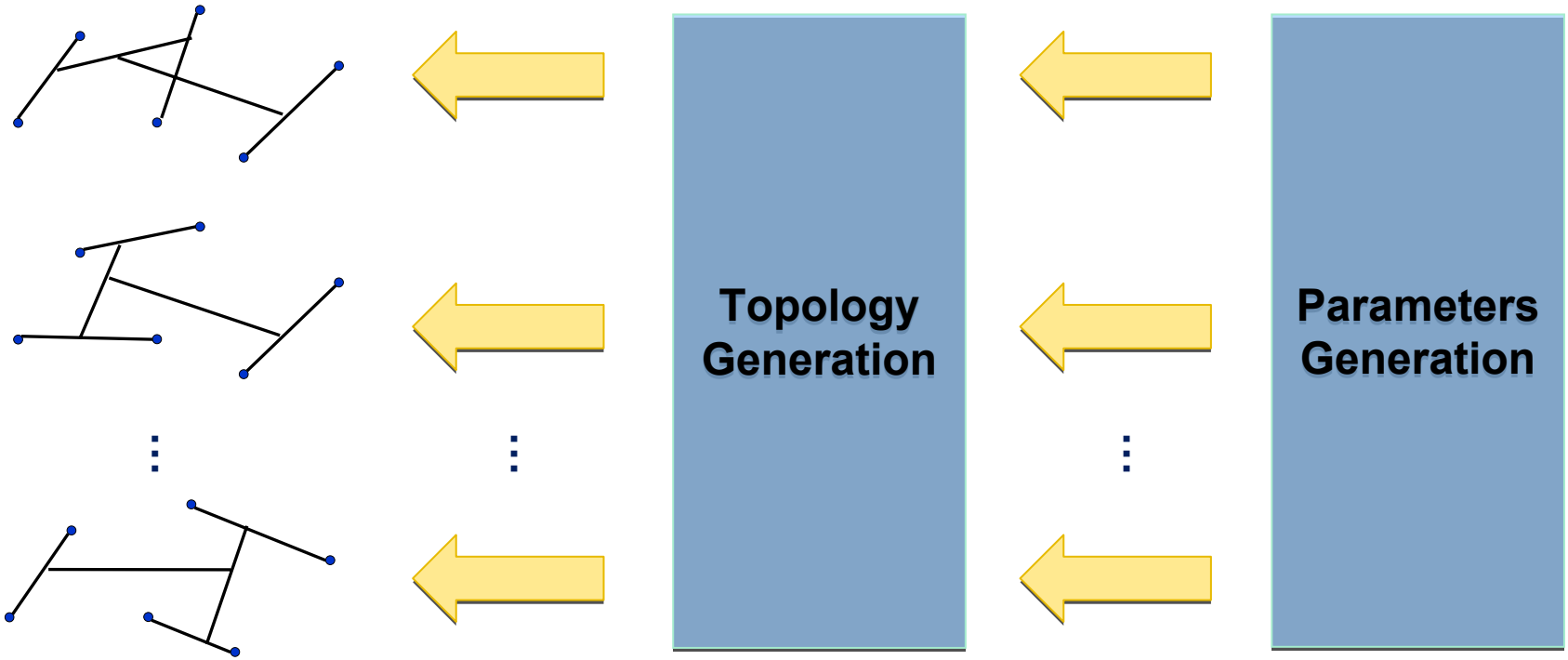
Blockage grid (vertical-major order)

Blockage grid (horizontal-major order)

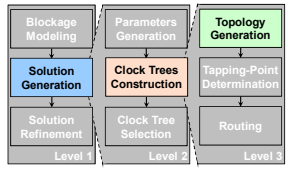



# Parameters Generation

- To generate a set of clock trees in solution generation
  - Make the cost function of topology generation with adjustable parameters for different cost metrics
  - Use dynamic ratio, to control the flow of topology generation to increase the solution space of tree topologies



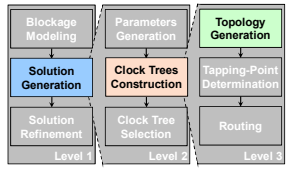
# Topology Generation




- Abstract topology is used for guiding the final clock tree to consider wirelength, loading, and blockage distribution
- Dynamic Nearest-Neighbor Algorithm (DNNA) is proposed
  - Greedy algorithm based on Nearest-Neighbor Algorithm (NNA)[1]
    - Bottom-up method which consists of several passes
    - Each pass merges a fixed percentage of subtree pairs into sub-trees based on their geometry relation, i.e. the cost
  - Dynamic ratio, $\delta$, is used to restrain the merged pairs in each pass within an appropriate cost range
    - Let c* is the minimum cost within all pairs in a certain pass
    - The cost of the last pair picked cannot be larger than $c^* \times \delta$ in the current pass

[1]M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," In Proc. DAC, pp. 612–616, 1993.

# Cost Function

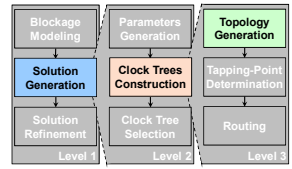


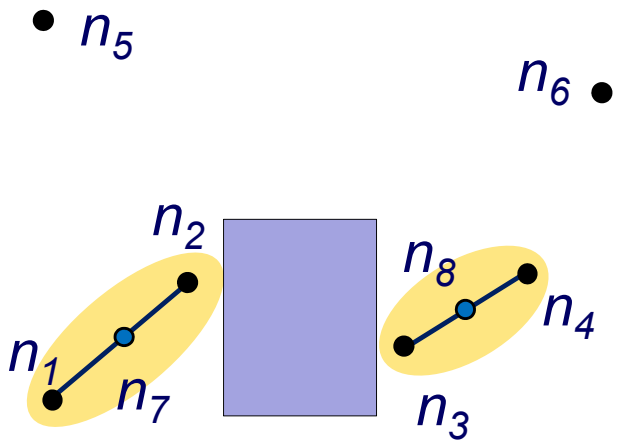

- Four concerns for two nodes $i$ and $j$ in the cost function
  - Distance, $C_d(i, j)$
    - The closer the two nodes, the smaller the cost
  - Blockage Overlap, $C_o(i, j)$
    - Higher cost for a route that potentially goes through blockages
  - Capacitance Unbalance, $C_u(i, j)$
    - The closer downstream capacitances, the smaller the cost
  - Capacitance Priority, $C_p(i, j)$
    - Give the merging priority to the pair with smaller total downstream capacitance can lead to better loading balance
- Overall cost function
  - Combine the four cost and define the overall cost function

$$\Phi(i,j) = c_d(i,j)\left(1+\frac{c_o(i,j)}{\alpha}\right)\left(1+\frac{c_u(i,j)}{\beta}\right)\left(1+\frac{c_p(i,j)}{\gamma}\right)$$

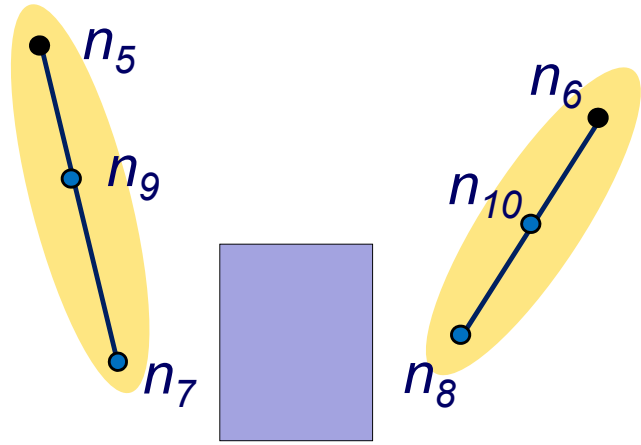
  - $\alpha$, $\beta$, and $\gamma$ are three user-specified parameters to adjust the weights of the costs

# Example of Topology Generation

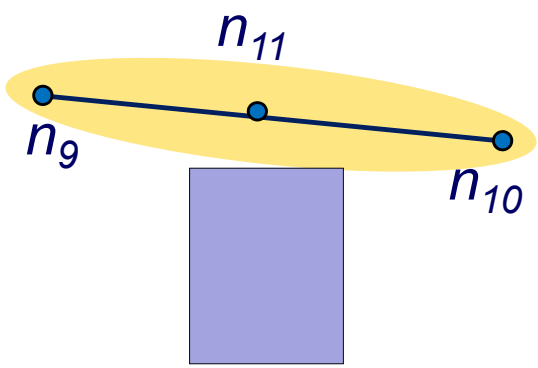


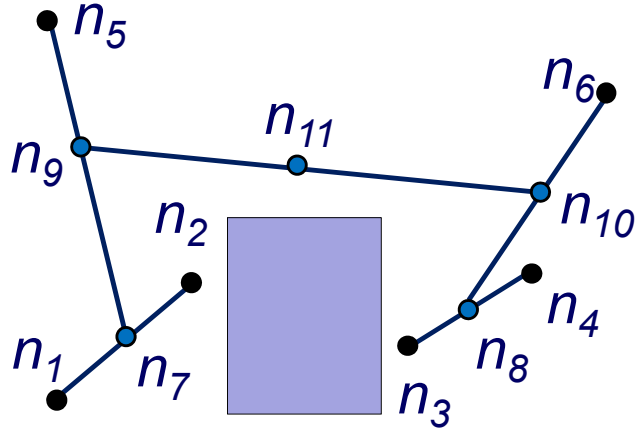

pass 1



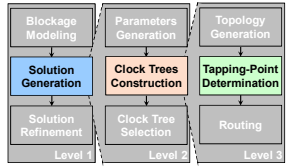

pass 2




pass 3

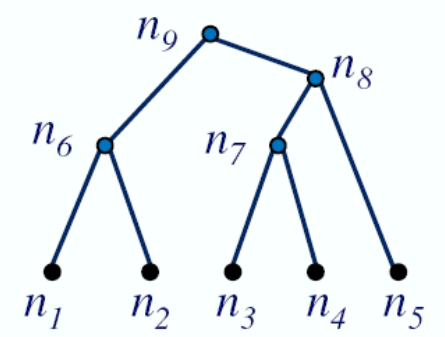



resulting abstract topology

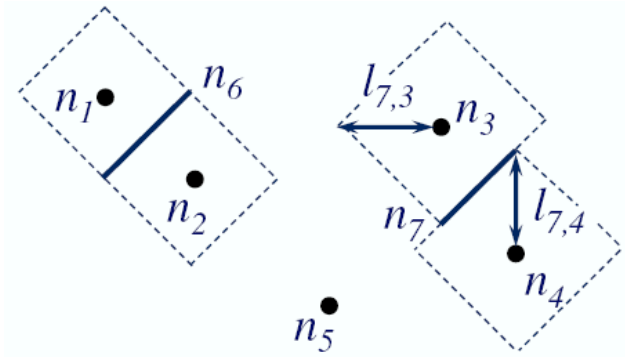



# Tapping-Point Determination

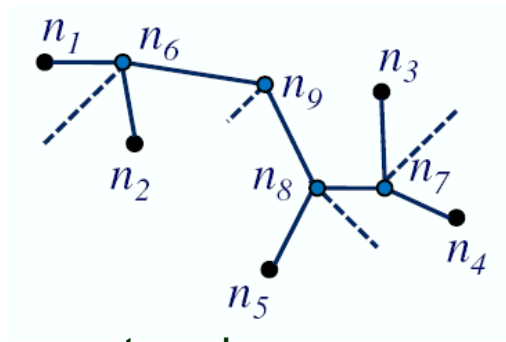
- Main tasks
  - Embed topology nodes as physical tapping points of clock tree
  - Specify the wiring structures of clock branches to be used for routing
- Based on the deferred-merge embedding (DME)[2] algorithm
  - Two phases, the bottom-up merging phase followed by the top-down embedding phase




target abstract topology




bottom-up pass

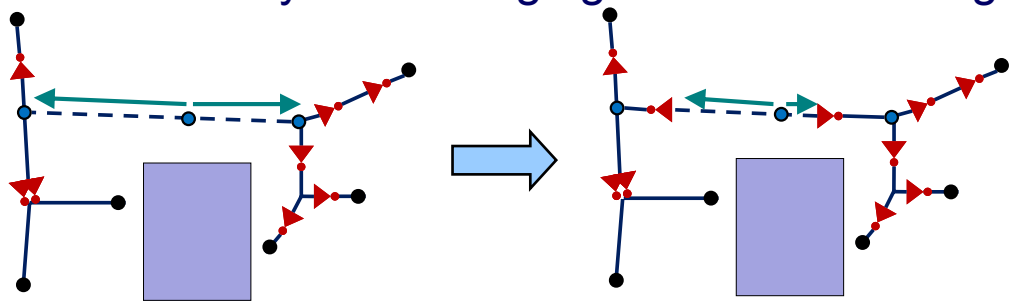



top-down pass

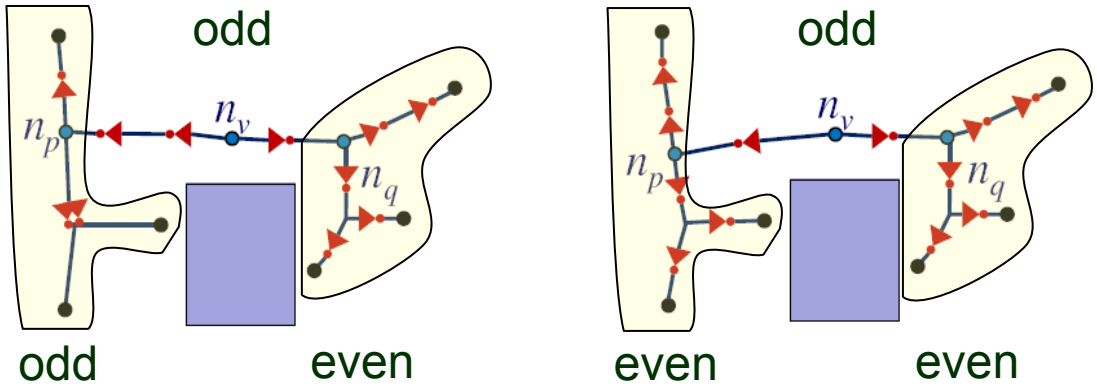
[2]K. D. Boese and A. B. Kahng, "Zero-skew clock routing trees with minimum wirelength," In Proc. ASICON, pp. 17–21, 1992.
T. H. Chao, Y. C. H. Hsu, and J. M. Ho, "Zero skew clock net routing," In Proc. DAC, pp. 518–523, 1992.
M. Edahiro, "Minimum skew and minimum path length routing in VLSI layout design," NEC Research and Development, 32(4), pp. 569–575, 1991.

# Merging Process

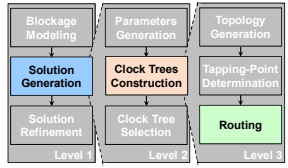
- Buffer insertion
  - Insert buffers recursively if the merging wires are too long



- Signal-parity concern and multiple sub-tree generation
  - Only the inverter buffer is provided in the library
  - Generate candidates for even and odd parities sub-trees for every node



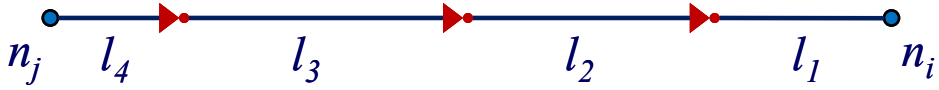
# Routing



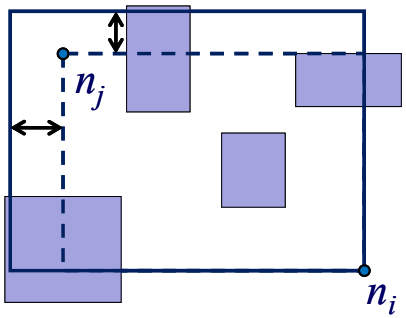

- Perform buffer placement and routing to complete the clock tree
  - A feasible buffer placement is that the distance of buffer-to-buffer are equal or less than the specification in the wiring structure
- Buffer placement
  - Walk-segment
    - A data structure to maintain the potential placement information
    - 45-degree line segment, representing continuous coordinates with equal Manhattan distance to other coordinates
  - Perform the breadth-first-search for buffer placement with the walking idea (place buffers across blockages step by step)
  - The algorithm is called Walk-Segment Breadth First Search (WSBFS)
- Routing
  - The routing is fulfilled by connecting them directly or having a simple detour after the buffer placement

# Walk-Segment Breadth First Search

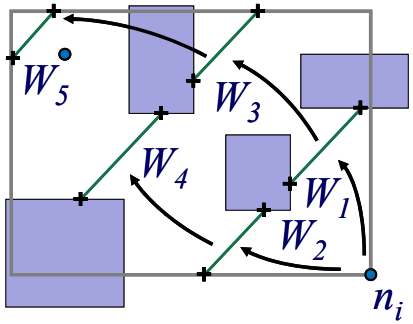
- Three main phases (the same as BFS)
  - Initiation phase
    - Define feasible placement region
    - Make a 0-step walk-segment at the starting point
  - Search phase
    - Keep popping walk-segments from the queue
    - Generate next-step walk-segments, and push them into the queue
  - Backtracking phase
    - Trace a series of walk-segments and place buffers on them



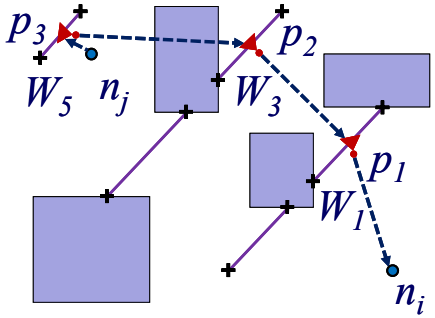

wiring structure



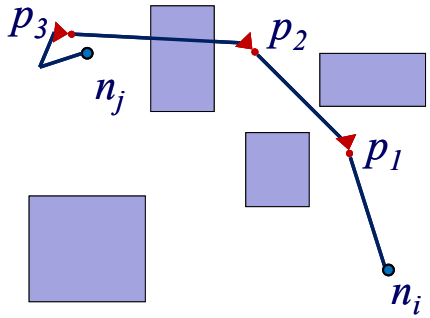

initiation phase




search phase

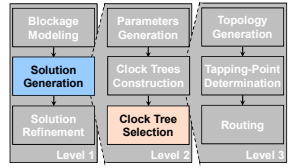



backtracing phase

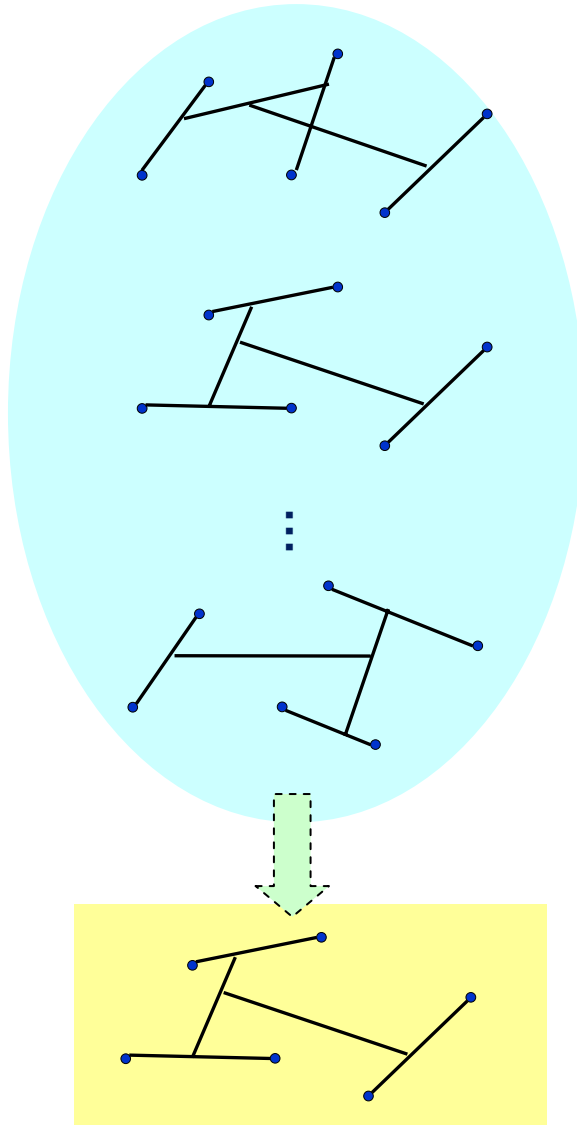



wire routing

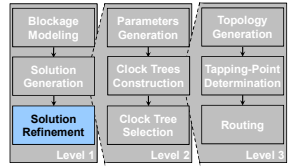
# Clock-Tree Selection



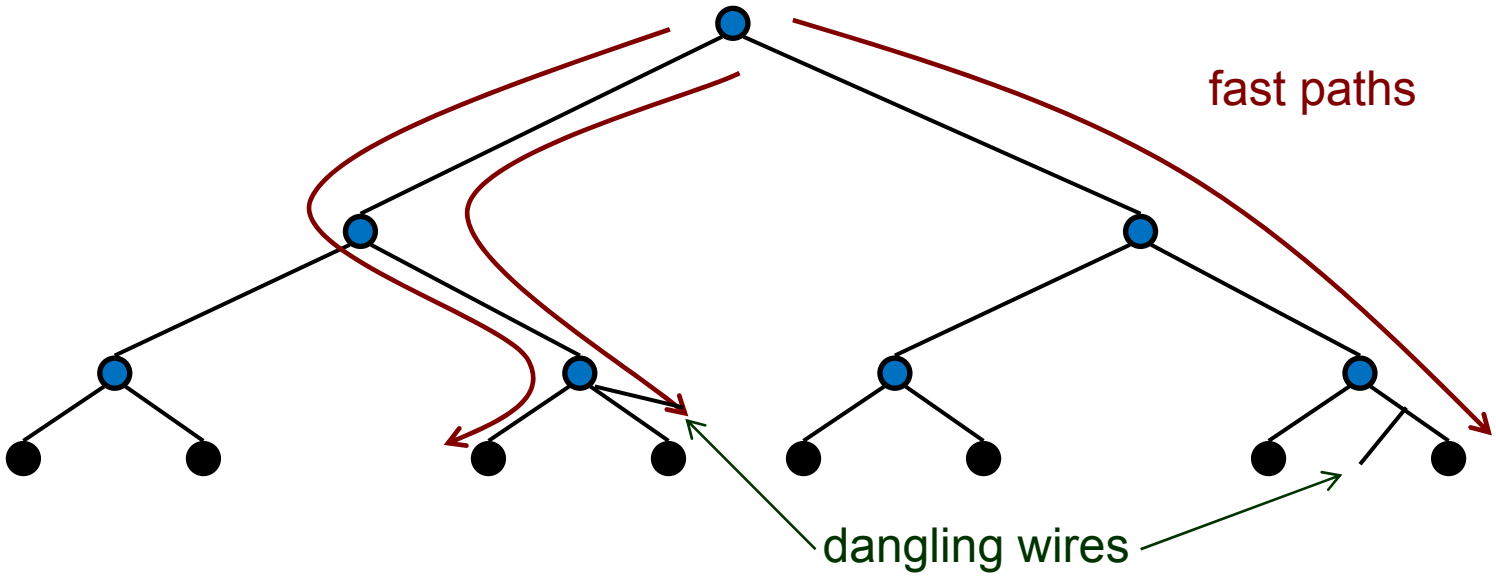

- Three criteria for selection
  - Buffer Level
    - The skew variation due to buffers is directly minimized with less buffer level
  - Signal Latency Time
    - Signal latency time is considered as a representation of path length
    - When the length is long, the variation of voltage causes more latency variation
  - Total Capacitance
    - Less capacitance often implies less snaking, and thus the structure is well-balanced
- Measure these components and select the one with the best average quality



# Solution Refinement




- Further optimize its nominal skew based on SPICE simulation, iteratively
  - Increase the delays of the paths with smaller latencies by adding dangling wires
  - Decrease the delays of the paths with larger latencies by removing dangling wires
- Repeat these process until a desired nominal skew is obtained or some stop criterion is met
- Example (abstract tree)



# Experimental Results (1/4)

- Implemented by C++ programming language on a 2.0 GHz Intel Xeon Linux workstation with 16 GB memory
- Tested the quality of resulting clock trees, nominal skew and CLR, based on the benchmarks used in the ISPD 2009 Clock Network Synthesis Contest
- Compare with the three winners (Team 4, 6 and 17) of the contest

| Benchmark | # Sinks | # Blockages | Chip Size (cm x cm) | | | Cap Limit (fF) |
|---|---|---|---|---|---|---|
| f11 | 121 | 0 | 1.10 | x | 1.10 | 118000 |
| f12 | 117 | 0 | 0.81 | x | 1.26 | 110000 |
| f21 | 117 | 0 | 1.26 | x | 1.17 | 125000 |
| f22 | 91 | 0 | 1.17 | x | 0.49 | 80000 |
| f31 | 273 | 88 | 1.71 | x | 1.71 | 250000 |
| f32 | 190 | 99 | 1.70 | x | 1.70 | 190000 |
| fnb1 | 330 | 53 | 0.26 | x | 0.21 | 42000 |

# Experimental Results (2/4)

- Compared with Teams 4, 6, and 17, the results we achieves averagely 22%, 57% and 777% better nominal skew, and 48%, 67% and 124% better CLR, respectively
- For the runtime comparison, we achieve speedups of more than 2X over Team 4 and more than 3X over Team 17, but is about 8% slower than Team 6

| | | CLR, Skew, Runtime results | | | | Comparison | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Team 4 | Team 6 | Team 17 | Ours | Team 4 | Team 6 | Team 17 | Ours |
| f11 | Skew (ps) | 4.712 | 5.166 | 6.345 | 4.478 | 1.05 | 1.15 | 1.42 | 1.00 |
| | CLR (ps) | 26.710 | 32.291 | 22.306 | 19.707 | 1.36 | 1.64 | 1.13 | 1.00 |
| | Runtime (s) | 14764 | 3892 | 23358 | 4639 | 3.18 | 0.84 | 5.04 | 1.00 |
| f12 | Skew (ps) | 4.763 | 5.883 | 5.440 | 4.088 | 1.17 | 1.44 | 1.33 | 1.00 |
| | CLR (ps) | 25.728 | 32.173 | 22.175 | 17.463 | 1.47 | 1.84 | 1.27 | 1.00 |
| | Runtime (s) | 13934 | 3944 | 14992 | 4231 | 3.29 | 0.93 | 3.54 | 1.00 |
| f21 | Skew (ps) | 5.266 | 6.077 | 3.199 | 3.868 | 1.36 | 1.57 | 0.83 | 1.00 |
| | CLR (ps) | 30.536 | 34.314 | 19.610 | 19.915 | 1.53 | 1.72 | 0.98 | 1.00 |
| | Runtime (s) | 14978 | 4587 | 26420 | 4629 | 3.24 | 0.99 | 5.71 | 1.00 |
| f22 | Skew (ps) | 3.359 | 7.099 | 2.965 | 3.671 | 0.92 | 1.93 | 0.81 | 1.00 |
| | CLR (ps) | 24.505 | 30.448 | 16.376 | 16.474 | 1.49 | 1.85 | 0.99 | 1.00 |
| | Runtime (s) | 7189 | 2005 | 9432 | 3937 | 1.83 | 0.51 | 2.40 | 1.00 |
| f31 | Skew (ps) | 7.564 | 10.867 | 187.881 | 4.762 | 1.59 | 2.28 | 39.45 | 1.00 |
| | CLR (ps) | 45.068 | 51.336 | 211.985 | 31.131 | 1.45 | 1.65 | 6.81 | 1.00 |
| | Runtime (s) | 40088 | 17333 | 3429 | 11112 | 3.61 | 1.56 | 0.31 | 1.00 |
| f32 | Skew (ps) | 5.341 | 6.440 | -- | 4.234 | 1.26 | 1.52 | -- | 1.00 |
| | CLR (ps) | 36.901 | 40.315 | -- | 23.041 | 1.60 | 1.75 | -- | 1.00 |
| | Runtime (s) | 3566 | 10599 | -- | 7293 | 0.49 | 1.45 | -- | 1.00 |
| fnb1 | Skew (ps) | -- | 7.228 | -- | 6.798 | -- | 1.06 | -- | 1.00 |
| | CLR (ps) | -- | 19.835 | -- | 15.731 | -- | 1.26 | -- | 1.00 |
| | Runtime (s) | -- | 477 | -- | 3719 | -- | 0.13 | -- | 1.00 |
| | | | | | average Skew comparison | **1.22** | **1.57** | **8.77** | 1.00 |
| | | | | | average CLR comparison | **1.48** | **1.67** | **2.24** | 1.00 |
| | | | | | average Runtime comparison | **2.61** | **0.92** | **3.40** | 1.00 |

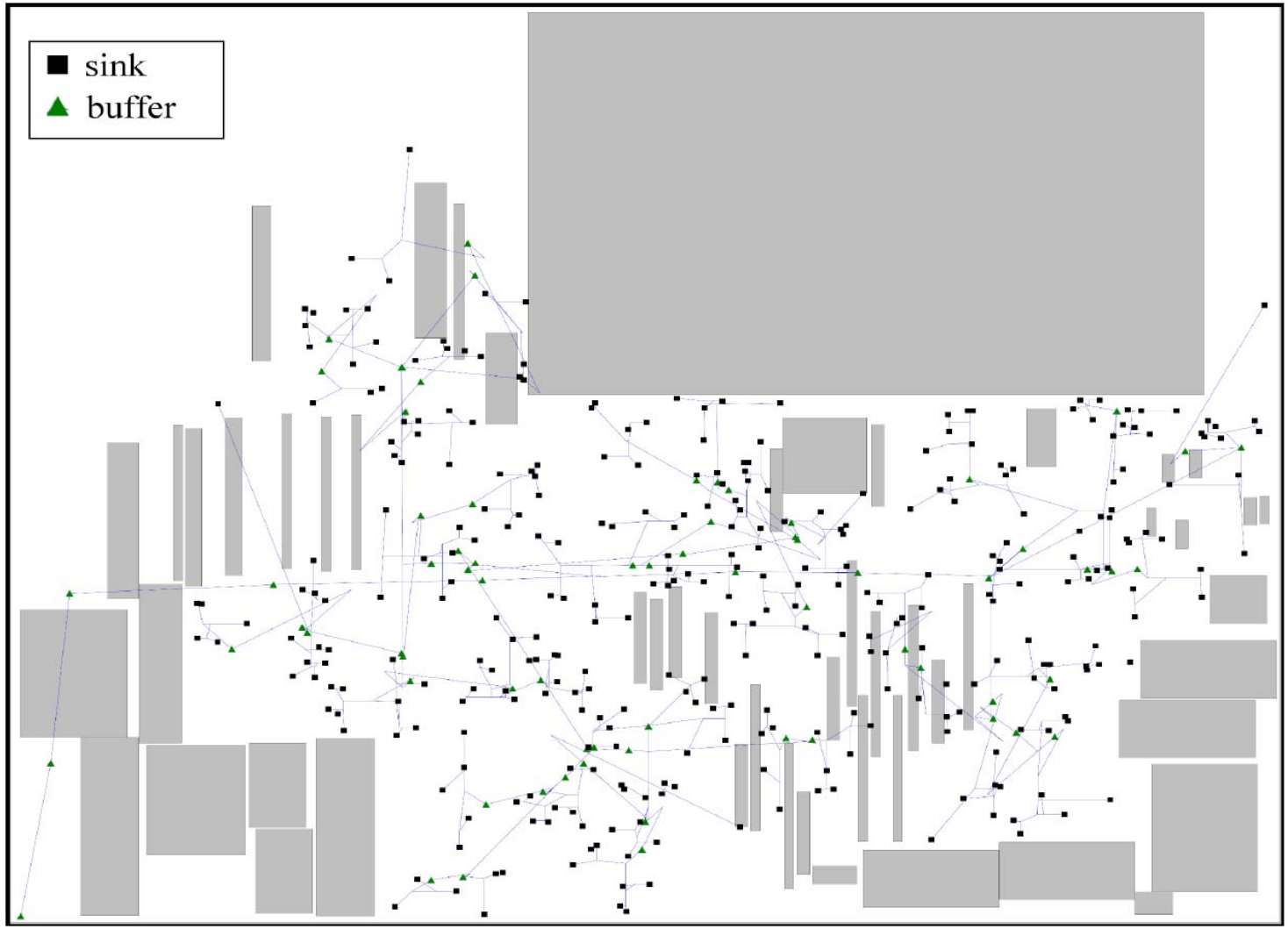
# Experimental Results (3/4)

- DNNA achieves averagely 34% and 9% better nominal skew and CLR respectively
  - The implementation of NNA is the version with setting fixed merging percentage to one-fifth

| | | CLR, Skew results | | Comparison | |
|---|---|---|---|---|---|
| | | NNA | DNNA | NNA | DNNA |
| f11 | Skew (ps) | 5.438 | 4.478 | 1.21 | 1.00 |
| | CLR (ps) | 20.513 | 19.707 | 1.04 | 1.00 |
| f12 | Skew (ps) | 4.645 | 4.088 | 1.14 | 1.00 |
| | CLR (ps) | 20.701 | 17.463 | 1.19 | 1.00 |
| f21 | Skew (ps) | 6.950 | 3.868 | 1.80 | 1.00 |
| | CLR (ps) | 21.645 | 19.915 | 1.09 | 1.00 |
| f22 | Skew (ps) | 7.579 | 3.671 | 2.06 | 1.00 |
| | CLR (ps) | 18.841 | 16.474 | 1.14 | 1.00 |
| f31 | Skew (ps) | 5.385 | 4.762 | 1.13 | 1.00 |
| | CLR (ps) | 32.155 | 31.131 | 1.03 | 1.00 |
| f32 | Skew (ps) | 4.177 | 4.234 | 0.99 | 1.00 |
| | CLR (ps) | 24.964 | 23.041 | 1.08 | 1.00 |
| fnb1 | Skew (ps) | 7.362 | 6.798 | 1.08 | 1.00 |
| | CLR (ps) | 16.828 | 15.731 | 1.07 | 1.00 |
| average Skew comparison | | | | **1.34** | 1.00 |
| average CLR comparison | | | | **1.09** | 1.00 |

# Experimental Results (4/4)

- Sample result of the circuit fnb1



# Conclusions

- We have presented a three-level framework with the three-stage TTR algorithm for blockage-avoiding buffered clock-tree synthesis
- Our algorithms can consider the slew rate, signal parity, process variation, and blockages at the same time
- Experimental results have shown that our approach obtain the *best solution quality* for both nominal skew and CLR minimization, compared to all the participating teams for the 2009 ISPD Clock Network Synthesis Contest

- Future work
  - Apply more accurate timing model to this framework
  - Consider other process variation issues


# Thank You!

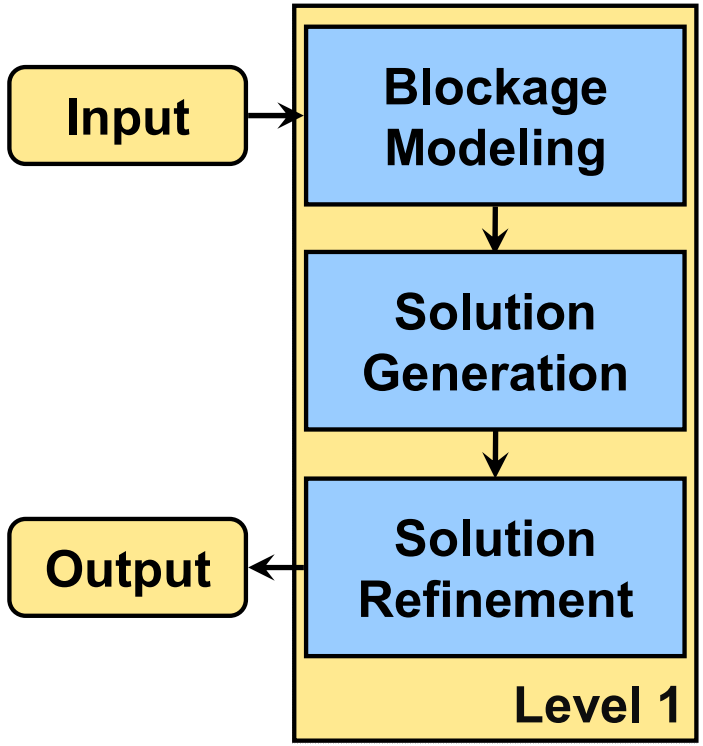
**Xin-Wei Shih**

**raistlin@eda.ee.ntu.edu.tw**


# Back Up

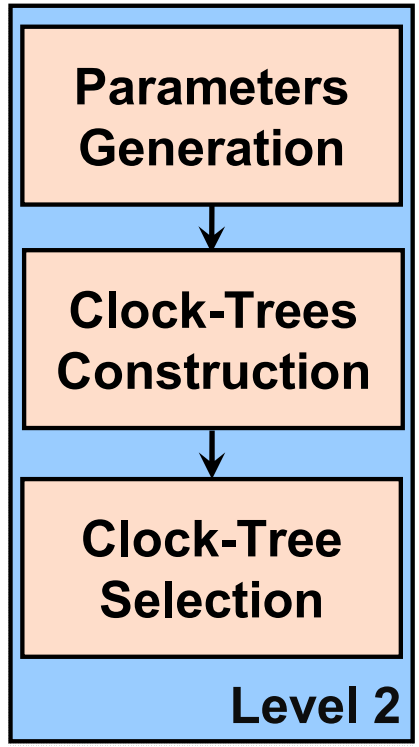
# Level 1 Flow

- Blockage Modeling
  - Construct a data structure to record the given blockage information (e.g., positions, sizes) for further processing
- Solution Generation
  - Find a buffered clock tree with desired CLR and nominal skew
  - Apply timing model to compute the clock delay efficiently
  - Focus more on the CLR optimization by finding a tree with less latency variation
- Solution Refinement
  - Further refine the tree structure to minimize the nominal skew according to SPICE simulation.



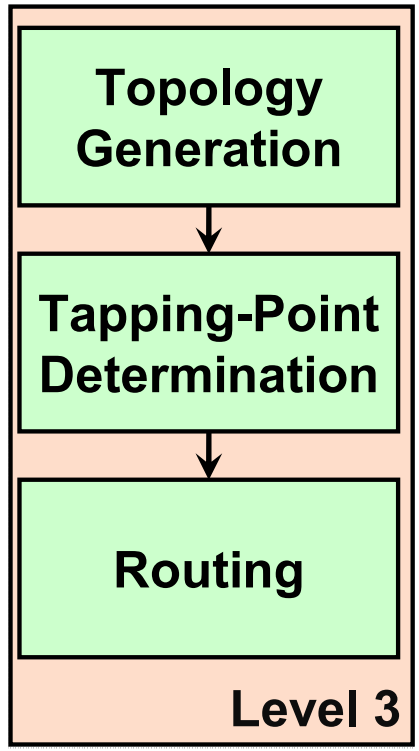
# Level 2 Flow : Solution Generation

- Parameters Generation
  - Determine the values of the parameters in the cost function for tree construction
- Clock-Trees Construction
  - Construct a set of candidate clock trees corresponding to the values of the parameters
- Clock-Tree Selection
  - Evaluate qualities of candidate clock trees
  - Select the best one as the output



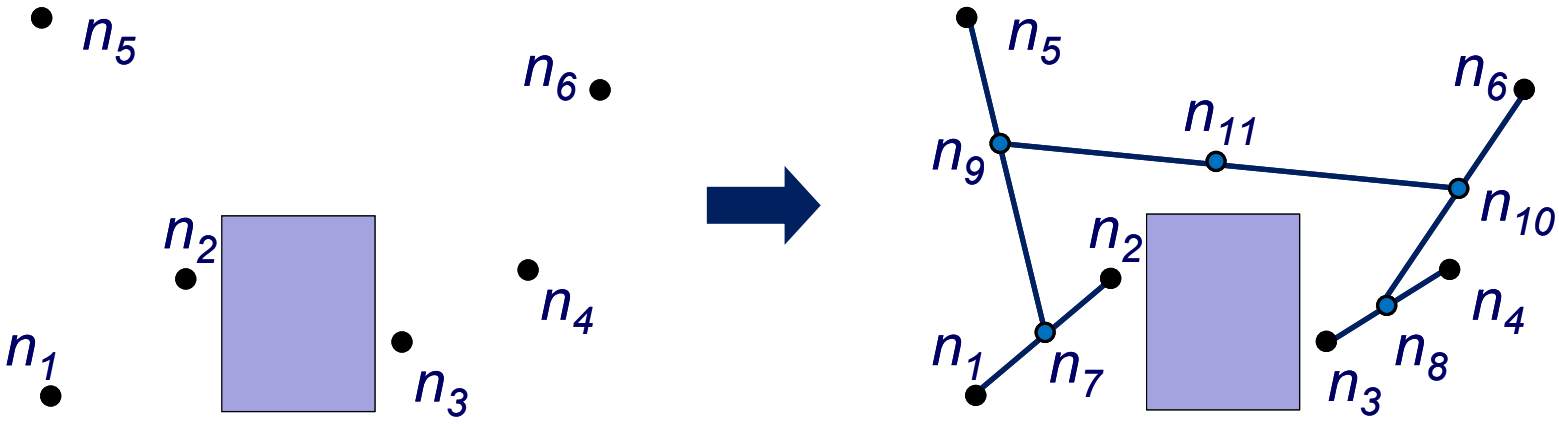
# Level 3 Flow : Clock-Trees Construction

- TTR algorithm
  - Topology generation
  - Tapping-point determination
  - Routing
- Topology Generation
  - Consider wirelength, loading, and blockage distribution
  - Generate an abstract topology with only high-level connection information of sinks
- Tapping-Point Determination
  - Embed internal nodes of the abstract topology (each embedded internal node is a tapping point of corresponding sub-trees)
  - Determine the wiring structure
- Routing
  - Perform buffer placement and routing



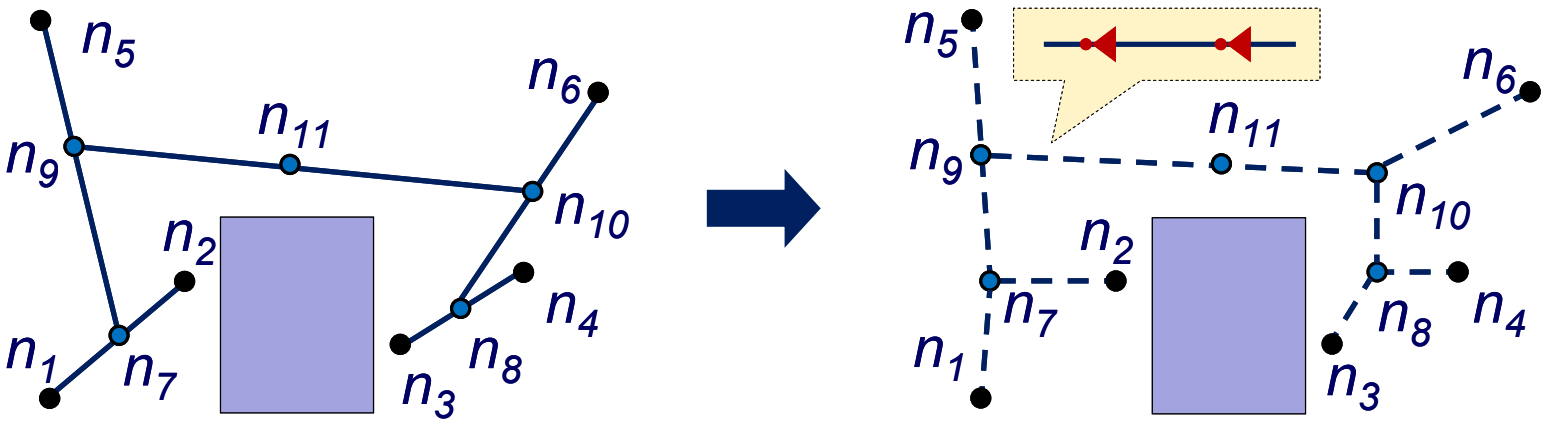
# Clock-Tree Construction (1/3)

- Example (Topology Generation)
  - $n_1$, $n_2$, $n_3$, $n_4$, $n_5$ and $n_6$ are sinks
  - $n_7$, $n_8$, $n_9$, and $n_{10}$ are internal nodes
  - $n_{11}$ is the tree root



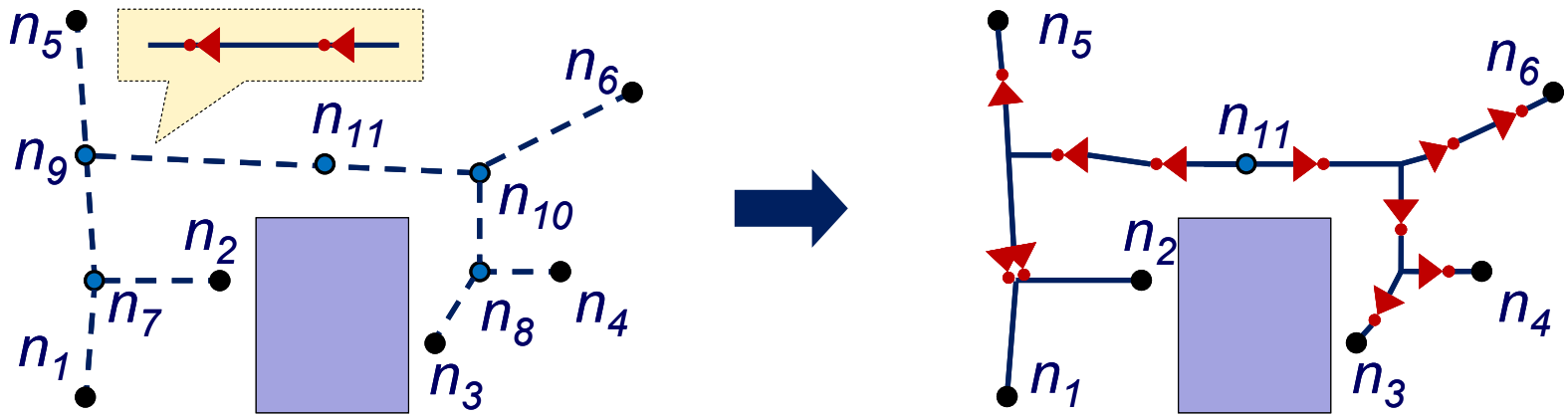
# Clock-Tree Construction (2/3)

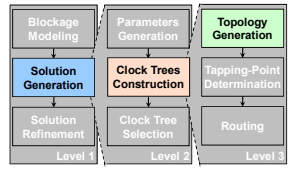
- Example (Tapping-Point Determination)
  - $n_7$, $n_8$, $n_9$, $n_{10}$ and $n_{11}$ are embedded
  - Sample wiring structure between $n_9$ and $n_{11}$ is depicted



# Clock-Tree Construction (3/3)

- Example (Routing)
  - All the buffers are placed and the wires are routed





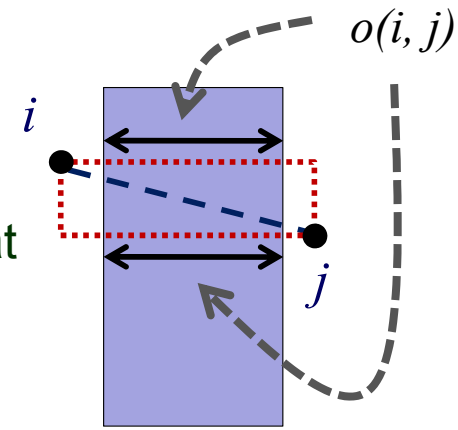

# Cost Function (1/2)

- Cost evaluation is the key to achieve the aforementioned three good properties
- Four concerns for two nodes $i$ and $j$ in the cost function:
  - (1) Distance
    - The closer the two nodes, the smaller the cost
    - $c_d(i,j) = |x_i - x_j| + |y_i - y_j|$ ,

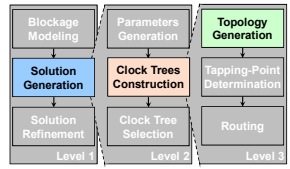
      nodes $i$ and $j$ at the coordinates $(x_i, y_i)$ and $(x_j, y_j)$
  - (2) Blockage Overlap
    - Higher cost for a route that potentially goes through blockages
    - $c_o(i,j) = \dfrac{o(i,j)}{2c_d(i,j)}$ ,

      $rct(i, j)$ is the rectangle formed with nodes $i$ and $j$ at the two opposite corners

      $o(i, j)$ is the overlap perimeter of $rct(i, j)$ and blockages



# Cost Function (2/2)




- (3) Capacitance Unbalance
  - The closer downstream capacitances, the smaller the cost
  - $$c_u(i,j) = \frac{|C_i^D - C_j^D|}{\max\{C_i^D, C_j^D\}} \quad ,$$

    $C^D_i$ and $C^D_j$ are the respective downstream capacitances of nodes $i$ and $j$
- (4) Capacitance Priority
  - Give the merging priority to the pair with smaller total downstream capacitance can lead to better loading balance
  - $$c_p(i,j) = \frac{C_i^D + C_j^D}{2\max\{C_m^D | \forall m\}}$$

$\alpha$, $\beta$, and $\gamma$ are three user-specified parameters to adjust the weights of the costs

- Overall cost function
  - Combine the four cost and define the overall cost function

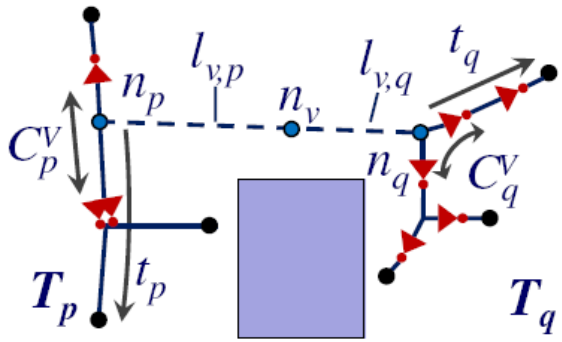
$$\Phi(i,j) = c_d(i,j)\left(1 + \frac{c_o(i,j)}{\alpha}\right)\left(1 + \frac{c_u(i,j)}{\beta}\right)\left(1 + \frac{c_p(i,j)}{\gamma}\right)$$

# Merging Equation

- To obtain the required length, we use a merging equation (bottom-up pass)
  - For the merging of two sub-trees $T_p$ and $T_q$:

$$\frac{1}{2}rcl_{v,p}^2 + rl_{v,p}C_p^V + t_p = \frac{1}{2}rcl_{v,q}^2 + rl_{v,q}C_q^V + t_q$$

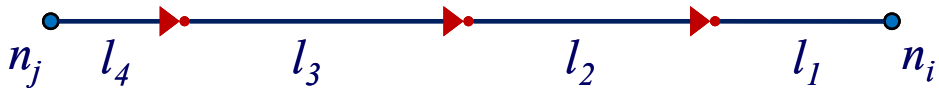
    - $L$ is the shared wire length for merging branches $B_p$ and $B_q$
    - $l_{v,p}$ and $l_{v,q}$ are required lengths for branches $B_p$ and $B_q$ ($l_{v,p}$+$l_{v,q}$= $L$)
    - $r$ and $c$ are the unit-length wire resistance and capacitance
    - $C^V_p$ and $C^V_q$ are the visible capacitances of $T_p$ and $T_q$
    - $t_p$ and $t_q$ are the delays from node $n_p$ and $n_q$ to their sinks




Example of merging

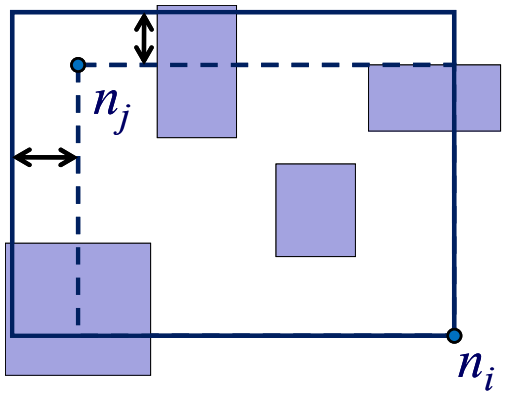
# Routing Example (1/2)

- Example



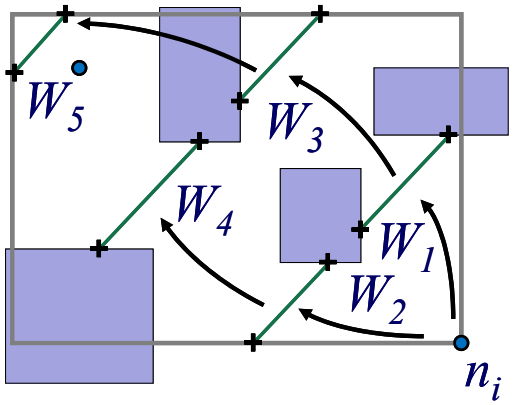

**Wiring structure**

The given wiring structure for two tapping points $n_i$ and $n_j$




**WSBFS initiation phase**

*The bounding box has horizontal and vertical extensions, instead of purely using $n_i$ and $n_j$ as the opposite corners of it



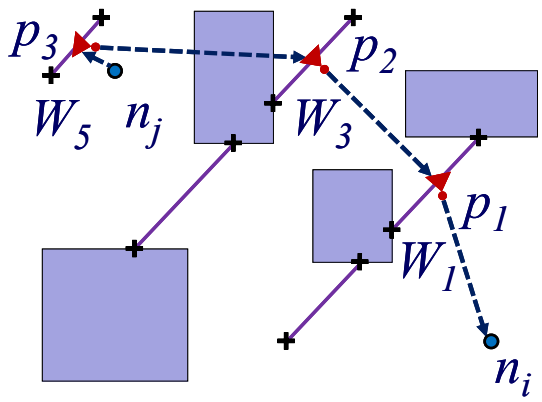

**WSBFS search phase**

*i-step $W_i$ is generated by $W_{i-1}$, every position in $W_i$ can find a position in $W_{i-1}$ with the distance equal to $l_i$

*The generation must avoid blockages, and thus there are more than one walk-segment separated by blockages for a single step

# Routing Example (2/2)

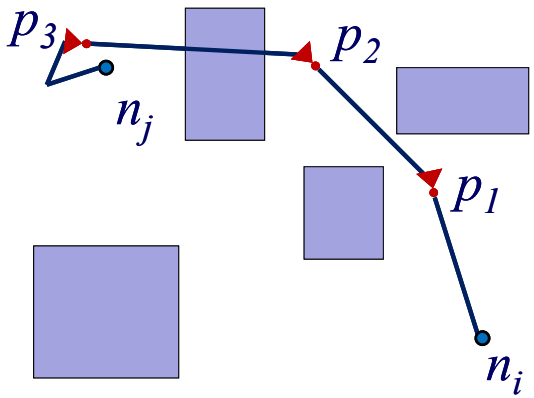
- Example (cont'd)




## WSBFS backtracking phase

*Following the step-generation relation, we trace back the walk-segment path, which forms a buffer placement order

*Along the path, every previously decided position can have the exact position in the walk-segment with the shortest distance




## Wire routing

*Because the buffer placement is constructed by matching the wirelength exactly, the routing could be established straightforward

*The search phase does not consider the last wirelength, and thus the routing between the end point and the last buffer may has a detour

# Experimental Results

- The runtime is dominated by the ngspice simulation which consumes more than 97% of the total runtime for all benchmark circuits

| Benchmark | Total time (s) | ngspice simulation | | Our construction | |
|---|---|---|---|---|---|
| | | time (s) | percentage | time (s) | percentage |
| f11 | 4639 | 4626 | 99.72% | 13 | **0.28%** |
| f12 | 4231 | 4220 | 99.74% | 11 | **0.26%** |
| f21 | 4629 | 4616 | 99.72% | 13 | **0.28%** |
| f22 | 3937 | 3928 | 99.77% | 9 | **0.23%** |
| f31 | 11112 | 11014 | 99.12% | 98 | **0.88%** |
| f32 | 7293 | 7262 | 99.57% | 31 | **0.43%** |
| fnb1 | 3719 | 3640 | 97.88% | 79 | **2.12%** |