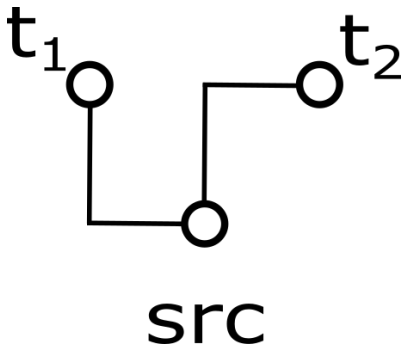
PathSteiner: Improving PathFinder with Quasi-Optimal Steiner-Tree Initialization

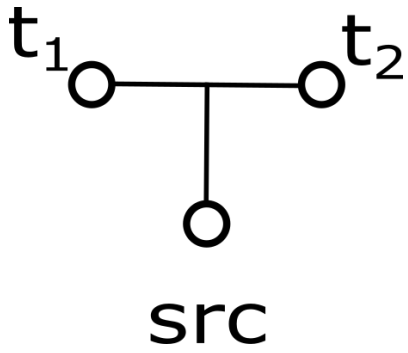
Shashwat Shrivastava, Luka Kurešević, Alexandros Poupakis, Chirag Ravishankar,
Dinesh Gaitonde, Stefan Nikolić, and Mirjana Stojilović

FCCM 2026, Atlanta, Georgia, USA

Route Tree Matters [FCCM'25]



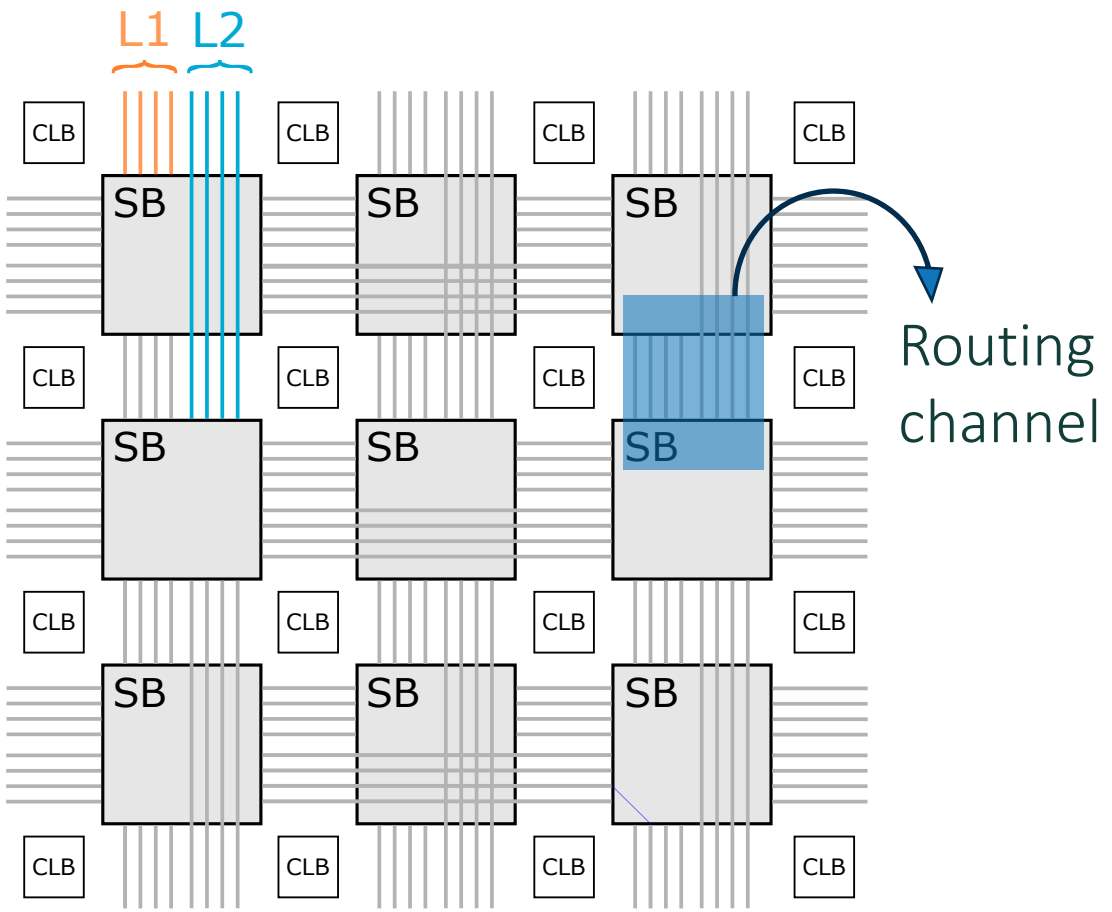
Constructing them → NP-hard



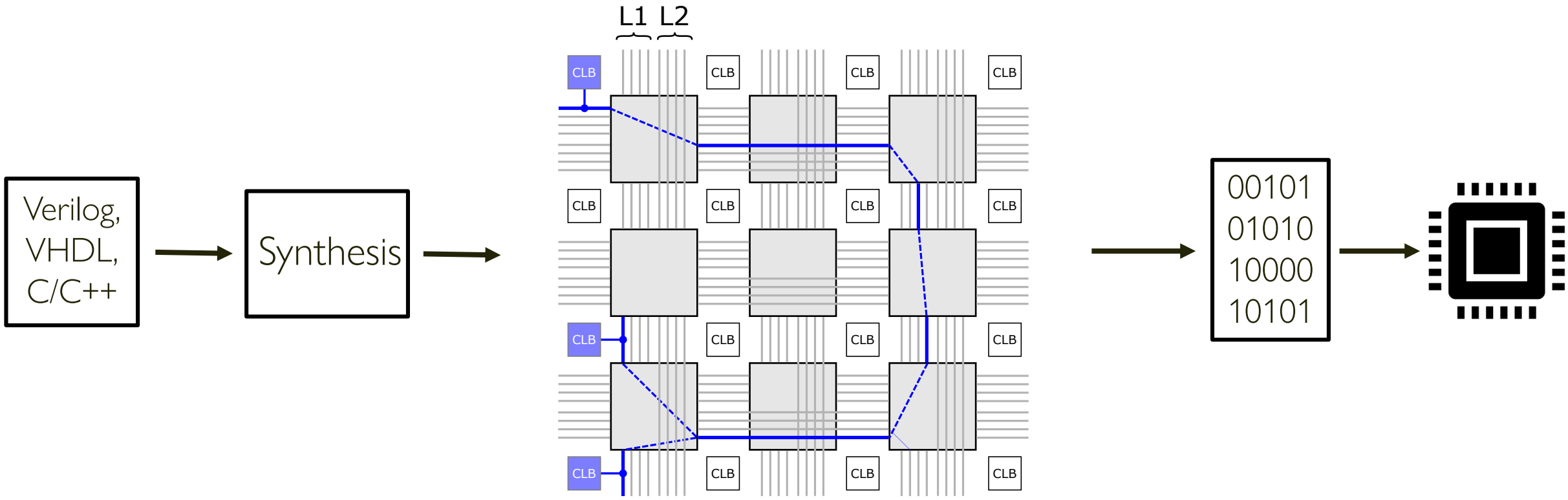
Shift from monolithic (single-stage) to multi-stage routing

FPGA Architecture 101

- Configurable logic blocks (CLB)
  - LUTs, FFs
- Switch blocks (SB)
  - Routing multiplexers
- Wires of various length
  - 1, 2, 4, 12

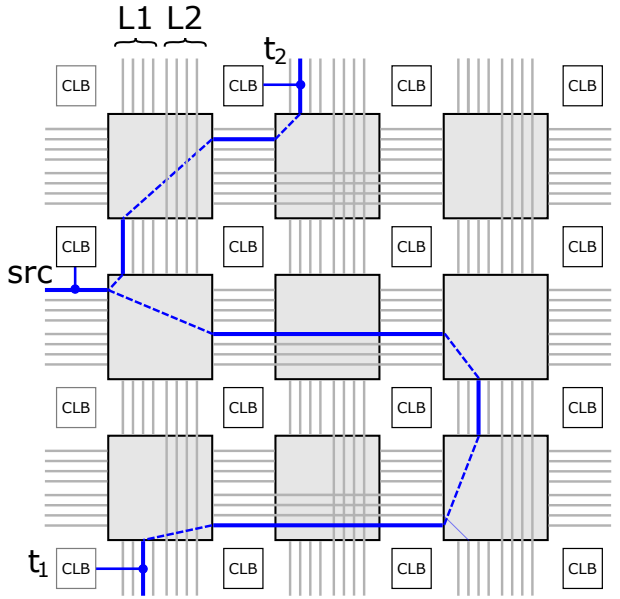


FPGA Compilation 101



Routing 101

- A netlist, placement, and a routing graph
- Find approximate shortest disjoint paths b/w each net's source and its sinks
- Trees corresponding to different signals must be disjoint



A net's routing tree

Routing Algorithm: PathFinder

1

```
for iteration in range(max_iterations):
        for net in nets:
                route_tree = route_net(net)
                increase_cost_of_used_nodes()
        update_cost_of_congested_nodes()
        if no congestion:
                return
```

Routing Algorithm: PathFinder

```
for iteration in range(max_iterations):
        for net in nets:
                route_tree = route_net(net)
                increase_cost_of_used_nodes()
        update_cost_of_congested_nodes()
        if no congestion:
                return
```

1

2

Routing Algorithm: PathFinder

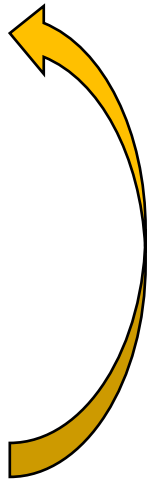
1.
2.
3.

```
for iteration in range(max_iterations):
        for net in nets:
                route_tree = route_net(net)
                increase_cost_of_used_nodes()
        update_cost_of_congested_nodes()
        if no congestion:
                return
```

Routing Algorithm: PathFinder

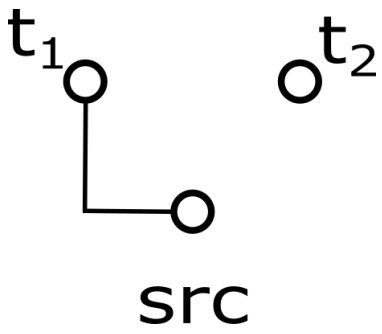
1
2
3

```
for iteration in range(max_iterations):
        for net in nets:
                route_tree = route_net(net)
                increase_cost_of_used_nodes()
        update_cost_of_congested_nodes()
        if no congestion:
                return
```

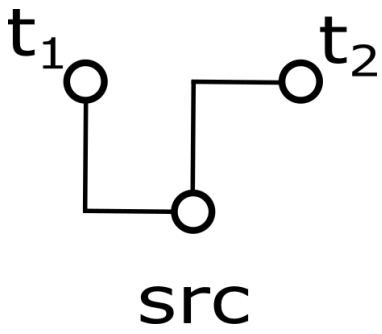


PathFinder: Route Tree Construction

- PathFinder constructs route trees sink-by-sink
- For each sink, finds path with lowest cost
- For a net with two sinks (pins), routes t_1 followed by t_2



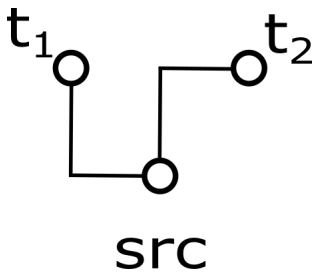
Routing t_1



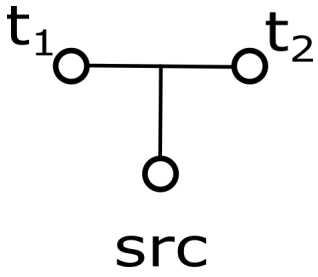
Routing t_2

But, PathFinder Builds Suboptimal Trees

- Suboptimal route trees created by PathFinder [FCCM'25]



PathFinder's route tree
(Wirelength = 4)



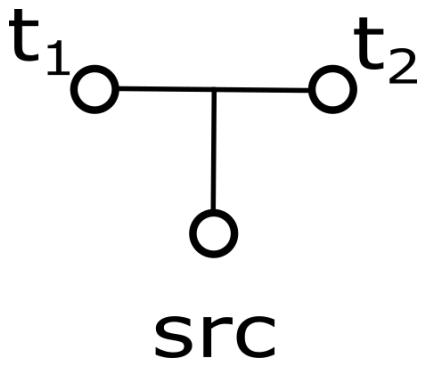
Optimal route tree
(Wirelength = 3)

- Smaller trees → fewer resources → less congestion → faster routing

Routing trees are critical for runtime and wirelength

Core Challenge

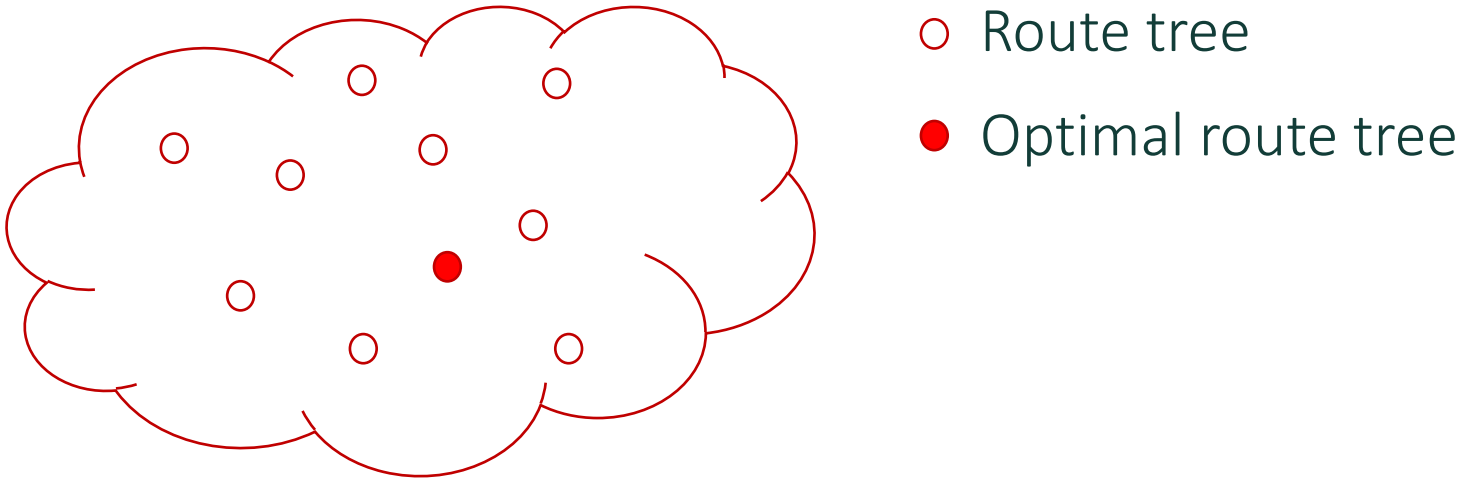
- Constructing optimal trees → Steiner minimal tree problem
- Given: A set of terminals (pins) and an FPGA routing graph
- Find: Minimum-cost tree connecting all terminals



No polynomial-time algorithm to find an optimal solution

Requires Heuristics

- Exact optimization is not practical → we rely on heuristics
- Route-tree solution space → find a near-optimal route tree



Route-tree solution space

Routing becomes a search for a near-optimal tree in a solution space

Outline

- Background
- Sink Shuffling
- PathSteiner
- Takeaways

Route All Sink Permutations

- Ideally, for each net, route all permutations of sink orders
  - Each sink order can be routed in parallel
- But, permutations grow super-exponentially for high-fanout nets
  - Not scalable
- Route only a subset of the possible sink orders
  - Randomly select multiple sink orders

Sink Shuffling [FCCM'25]

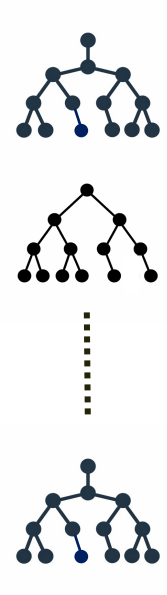
1. Generate multiple random sink orders

$t_1, t_2, t_3, \ldots t_n$

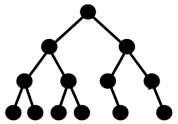
$t_5, t_1, t_n, \ldots t_3$

$t_n, t_1, t_3, \ldots t_5$

2. Route each sink order

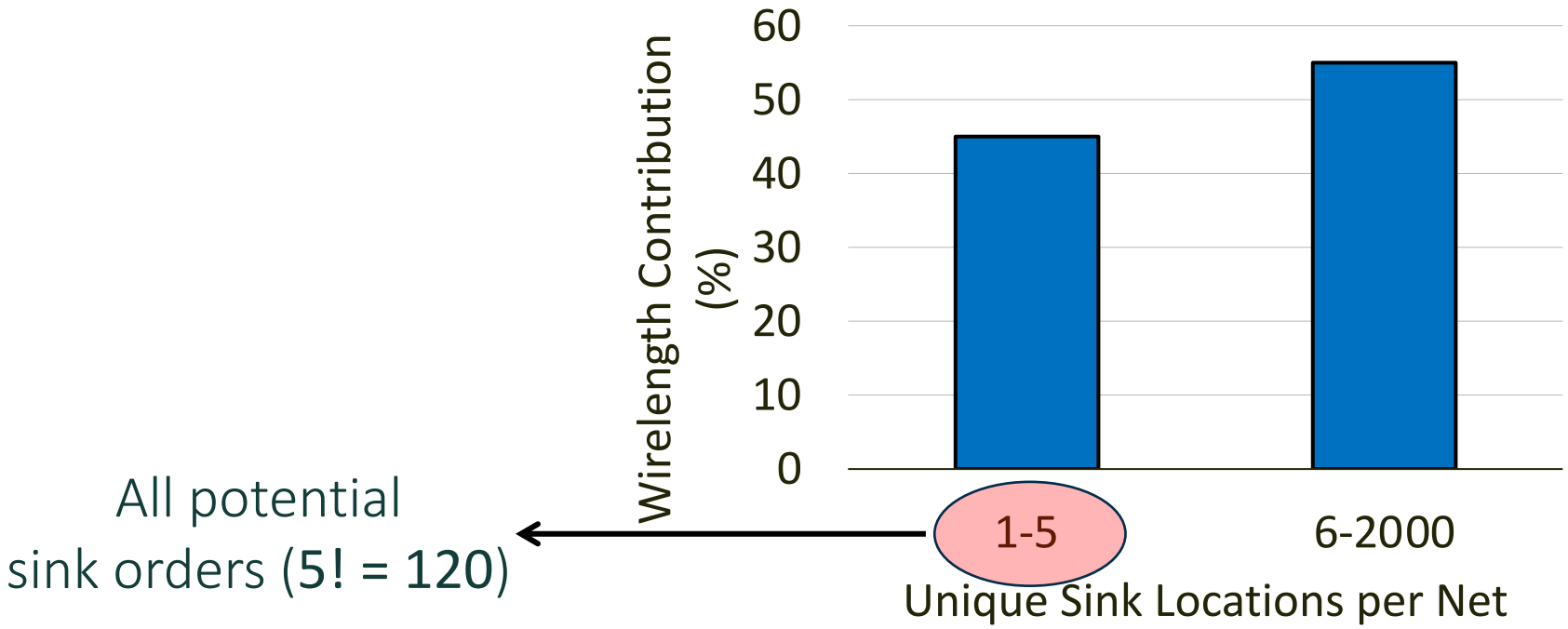


3. Select best tree



Limitation #1

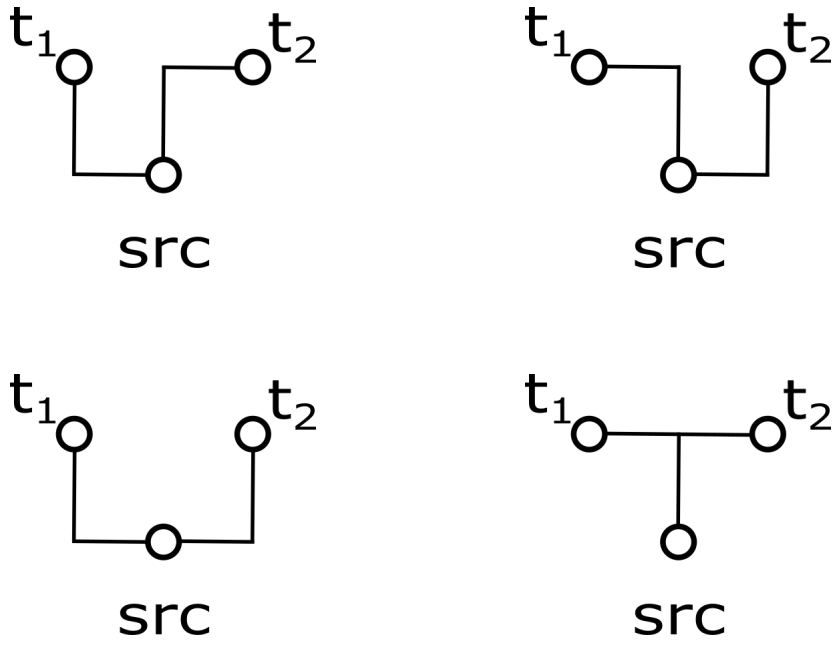
- Random sink-order shuffling provides limited coverage of all possible orders
- Even with exhaustive shuffling for nets with sinks in at most five unique tiles



<50% of final routed wirelength targeted for optimization at best

Limitation #2

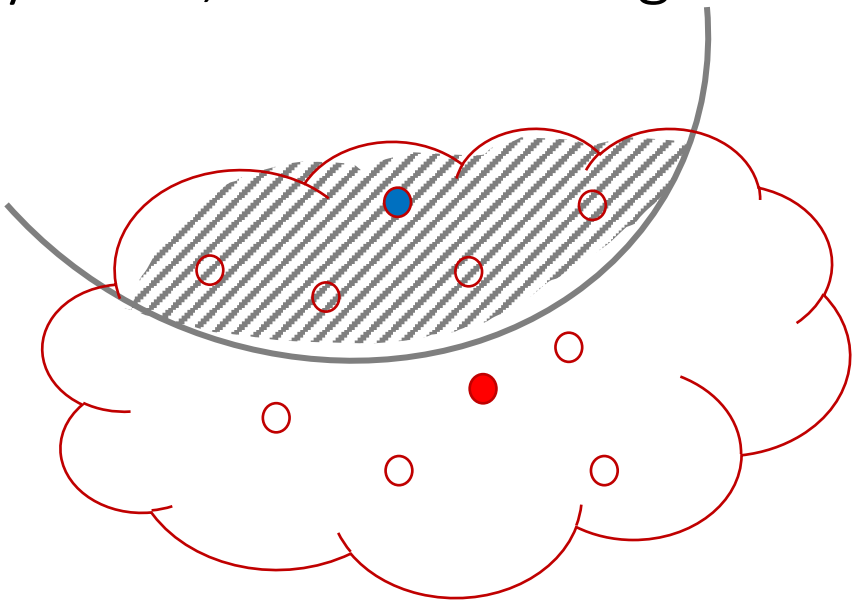
All trees are equally likely with equal minimum path length



Exploring all sink orders still does not guarantee an optimal tree

Sink Shuffling: Summary

- Limited, largely random exploration of the solution space
- Generates many trees, but does not guide the search



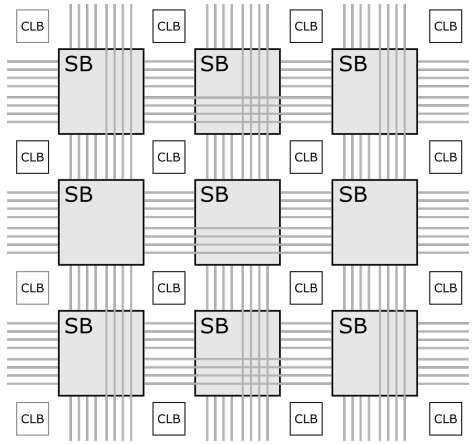
Route-tree solution space

- Route tree
- Optimal route tree
- Best amongst sink shuffles

Instead of randomly searching for a good tree, what if we start from one and try to preserve it?

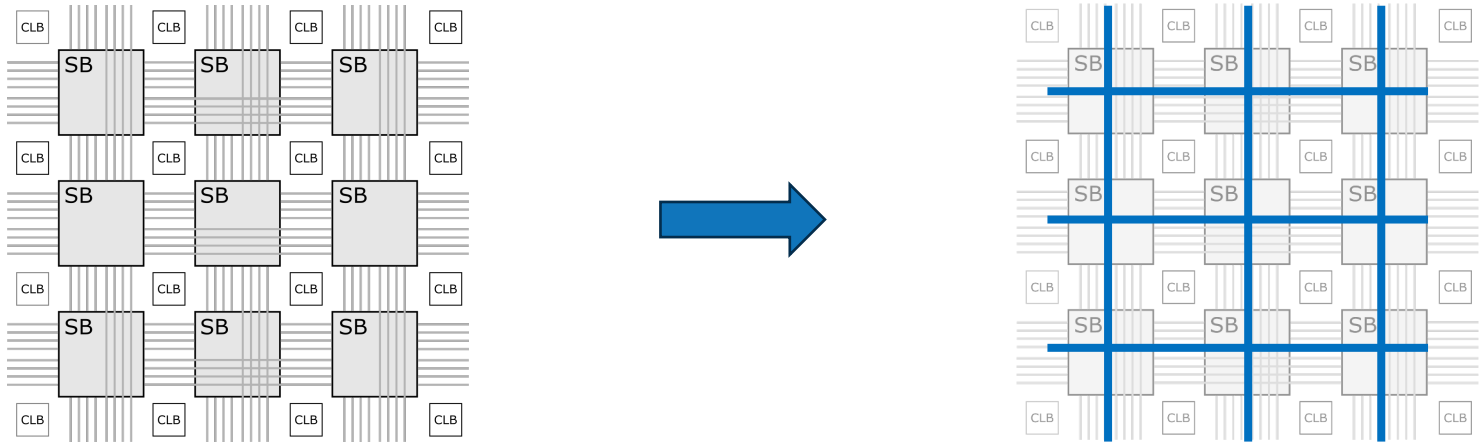
Initialize PathFinder?

- But, we cannot compute optimal trees on the FPGA routing graph → $O(c^n)$



Initialize PathFinder with Coarse Trees

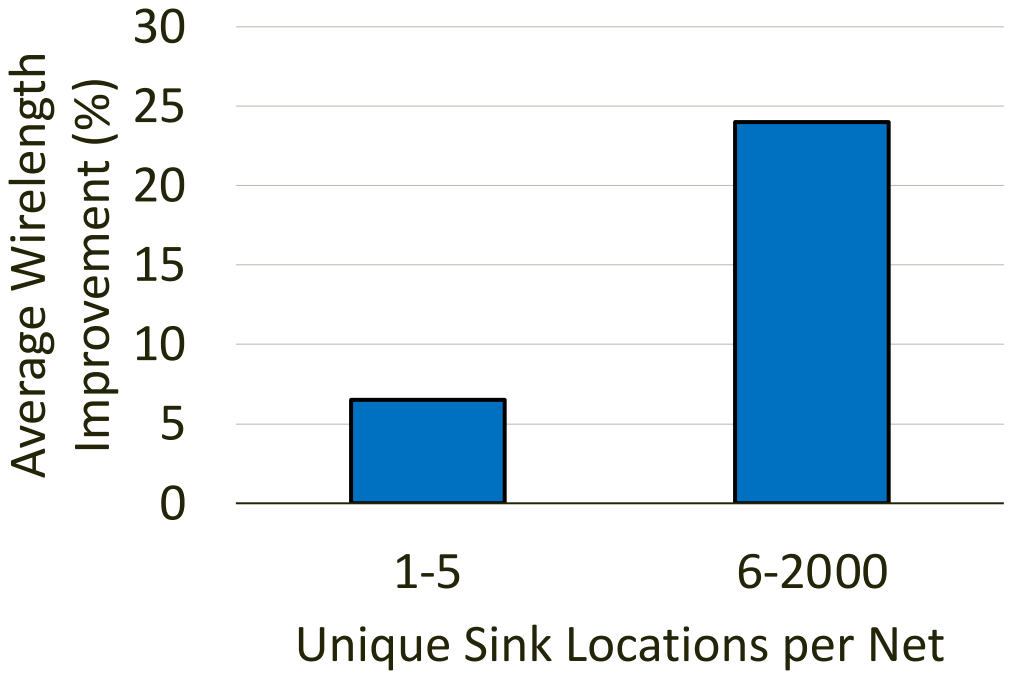
- But, we can on a rectilinear (Manhattan) grid



- Algorithms like FLUTE compute optimal/quasi-optimal trees → $O(n)$
  - Optimal for up to 9 terminals, quasi-optimal beyond

Ignores congestion but gives near-optimal coarse tree

FLUTE vs PathFinder Initialization

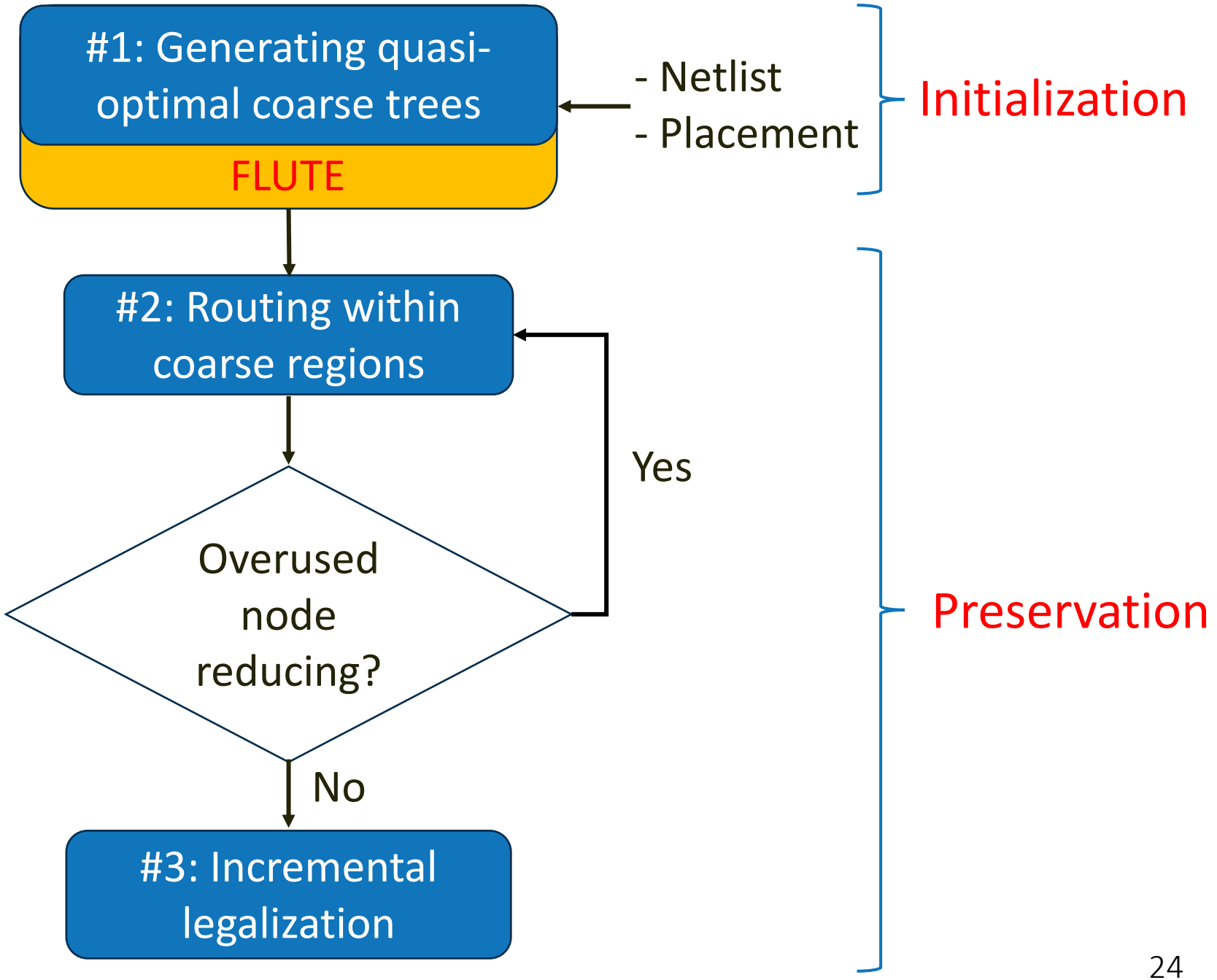


15% better wirelength with FLUTE compared to PathFinder's first iteration

Outline

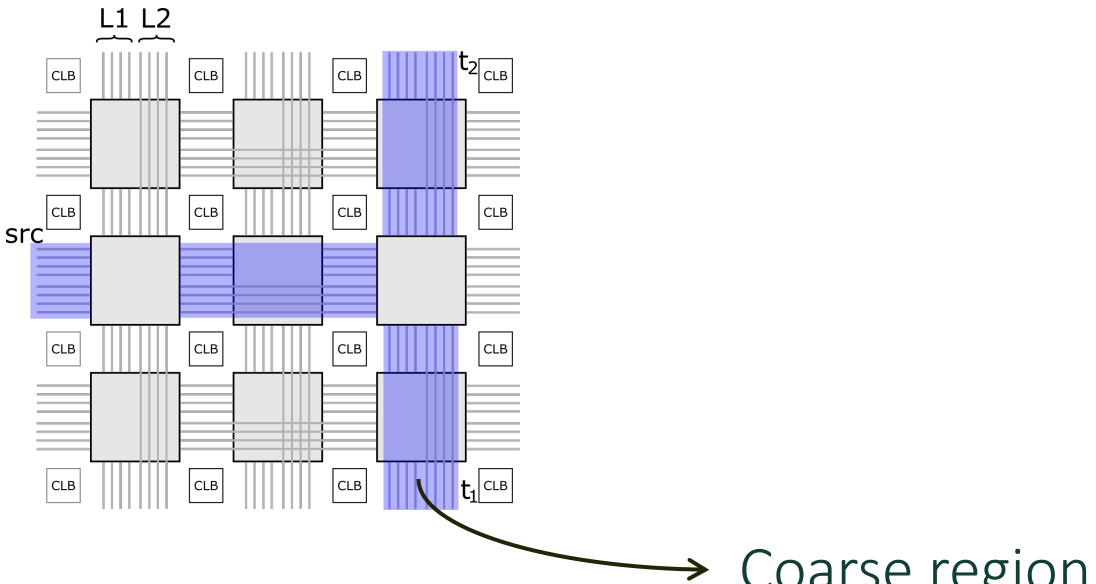
- Background
- Sink Shuffling
- PathSteiner
- Takeaways

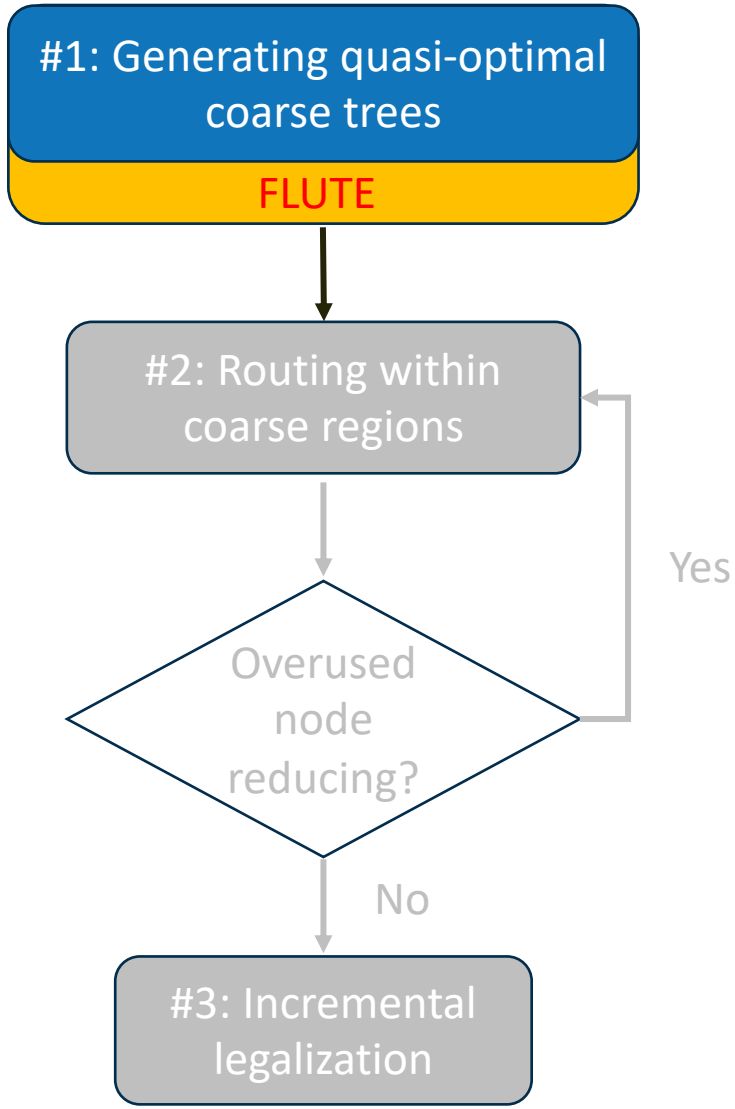
PathSteiner



Generating Quasi-Optimal Coarse Trees

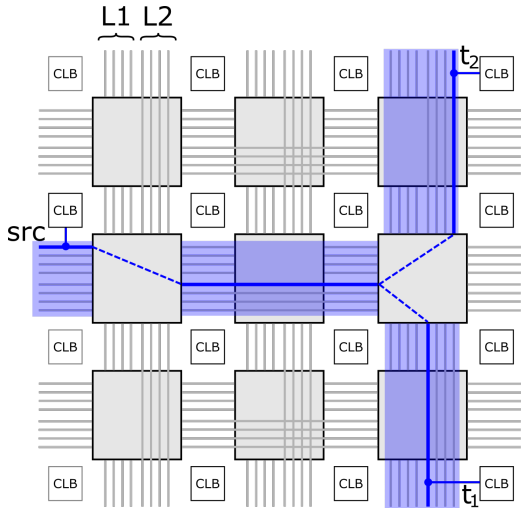
- Use FLUTE to generate quasi-optimal coarse trees
- Congestion-oblivious coarse trees



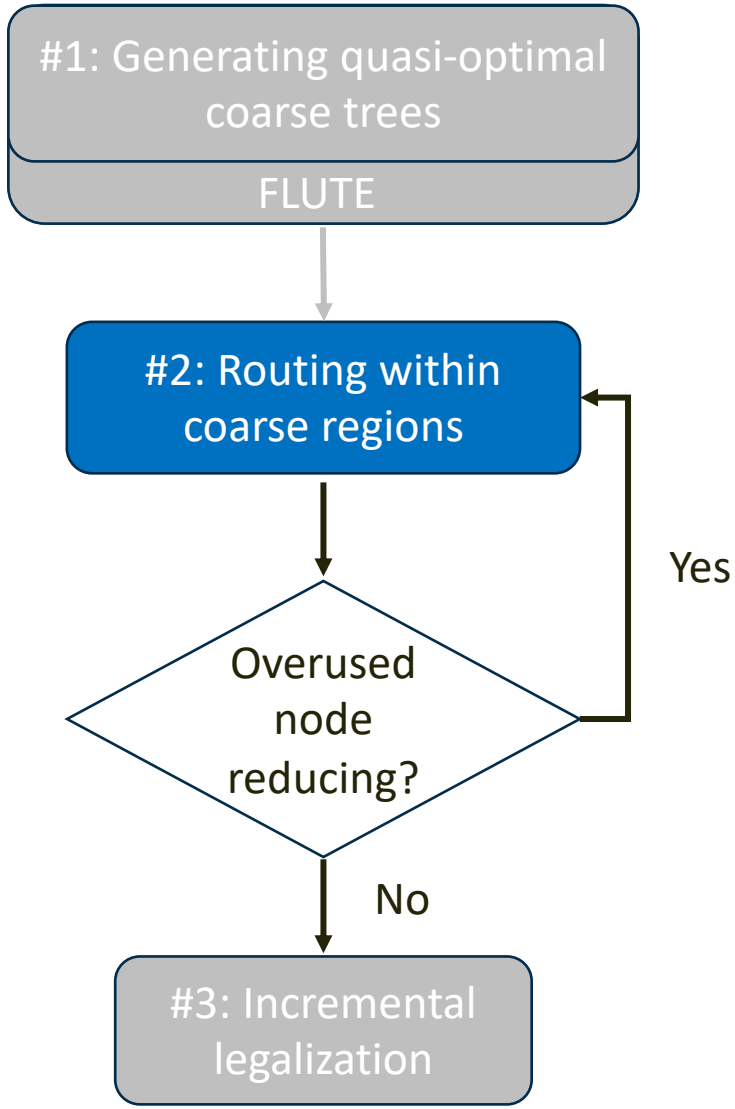


Routing Within Coarse Regions

- Modified congestion-negotiation of PathFinder
- Strictly constrained to rectilinear edges

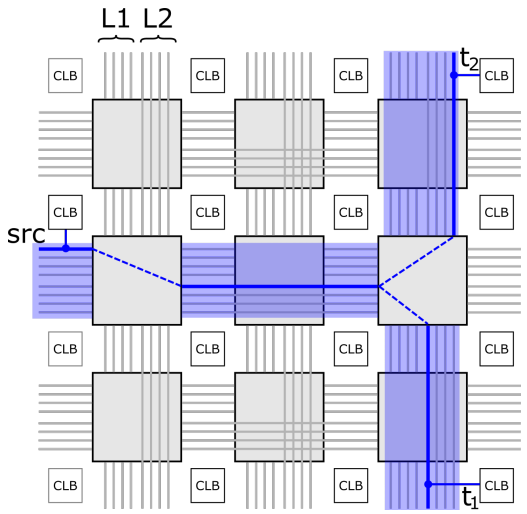


- Unlikely to legalize within coarse regions

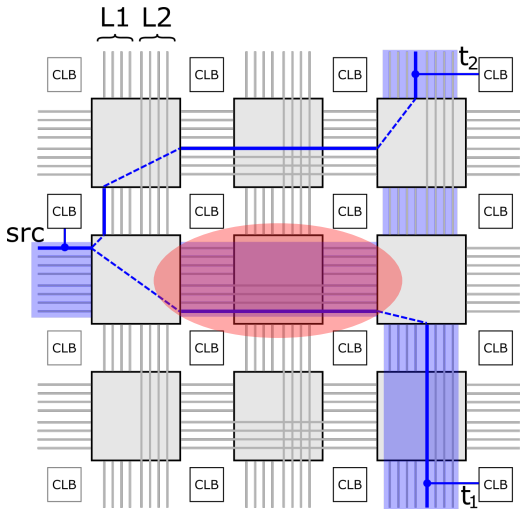


Incremental Legalization

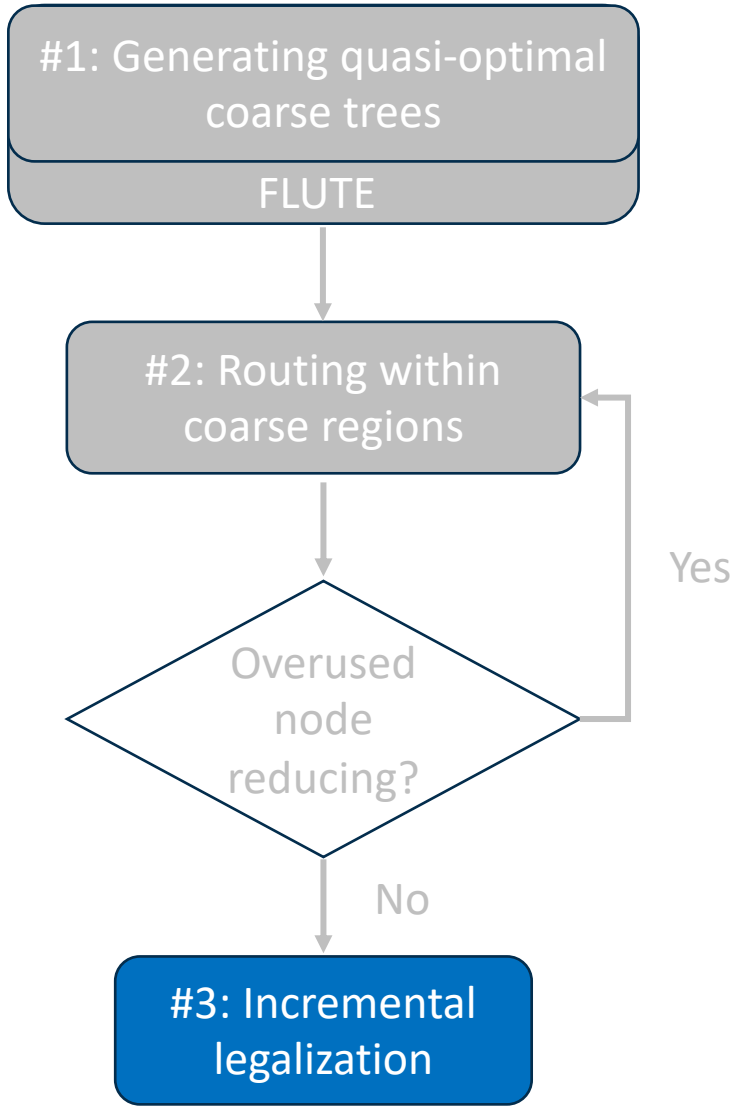
- Softly encourage reuse of nodes within coarse regions
- For a node u: $cost(u) = pfCost(u) \cdot (1 - bias(u))$



Tree preserved with bias



Tree changes with increasing congestion

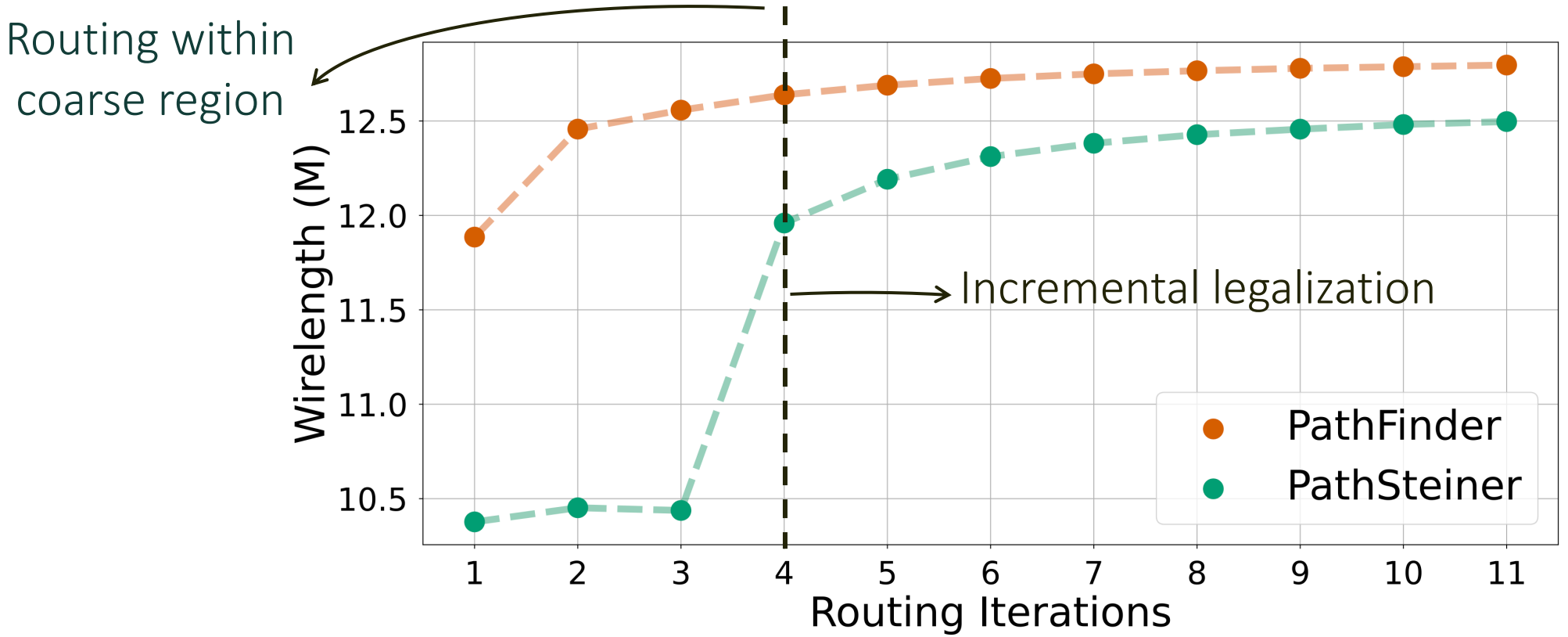


Experimental Setup

- Architecture: Closely resembling AMD UltraScale
- Benchmarks: ISPD16
- Placements: UTPlaceF
- Router: Verilog-to-Routing 8 (VTR8)

PathFinder vs PathSteiner: Wirelength

For the most congested benchmark (FPGA05)

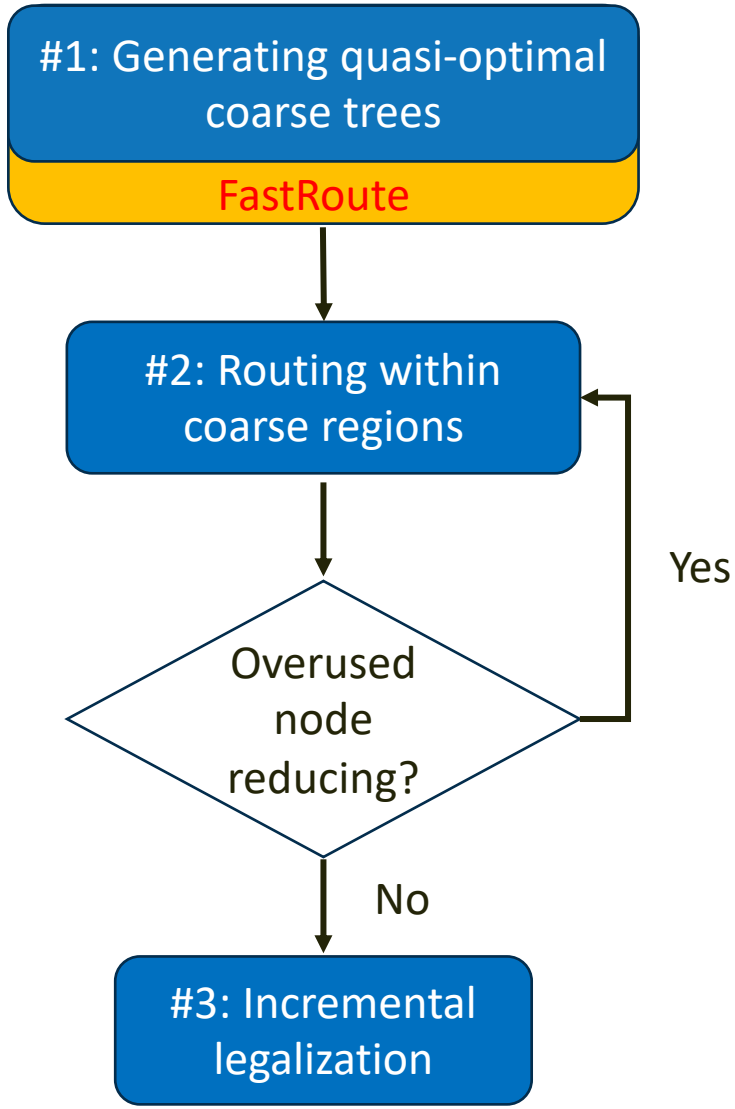


5.6% wirelength ↓ on average

PathSteiner with FastRoute

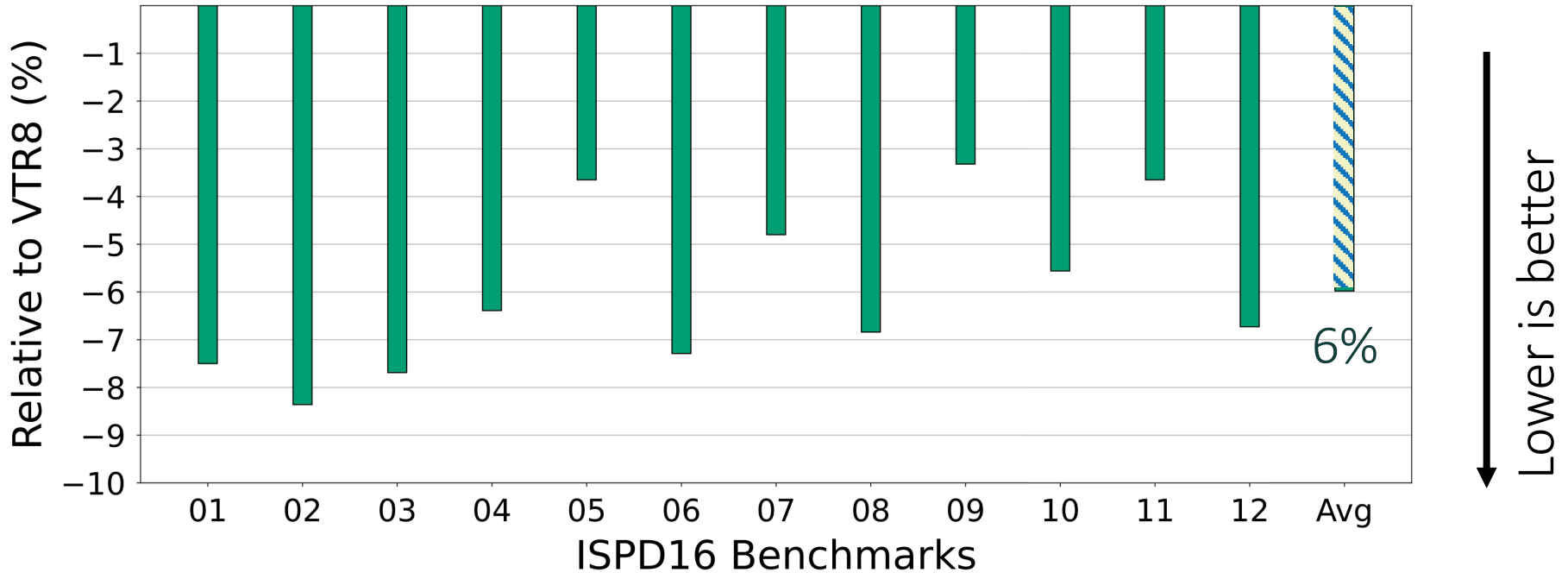
- Incremental legalization is needed because
  - FLUTE is congestion-oblivious
  - FPGAs have fixed and discrete architecture

- Replace FLUTE with FastRoute
  - A global ASIC router (OpenROAD Project)



Wirelength Reduction Per Benchmark

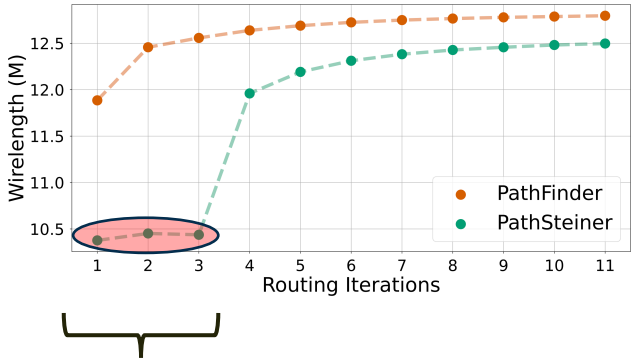
FastRoute adds marginal benefit (+0.4 percentage points)



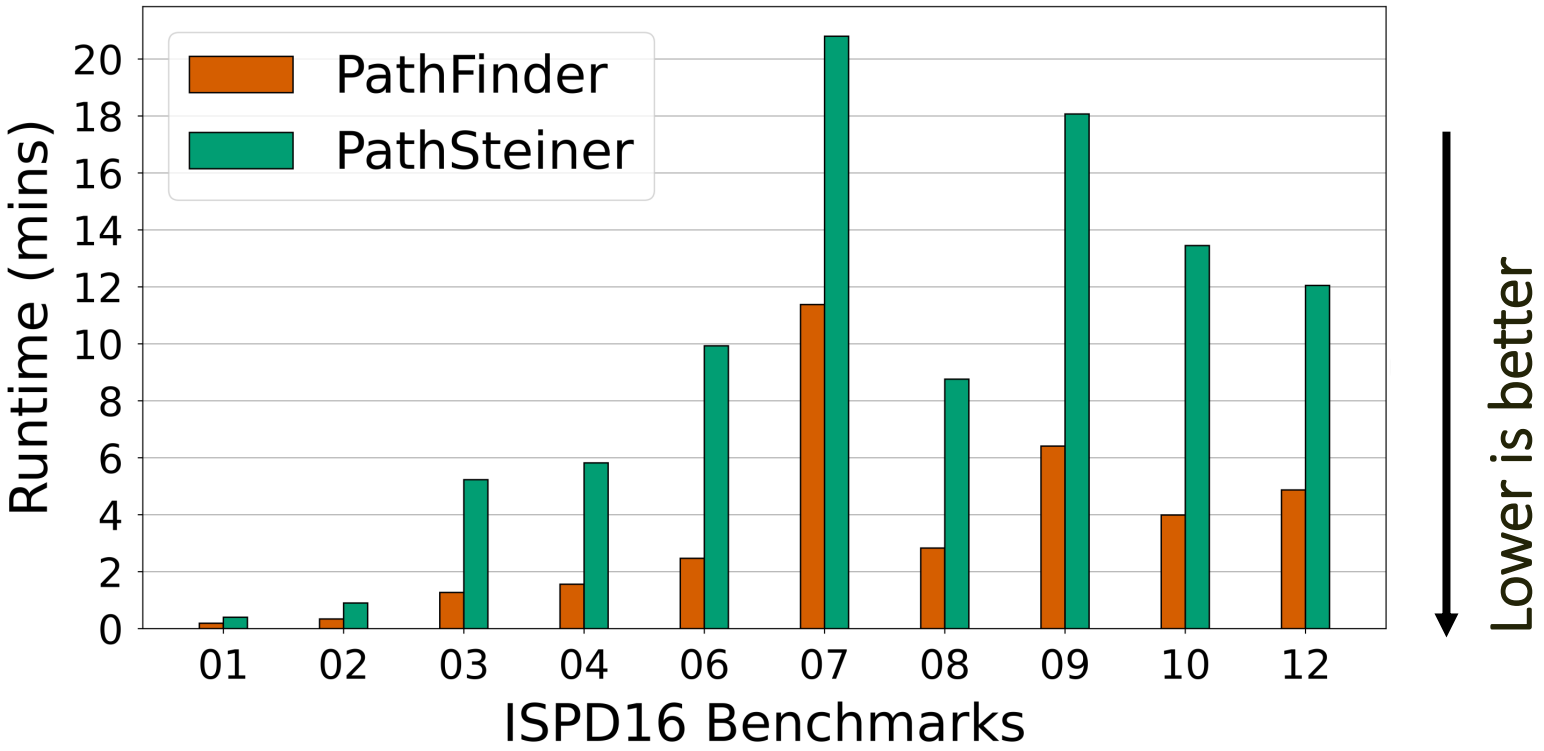
6% wirelength ↓ on average

Runtime for Benchmarks Taking Minutes

Runtime increases mainly due to multiple iterations within coarse-region routing



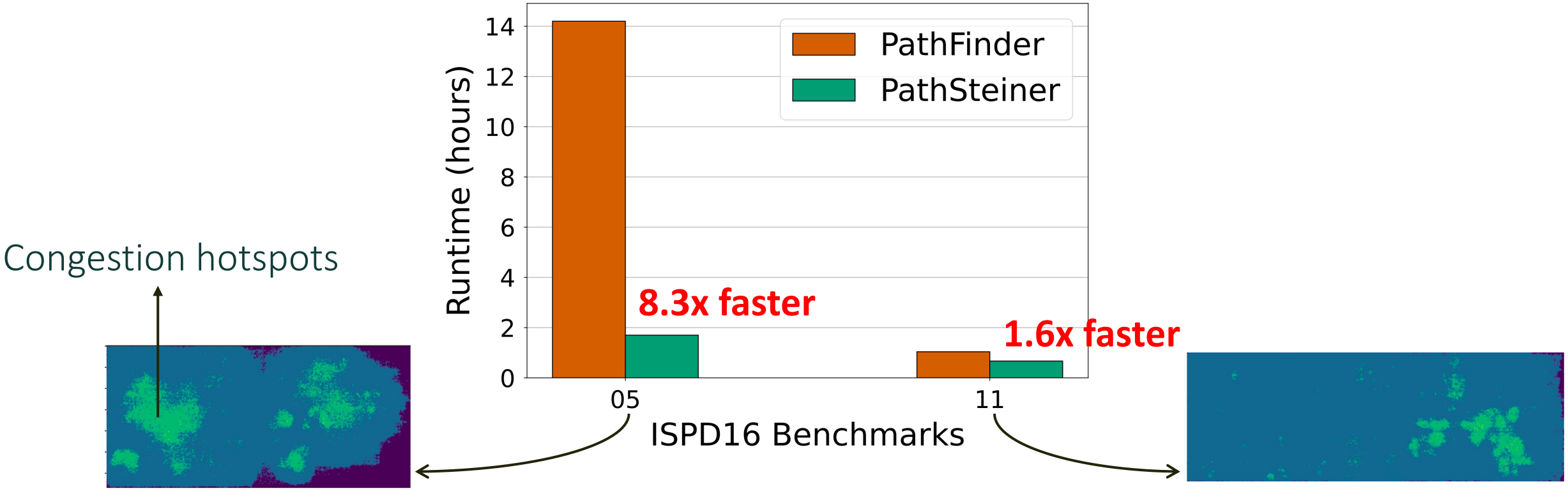
Two extra iterations



Geomean runtime increases by 2x

Runtime for Benchmarks Taking Hours

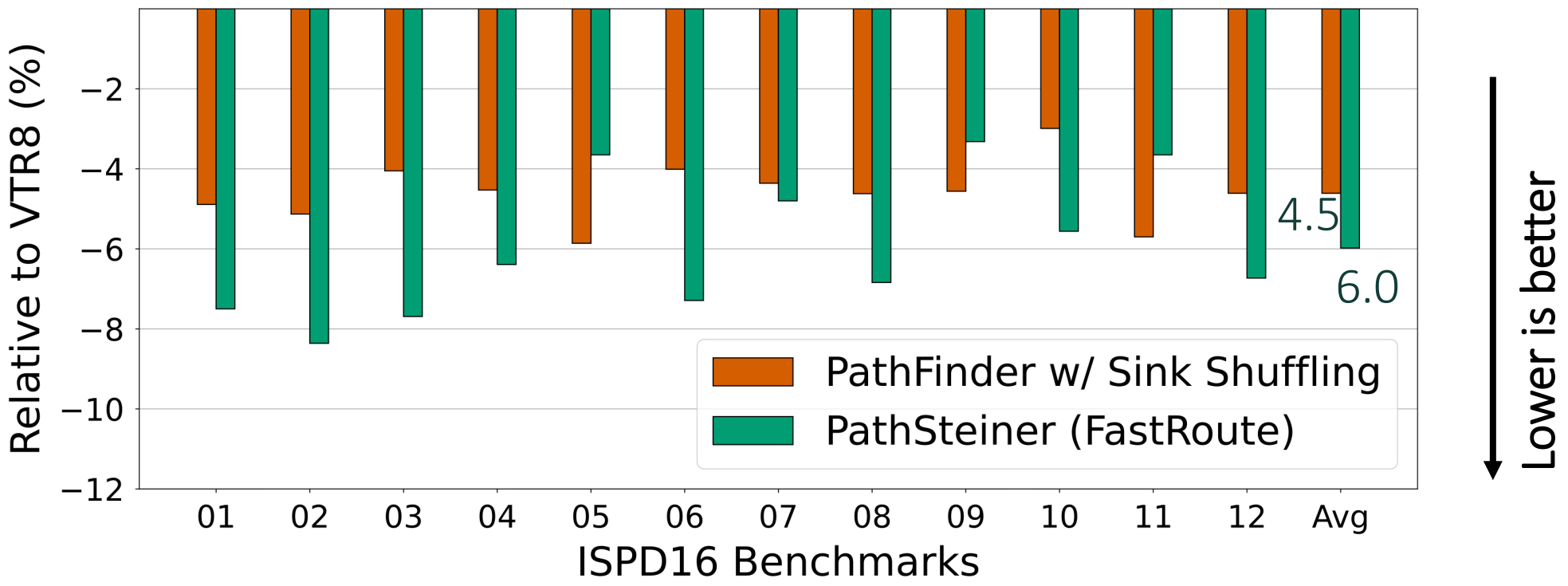
The overhead of multiple iterations is offset by quasi-optimal initial trees



Runtime improves for the two most congested benchmarks

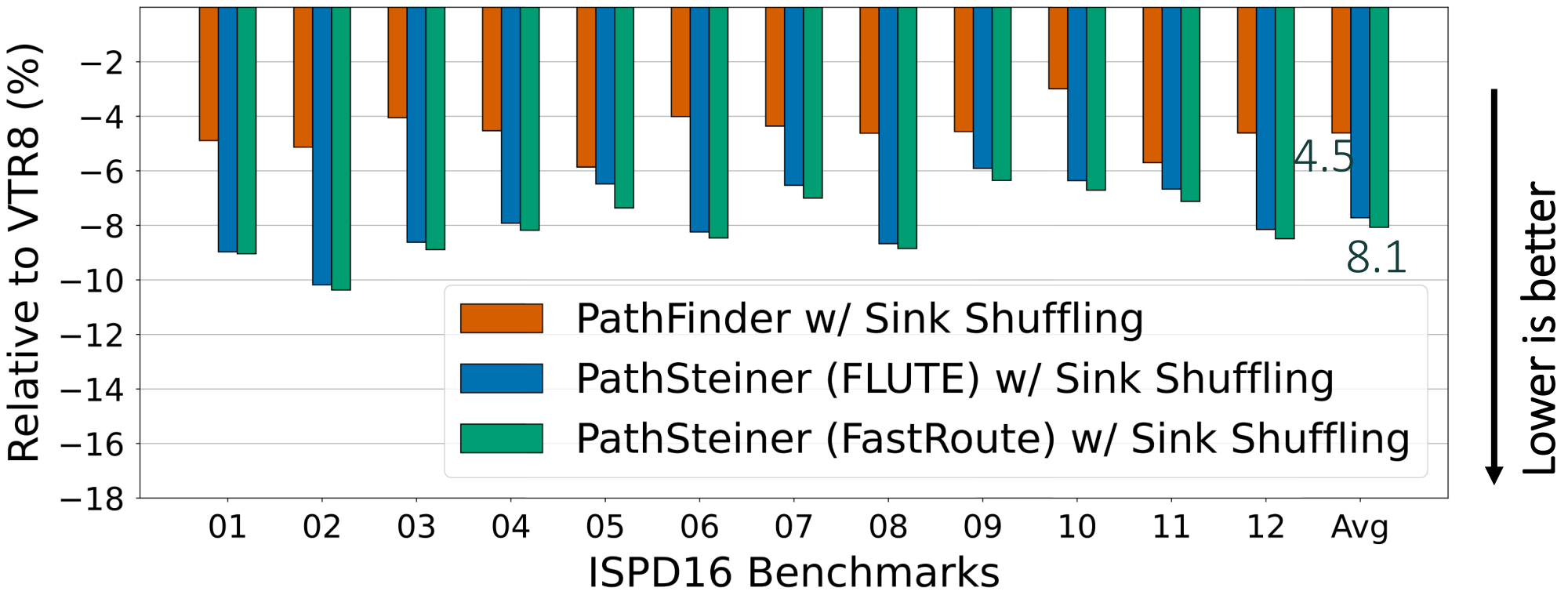
Sink Shuffling vs PathSteiner: Wirelength

PathFinder with sink shuffling: 4.5% wirelength reduction



PathSteiner outperforms PathFinder with sink shuffling on average

Combining PathSteiner with Sink Shuffling



PathSteiner with sink shuffling reduces wirelength by 8%

Outline

- Background
- Sink Shuffling
- PathSteiner
- Takeaways

Takeaways

- Quasi-optimal initialization and tree preservation → better route trees
- 8% average wirelength improvement with PathSteiner and sink shuffling
- 8× reduction in runtime for the most congested benchmark
- We propose shifting from monolithic (single-stage) to multi-stage routing

Copyright and Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

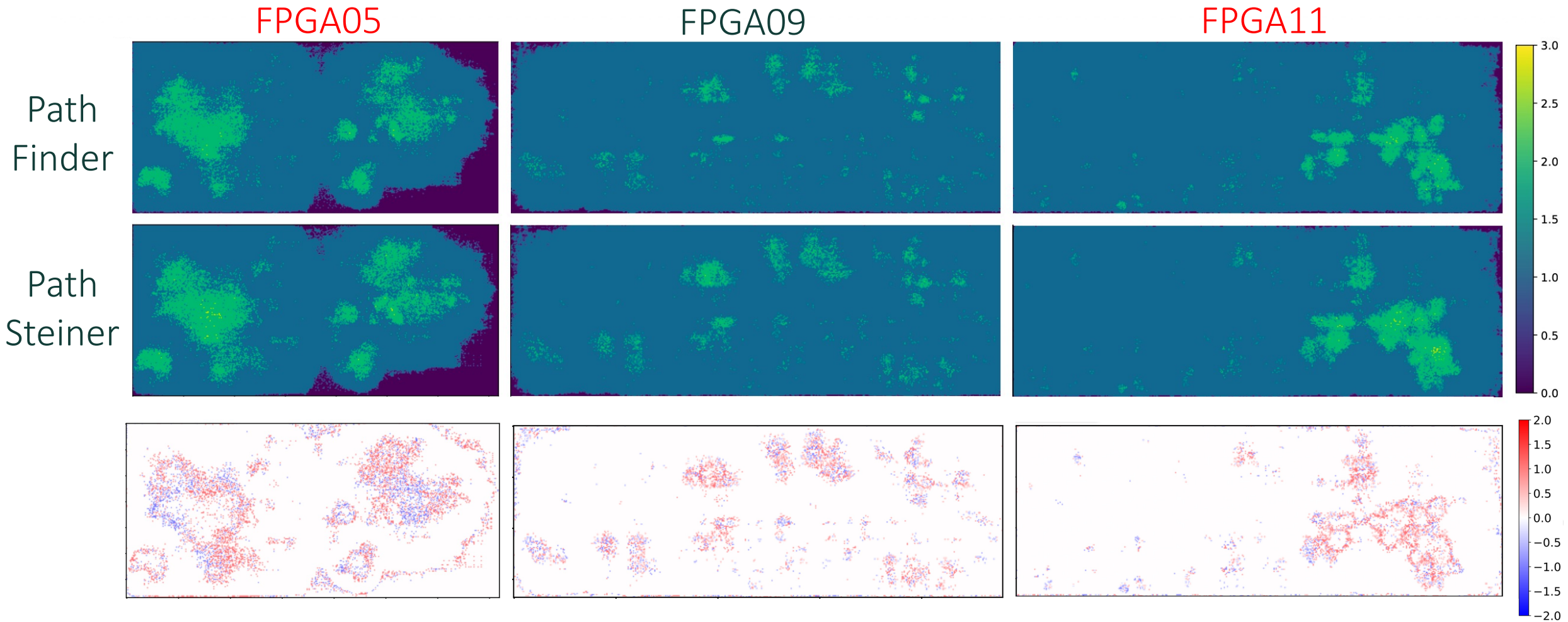
THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Takeaways

Artifacts

- Quasi-optimal initialization and tree preservation → better route trees
- 8% average wirelength improvement with PathSteiner and sink shuffling
- 8× reduction in runtime for the most congested benchmark
- We propose shifting from monolithic (single-stage) to multi-stage routing

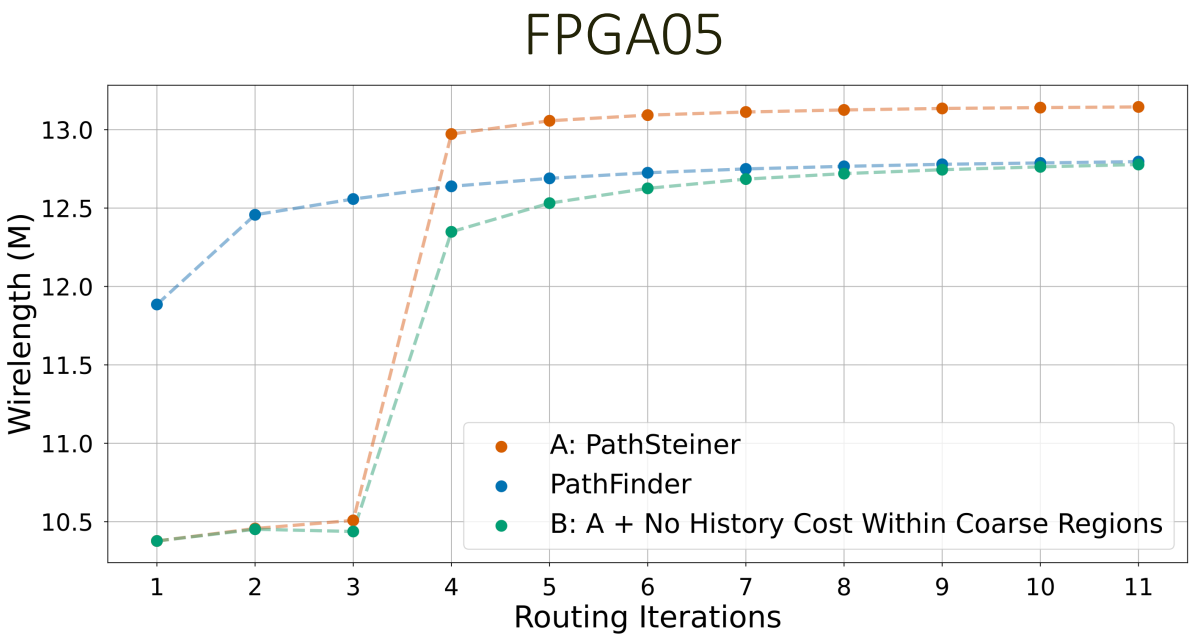
PathSteiner - PathFinder



Congested hotspots becoming spatially bigger

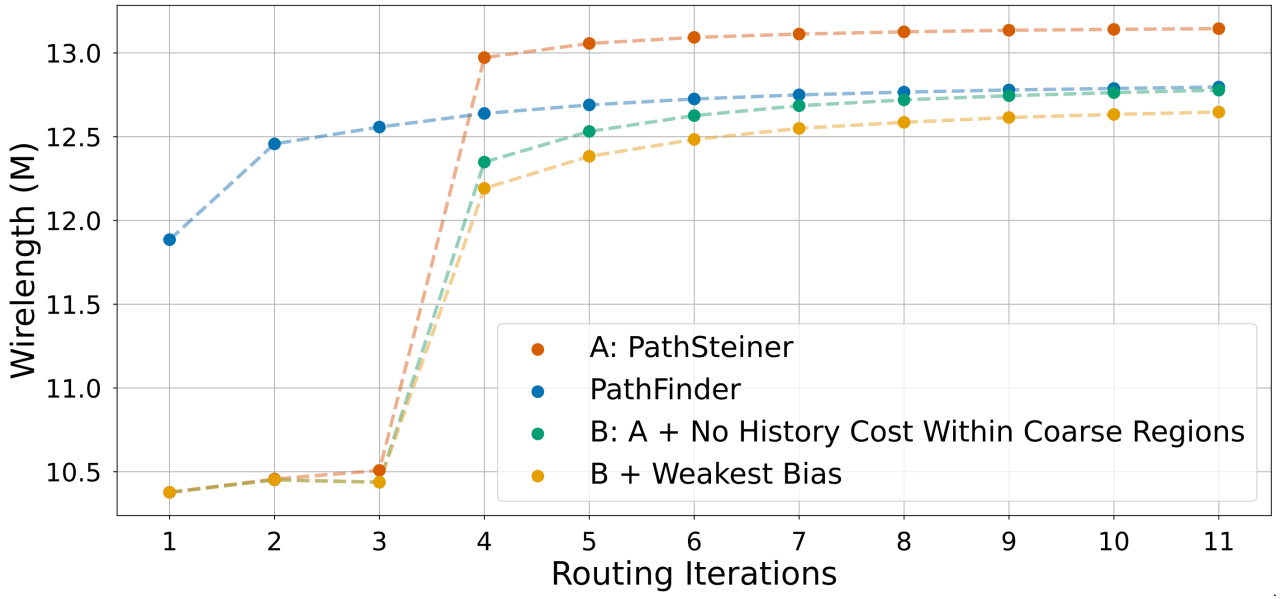
Changing Congestion Resolution Strategy

- No history cost accumulation when routing within coarse regions
- +1.1% incremental gain, reaching 4.5% total wirelength reduction on average
- FPGA05 is still worse



Applying Bias

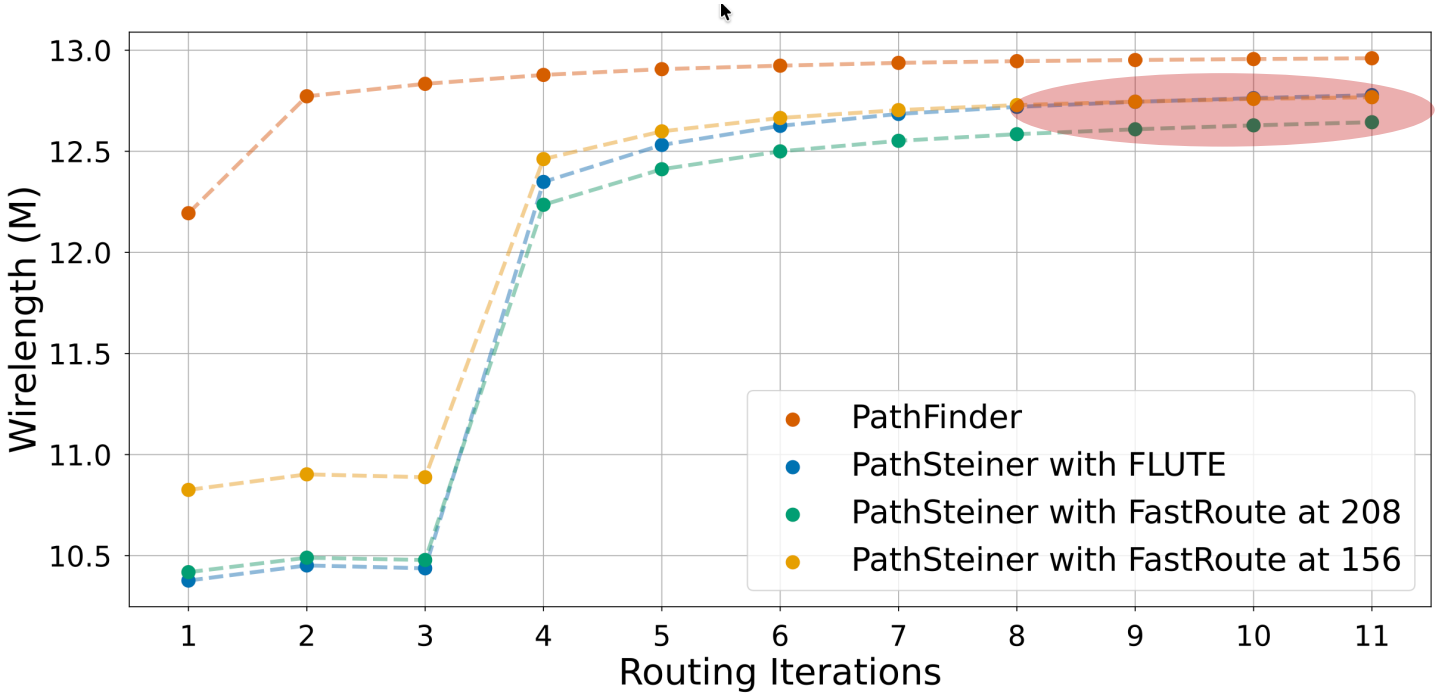
- Controlling the full removal of geometric guidance
- Sweep $bias\ parameter \in \{0.2, 0.4, 0.6, 0.8\}$
- With the weakest bias (0.2), FPGA05 improves compared to baseline finally



Bias tuning delivers a 5.6% average reduction in final wirelength

Limits of Channel Capacity Reduction

- Reduce global channel capacity by 25% and 50% to improve legalization more
- But run detailed routing at max channel capacity



Lower global channel capacity → worse wirelength