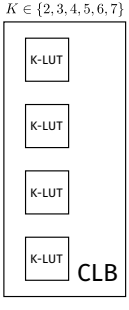
Turning PathFinder Upside-Down:
Exploring FPGA Switch-Blocks by Negotiating Switch Presence

Stefan Nikolić and Paolo Ienne

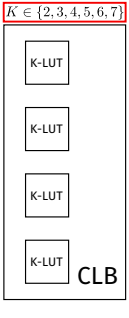
FPL'21, Dresden, 02.09.2021

École Polytechnique Fédérale de Lausanne

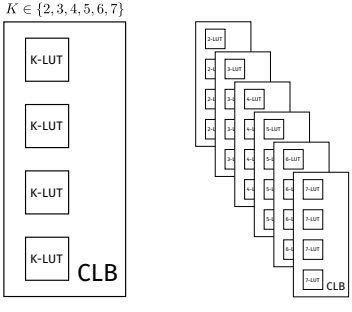
How Do We Explore FPGA Architecture?



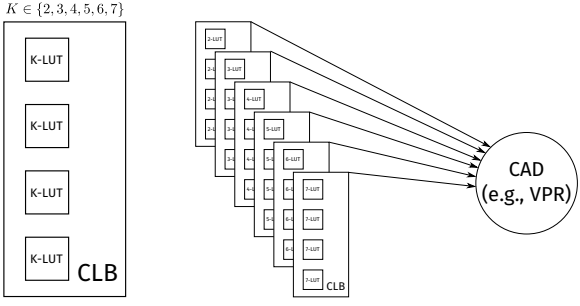
How Do We Explore FPGA Architecture?



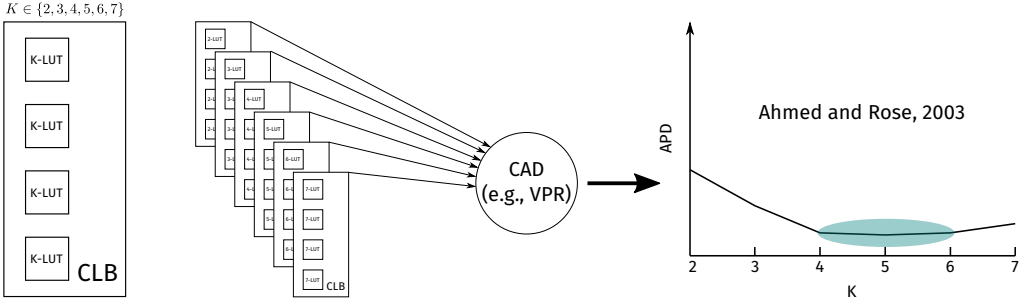
How Do We Explore FPGA Architecture?



How Do We Explore FPGA Architecture?

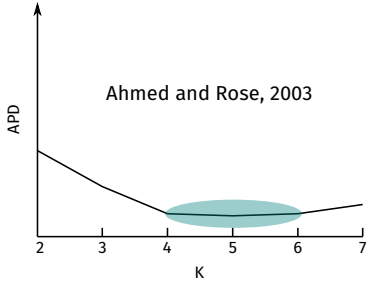


How Do We Explore FPGA Architecture?



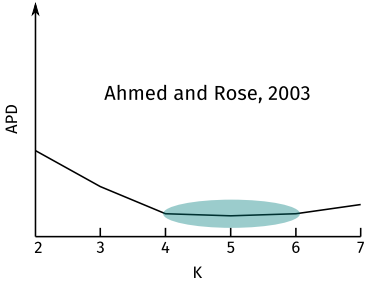
How Do We Explore FPGA Architecture?

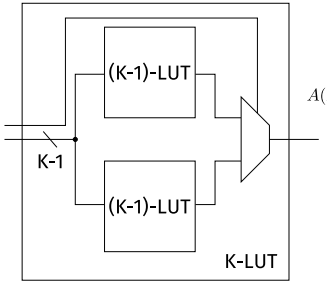
Did we explore enough?



How Do We Explore FPGA Architecture?

Did we explore enough?

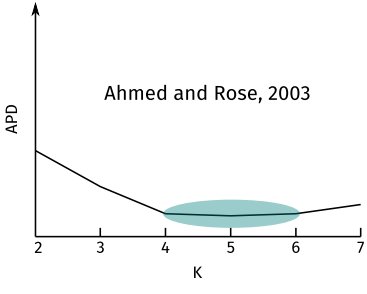


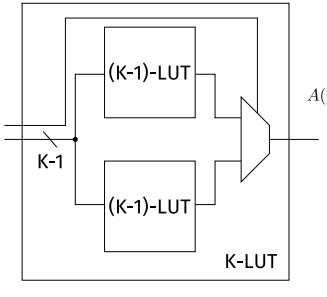


$A(K\text{-LUT}) > 2 \times A((K-1)\text{-LUT})$

How Do We Explore FPGA Architecture?

Did we explore enough?





$A(K\text{-LUT}) > 2 \times A((K-1)\text{-LUT})$

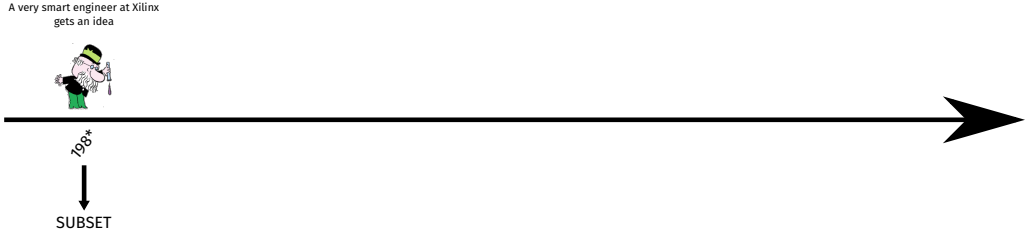
Similar approach applicable to cluster size, channel composition, etc.

How Do We Explore FPGA Architecture?

What about Switch-Block Patterns?

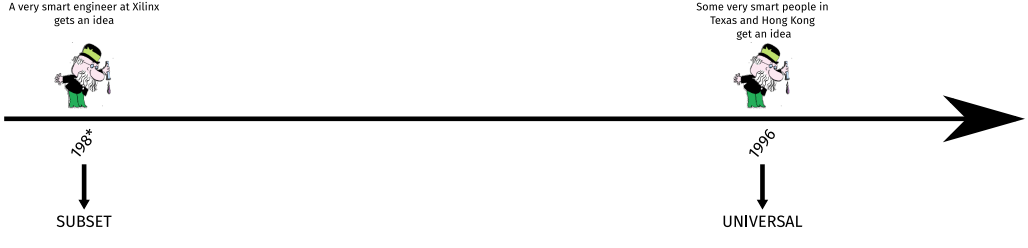
How Do We Explore FPGA Architecture?

What about Switch-Block Patterns?



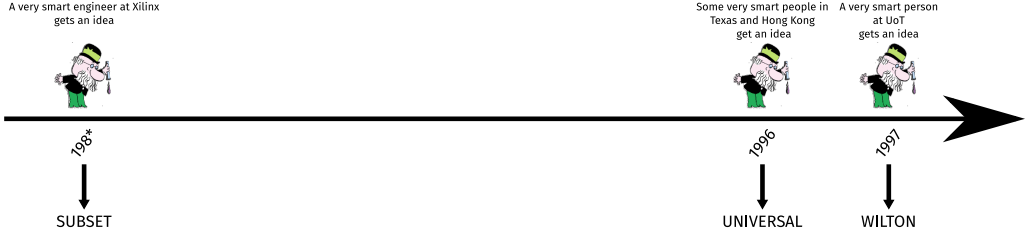
How Do We Explore FPGA Architecture?

What about Switch-Block Patterns?



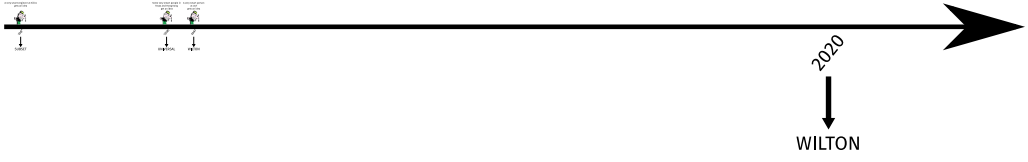
How Do We Explore FPGA Architecture?

What about Switch-Block Patterns?



How Do We Explore FPGA Architecture?

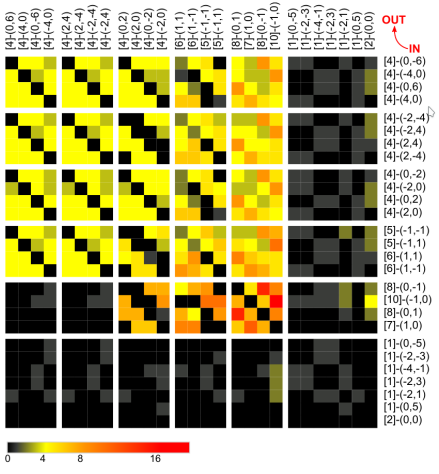
What about Switch-Block Patterns? (apologies for a bit of an exaggeration)



Meanwhile in Industry…

NetCracker: A Peek into the Routing Architecture of Xilinx 7-Series FPGAs

Morten B. Petersen, Stefan Nikolić and Mirjana Stojilović



Different

Architectural Enhancements in Intel® Agilex™ FPGAs

Jeff Chromczak
jeff.chromczak@intel.com
Intel Corporation
Toronto, Canada

Mark Wheeler
mark.wheeler@intel.com
Intel Corporation
Toronto, Canada

Charles Chiasson
charles.chiasson@intel.com
Intel Corporation
Seattle, USA

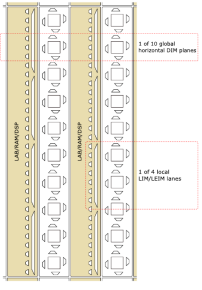
Dana How
dana.how@intel.com
Intel Corporation
San Jose, USA

Martin Langhammer
martin.langhammer@intel.com
Intel Corporation
United Kingdom

Tim Vanderhoek
tim.vanderhoek@intel.com
Intel Corporation
Toronto, Canada

Grace Zgheib
grace.zgheib@intel.com
Intel Corporation
San Jose, USA

Ilya Ganusov
ilya.ganusov@intel.com
Intel Corporation
San Jose, USA



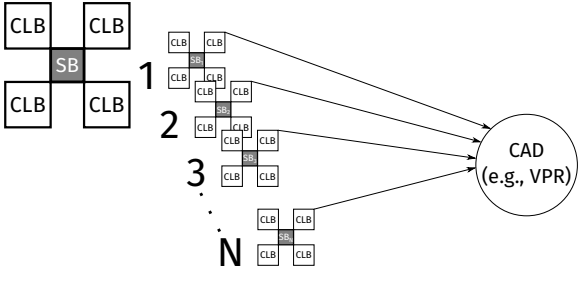
Difference increasing (Technology scaling)

How Do We Explore FPGA Architecture?

Can’t we automate SB-pattern exploration too?

How Do We Explore FPGA Architecture?

Can’t we automate SB-pattern exploration too?



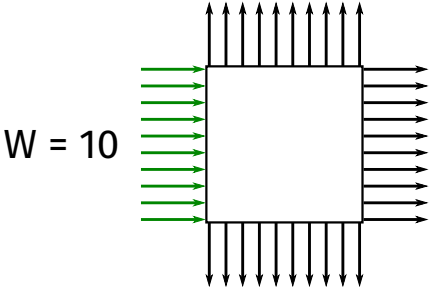
Sure!

How Do We Explore FPGA Architecture?

How big is N?

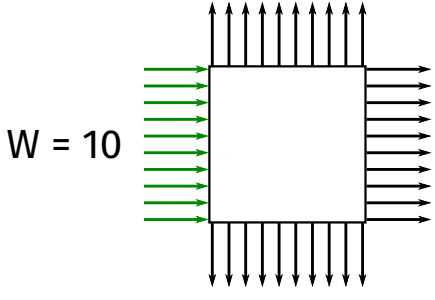
How Do We Explore FPGA Architecture?

How big is N?



How Do We Explore FPGA Architecture?

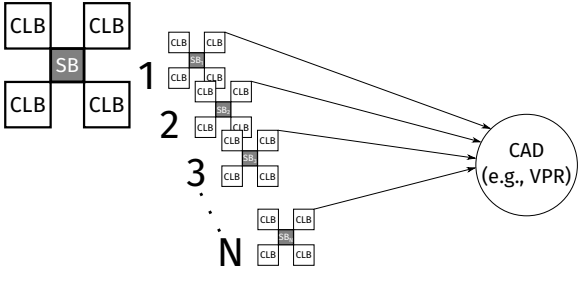
How big is N?



$$\sum_{k=10}^{30} \binom{30}{k} \cdot 10! \cdot \left\{ {k \atop 10} \right\} \qquad \sim 10^{31}$$

How Do We Explore FPGA Architecture?

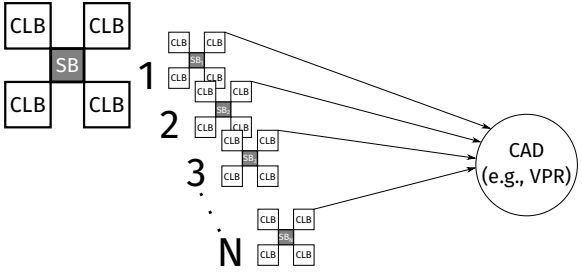
Can’t we automate SB-pattern exploration too?



Sure!

How Do We Explore FPGA Architecture?

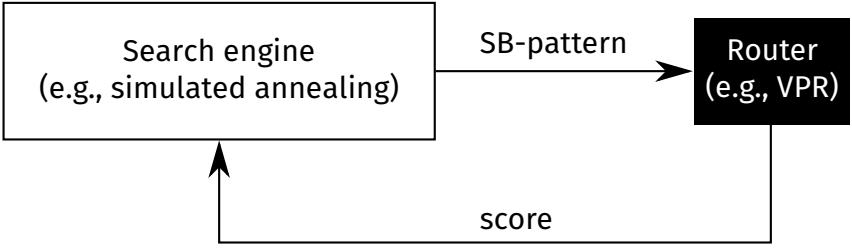
Can’t we automate SB-pattern exploration too?



Surely not like this!

Automated Switch-Pattern Exploration

An Intuitive Solution: Iterative Improvement



[1] M. Lin, J. Wawrzynek, and A. El Gamal, “Exploring FPGA routing architecture stochastically”, TCAD’10

Using the router as a black box

to evaluate enumerated solutions

is inefficient

This inefficient...

A Little Analogy

You enter a restaurant and order a soup.

A Little Analogy

You enter a restaurant and order a soup.

Waiter: “How do you find the taste of the soup sir, on scale 0–9?”

A Little Analogy

You enter a restaurant and order a soup.

Waiter: “How do you find the taste of the soup sir, on scale 0–9?”
You: “2”

A Little Analogy

You enter a restaurant and order a soup.

Waiter: “How do you find the taste of the soup sir, on scale 0–9?”
You: “2”

The waiter goes away with your soup, adds some salt and comes back.

A Little Analogy

You enter a restaurant and order a soup.

Waiter: “How do you find the taste of the soup sir, on scale 0–9?”
You: “2”

The waiter goes away with your soup, adds some salt and comes back.

Waiter: “Please try it now sir.”

A Little Analogy

You enter a restaurant and order a soup.

Waiter: “How do you find the taste of the soup sir, on scale 0–9?”
You: “2”

The waiter goes away with your soup, adds some salt and comes back.

Waiter: “Please try it now sir.”
You: “4”

A Little Analogy

You enter a restaurant and order a soup.

Waiter: “How do you find the taste of the soup sir, on scale 0–9?”
You: “2”

The waiter goes away with your soup, adds some salt and comes back.

Waiter: “Please try it now sir.”
You: “4”

The waiter disappears with the soup again and adds some pepper.

A Little Analogy

You enter a restaurant and order a soup.

Waiter: “How do you find the taste of the soup sir, on scale 0–9?”
You: “2”

The waiter goes away with your soup, adds some salt and comes back.

Waiter: “Please try it now sir.”
You: “4”

The waiter disappears with the soup again and adds some pepper.

Waiter: “How about now sir?”

A Little Analogy

You enter a restaurant and order a soup.

Waiter: “How do you find the taste of the soup sir, on scale 0–9?”
You: “2”

The waiter goes away with your soup, adds some salt and comes back.

Waiter: “Please try it now sir.”
You: “4”

The waiter disappears with the soup again and adds some pepper.

Waiter: “How about now sir?”
You: “3”

A Little Analogy

You enter a restaurant and order a soup.

Waiter: “How do you find the taste of the soup sir, on scale 0–9?”
You: “2”

The waiter goes away with your soup, adds some salt and comes back.

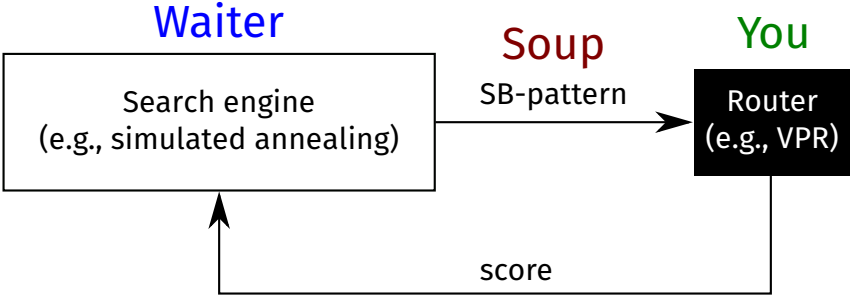
Waiter: “Please try it now sir.”
You: “4”

The waiter disappears with the soup again and adds some pepper.

Waiter: “How about now sir?”
You: “3”

The waiter disappears yet again, but this time you leave the table too.

A Little Analogy



A Little Analogy

Now something that we are a bit more accustomed to:

You enter a restaurant and order a soup.

A Little Analogy

Now something that we are a bit more accustomed to:

You enter a restaurant and order a soup.

Waiter: “Here you go sir, and here are all the spices we have available, in case you miss something.”

A Little Analogy

Now something that we are a bit more accustomed to:

You enter a restaurant and order a soup.

Waiter: “Here you go sir, and here are all the spices we have available, in case you miss something.”

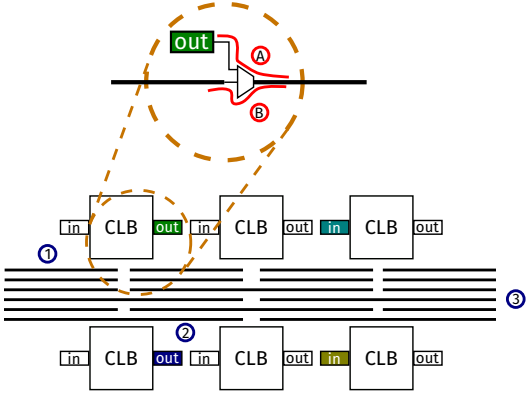
You try the soup and add a bit of each spice that the waiter gave you, according to your taste and habit.

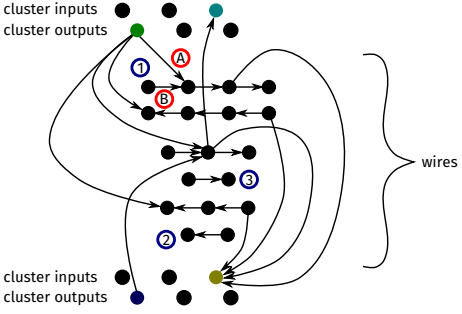
Can the router
spice up its own soup?

Can the router
design the switch-pattern?

A Quick Recap on FPGA Routers

Representing an FPGA as a Graph





Each node has a cost

Very Simplified Algorithm

```
repeat
    foreach CONNECTION IN THE CIRCUIT do
        route using shortest path in the RR-graph;

        update node costs;


    end
until OVERUSED NODES EXIST;
```

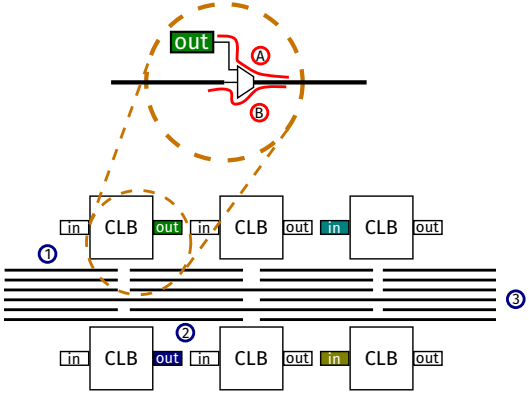
Very Simplified Algorithm

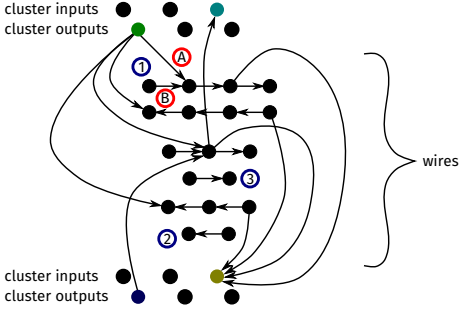
```
repeat
    foreach CONNECTION IN THE CIRCUIT do
        route using shortest path in the RR-graph;

        update node costs;

    end
until OVERUSED NODES EXIST;
```

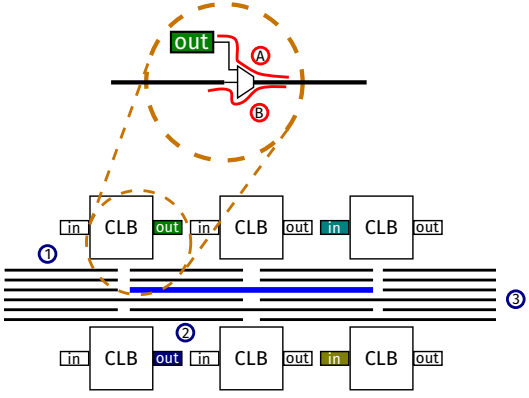
Very Simplified Algorithm

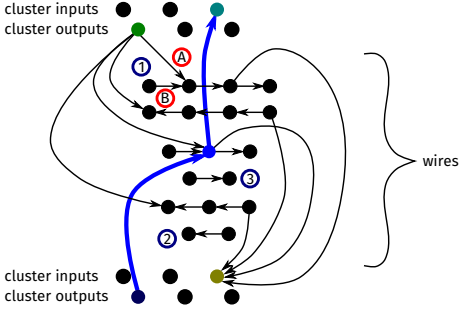




Each node has a cost

Very Simplified Algorithm





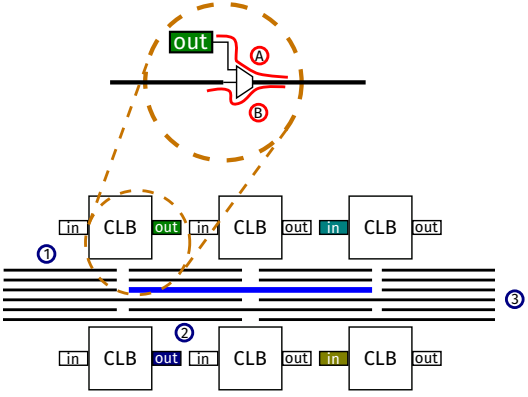
Each node has a cost

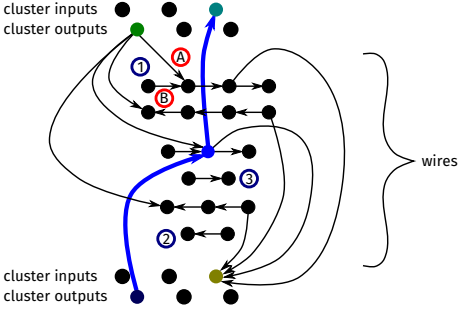
Very Simplified Algorithm

```
repeat
    foreach CONNECTION IN THE CIRCUIT do
        route using shortest path in the RR-graph;
        (different signals can overlap)
        update node costs;

    end
until OVERUSED NODES EXIST;
```

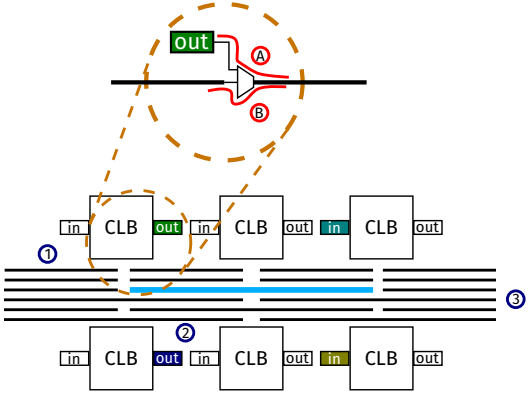
Very Simplified Algorithm

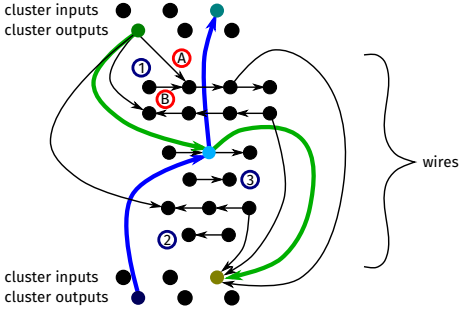




Each node has a cost

Very Simplified Algorithm



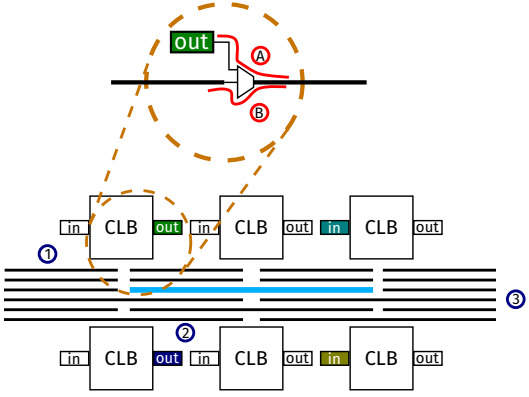


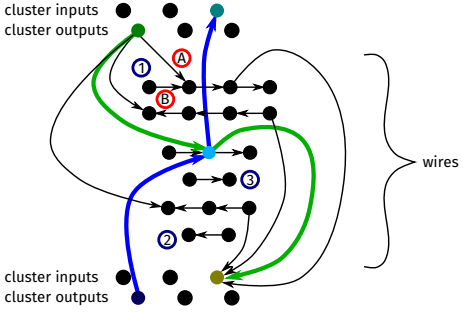
Each node has a cost

Very Simplified Algorithm

```
repeat
    foreach CONNECTION IN THE CIRCUIT do
        route using shortest path in the RR-graph;
        (different signals can overlap)
        update node costs;
    end
    update node costs;
until OVERUSED NODES EXIST;
```

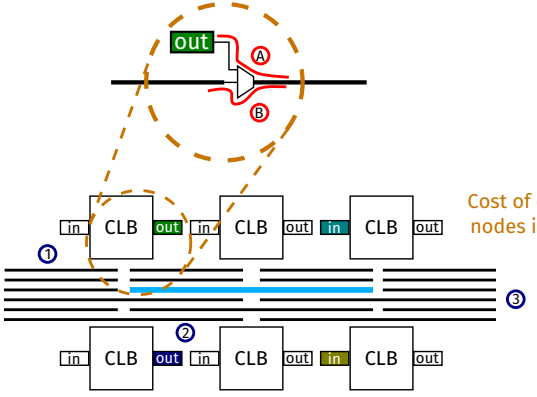
Very Simplified Algorithm



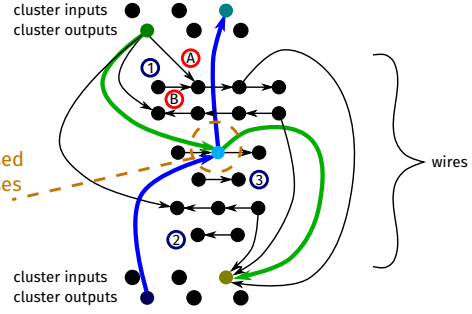


Each node has a cost

Very Simplified Algorithm



Cost of overused nodes increases

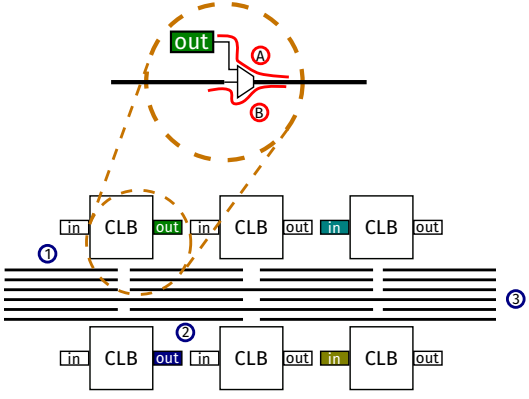


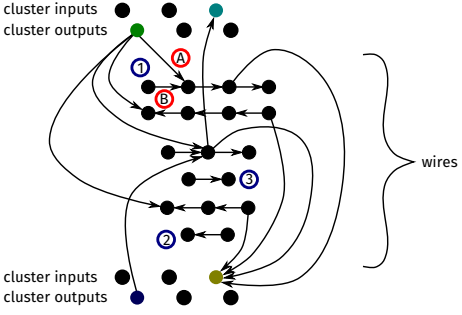
Each node has a cost

Very Simplified Algorithm

```
repeat
    foreach CONNECTION IN THE CIRCUIT do
        route using shortest path in the RR-graph;
        (different signals can overlap)
        update node costs;
    end
    update node costs;
    throw away all routing (rip-up);
until OVERUSED NODES EXIST;
```

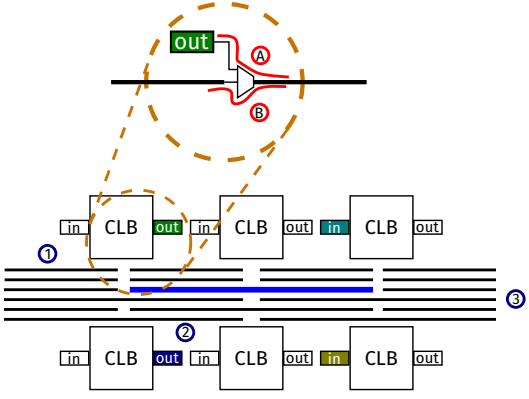
Very Simplified Algorithm

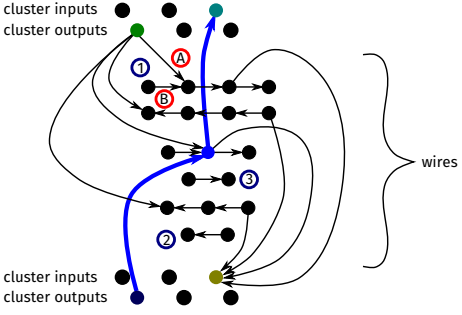




Each node has a cost

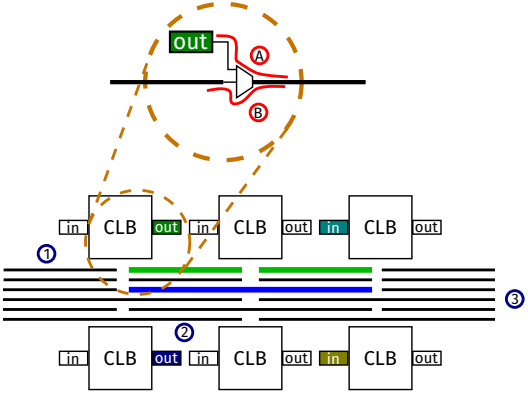
Very Simplified Algorithm

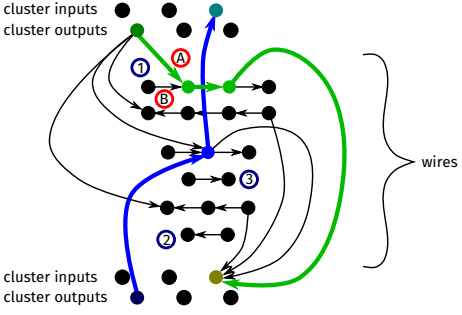




Each node has a cost

Very Simplified Algorithm

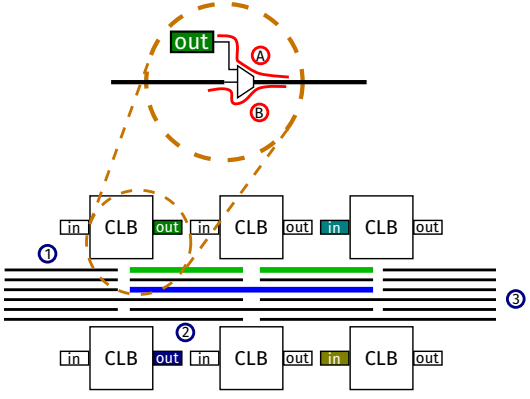


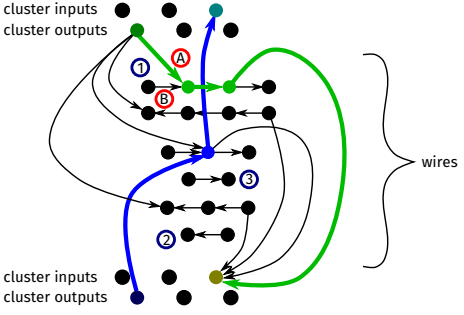


Each node has a cost

Very Simplified Algorithm

Signals negotiate which one will give up its desired nodes

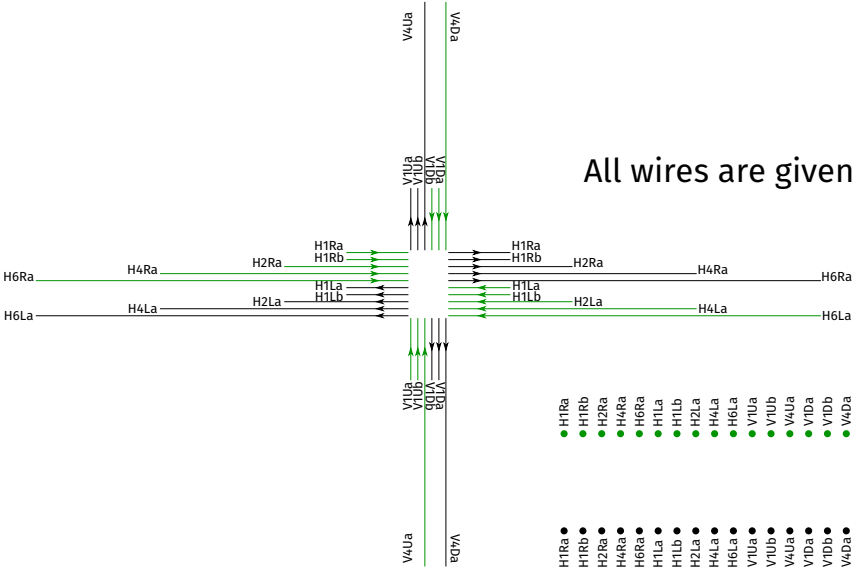




Each node has a cost

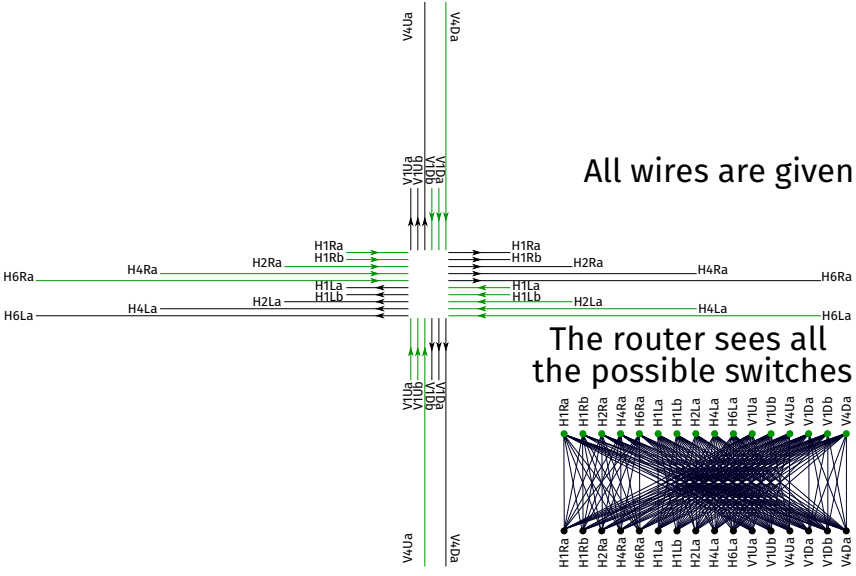
Negotiating Switch Presence

Letting the Router Design the SB-Pattern



All wires are given

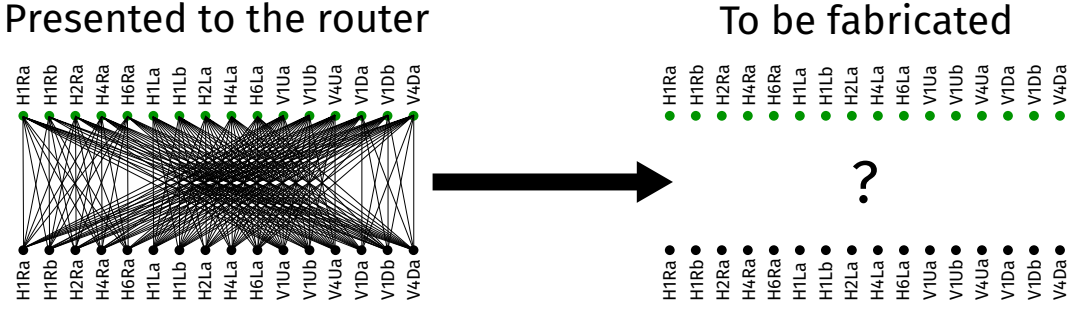
Letting the Router Design the SB-Pattern



All wires are given

The router sees all the possible switches

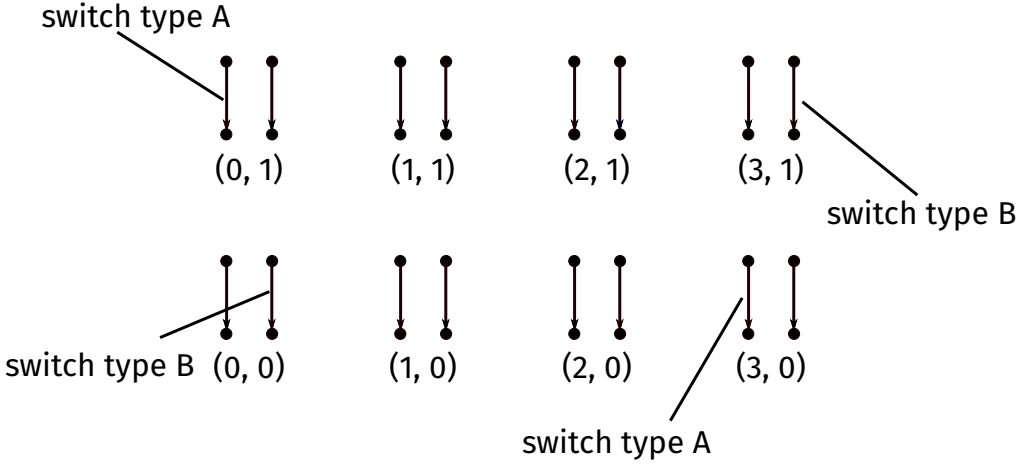
Letting the Router Design the SB-Pattern



Use the router's decisions to select switches for fabrication

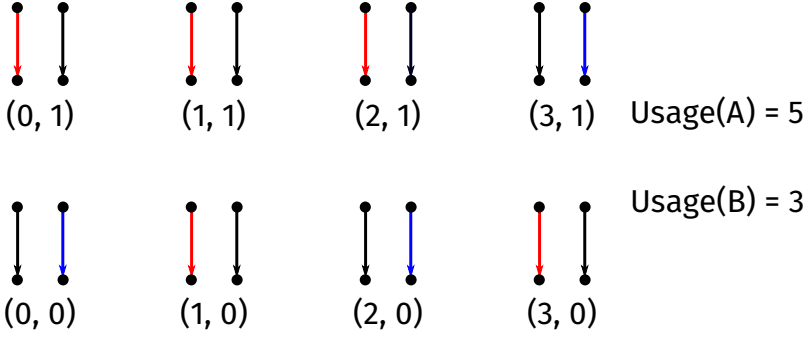
A Simple Greedy Solution

Usage(*e*) = # SB instances (tiles) in which a switch *type e* is used.



A Simple Greedy Solution

Usage(*e*) = # SB instances (tiles) in which a switch *type e* is used.



A Simple Greedy Solution

Usage(e) = # SB instances (tiles) in which a switch *type* e is used.

1. Set the cost of all switch types not yet taken to some small cost ϵ.

A Simple Greedy Solution

Usage(e) = # SB instances (tiles) in which a switch *type* e is used.

1. Set the cost of all switch types not yet taken to some small cost ϵ.
2. Route all circuits.

A Simple Greedy Solution

Usage(e) = # SB instances (tiles) in which a switch *type* e is used.

1. Set the cost of all switch types not yet taken to some small cost ϵ.
2. Route all circuits.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.

A Simple Greedy Solution

$Usage(e)$ = # SB instances (tiles) in which a switch *type* e is used.

1. Set the cost of all switch types not yet taken to some small cost ϵ.
2. Route all circuits.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.
4. Take all switch types with $Usage > Usage_{max}/\theta, \theta > 1$.

A Simple Greedy Solution

$Usage(e)$ = # SB instances (tiles) in which a switch *type* e is used.

1. Set the cost of all switch types not yet taken to some small cost ϵ.
2. Route all circuits.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.
4. Take all switch types with $Usage > Usage_{max}/\theta, \theta > 1$.
5. Set the cost of all taken switch types to 0.

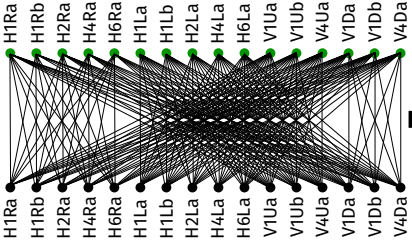
A Simple Greedy Solution

$Usage(e)$ = # SB instances (tiles) in which a switch *type* e is used.

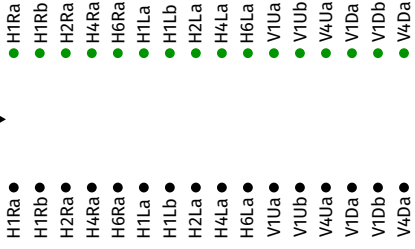
1. Set the cost of all switch types not yet taken to some small cost ϵ.
2. Route all circuits.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.
4. Take all switch types with $Usage > Usage_{max}/\theta, \theta > 1$.
5. Set the cost of all taken switch types to 0.
6. If there are newly taken switch types, go to step 2.

Letting the Router Design the SB-Pattern

Presented to the router

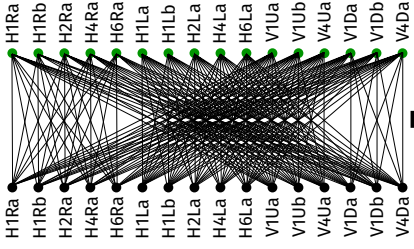


To be fabricated

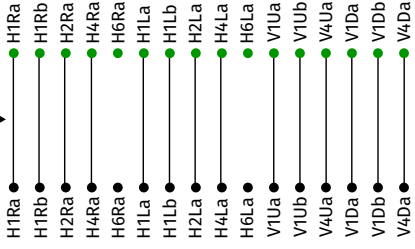


Letting the Router Design the SB-Pattern

Presented to the router

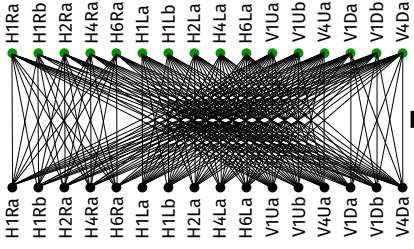


To be fabricated

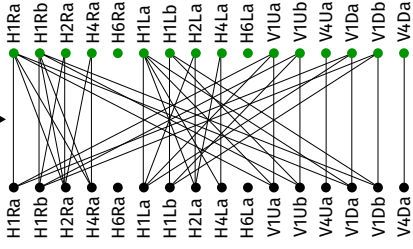


Letting the Router Design the SB-Pattern

Presented to the router

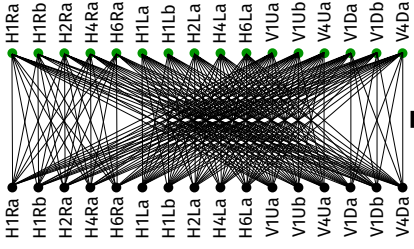


To be fabricated

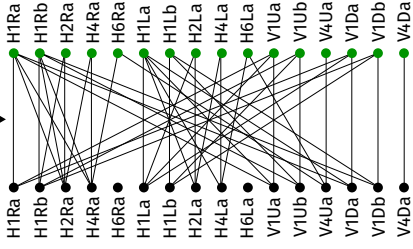


Letting the Router Design the SB-Pattern

Presented to the router

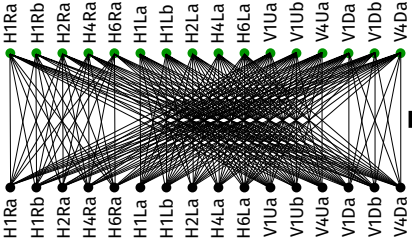


To be fabricated

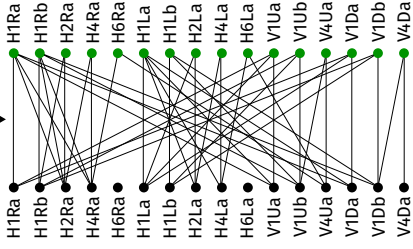


Letting the Router Design the SB-Pattern

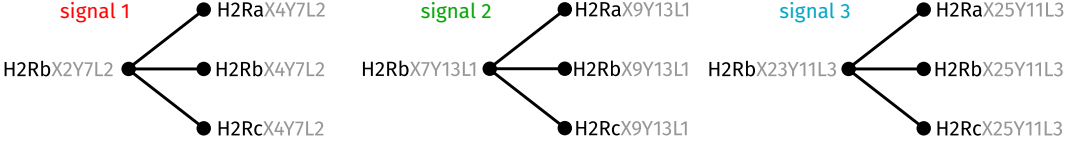
Presented to the router



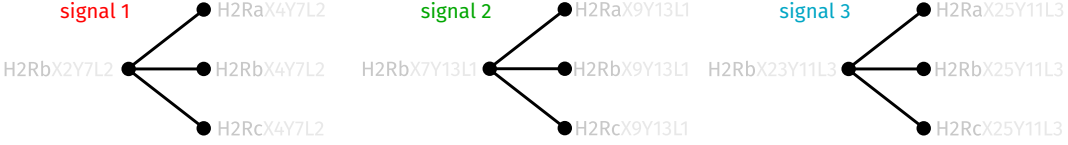
To be fabricated



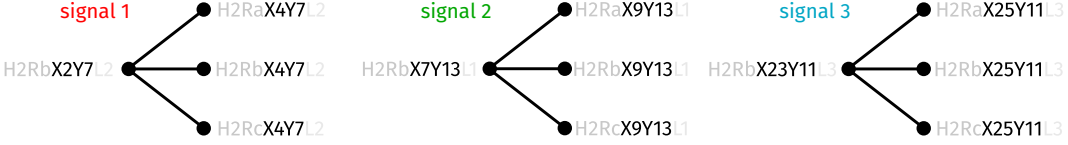
A Simple Greedy Solution: Failure



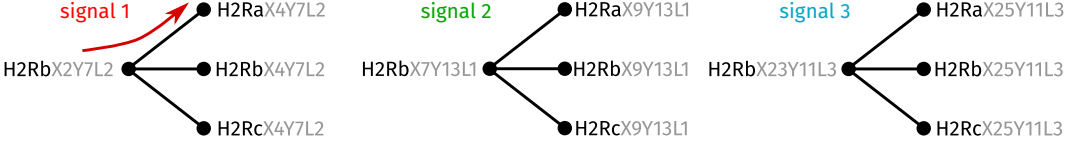
A Simple Greedy Solution: Failure



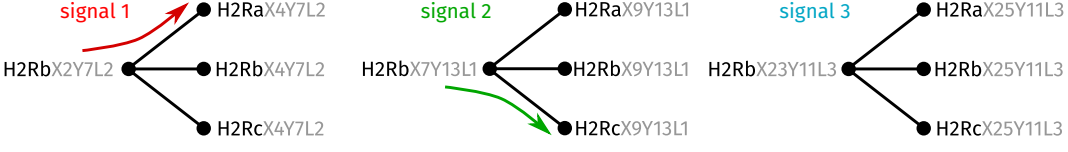
A Simple Greedy Solution: Failure



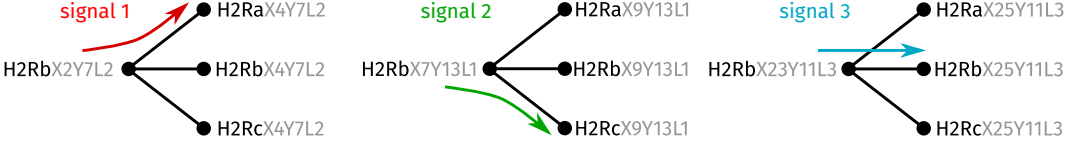
A Simple Greedy Solution: Failure



A Simple Greedy Solution: Failure



A Simple Greedy Solution: Failure



A Simple Greedy Solution: Failure

1. Set the cost of all switch types not yet taken to some small cost ϵ.
2. Route all circuits.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.
4. Take all switch types with $Usage > Usage_{max}/\theta, \theta > 1$.
5. Set the cost of all taken switch types to 0.
6. If there are newly taken switch types, go to step 2.

A Simple Greedy Solution: Failure

1. Set the cost of all switch types not yet taken to some small cost ϵ.
2. Route all circuits.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.
4. Take all switch types with $Usage > Usage_{max}/\theta, \theta > 1$.
5. Set the cost of all taken switch types to 0.
6. If there are newly taken switch types, go to step 2.

Must take all three switch types from the example

Avalanche Costs (Key Idea)

Avalanche Costs: Rationale

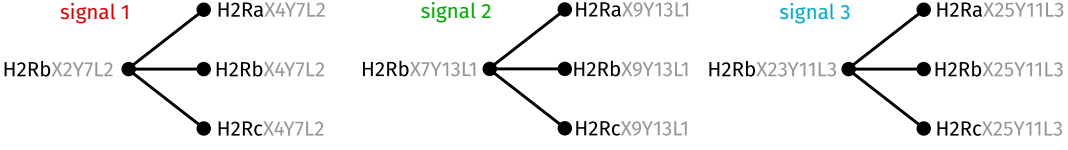
```
repeat
    foreach CONNECTION IN THE CIRCUIT do
        route using shortest path in the RR-graph;
        update node costs;
    end
    update node costs;
    throw away all routing (rip-up);
until OVERUSED NODES EXIST;
```

Avalanche Costs: Rationale

What if switch costs were inversely related to Usage?
(Usage is common to all instances of the same switch type)

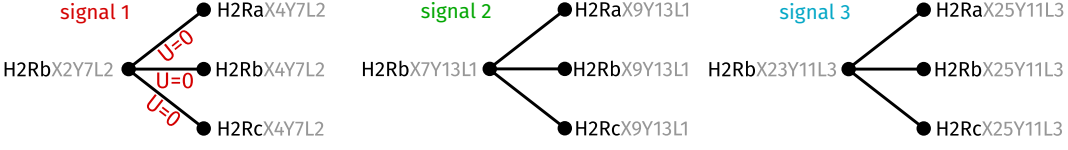
Avalanche Costs: Rationale

What if switch costs were inversely related to Usage?
(Usage is common to all instances of the same switch type)



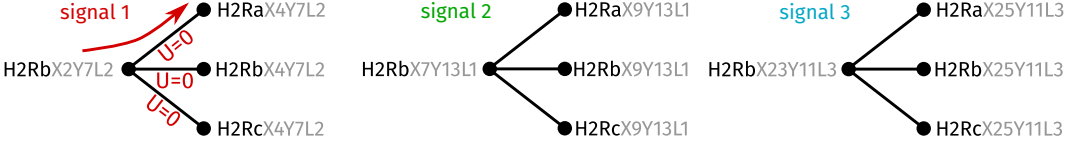
Avalanche Costs: Rationale

What if switch costs were inversely related to Usage?
(Usage is common to all instances of the same switch type)



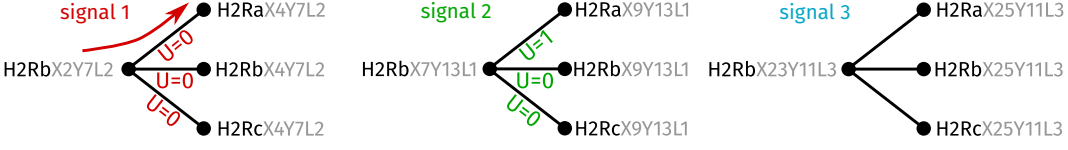
Avalanche Costs: Rationale

What if switch costs were inversely related to Usage?
(Usage is common to all instances of the same switch type)



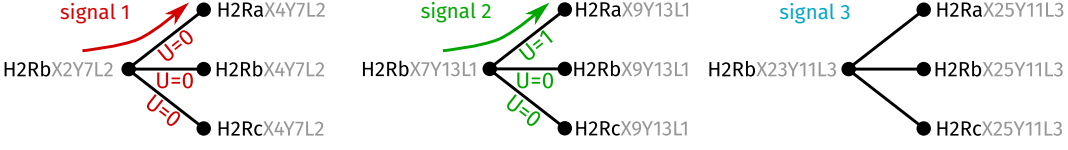
Avalanche Costs: Rationale

What if switch costs were inversely related to Usage?
(Usage is common to all instances of the same switch type)



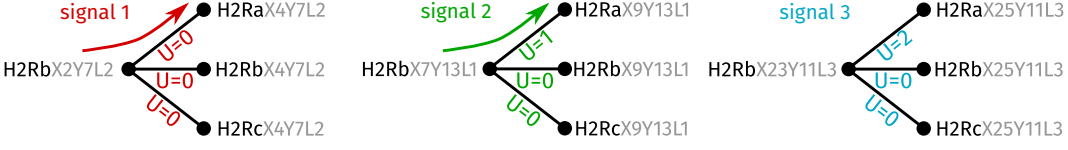
Avalanche Costs: Rationale

What if switch costs were inversely related to Usage?
(Usage is common to all instances of the same switch type)



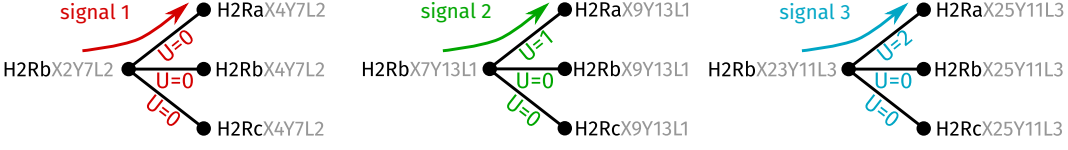
Avalanche Costs: Rationale

What if switch costs were inversely related to Usage?
(Usage is common to all instances of the same switch type)



Avalanche Costs: Rationale

What if switch costs were inversely related to Usage?
(Usage is common to all instances of the same switch type)



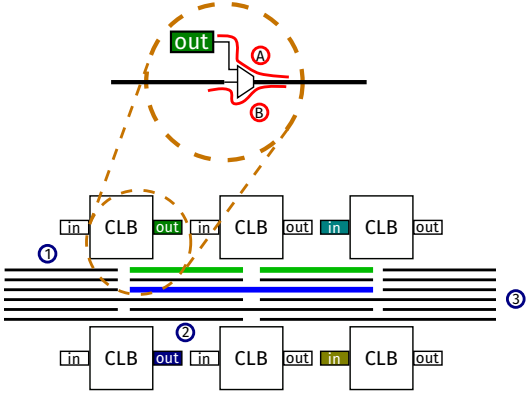
Avalanche Costs: Rationale

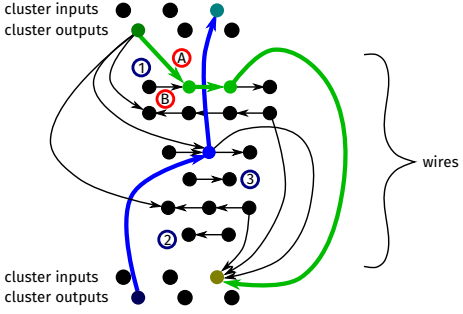
```
repeat
    foreach CONNECTION IN THE CIRCUIT do
        route using shortest path in the RR-graph;
        update node costs;
    end
    update node costs;
    throw away all routing (rip-up);
until OVERUSED NODES EXIST;
```

After rip-up, more signals move to switches with higher Usage, creating an avalanche effect

Avalanche Costs: Similarity with Congestion Negotiation

Signals negotiate which one will give up its desired nodes





Each node has a cost

Avalanche Costs: Similarity with Congestion Negotiation

Signals negotiate which one will give up its desired nodes

Circuit Routing with Pathfinder

SB-Pattern Design with Avalanche Costs

Avalanche Costs: Similarity with Congestion Negotiation

Signals negotiate which one will give up its desired nodes

Spreads the routes over more wire **instances**	
Circuit Routing with Pathfinder	SB-Pattern Design with Avalanche Costs

Avalanche Costs: Similarity with Congestion Negotiation

Signals negotiate which one will give up its desired nodes

Spreads the routes over more wire **instances**

$\Longrightarrow$ congestion-free routing

Circuit Routing with Pathfinder

SB-Pattern Design with Avalanche Costs

Avalanche Costs: Similarity with Congestion Negotiation

Signals negotiate which one will give up its desired nodes

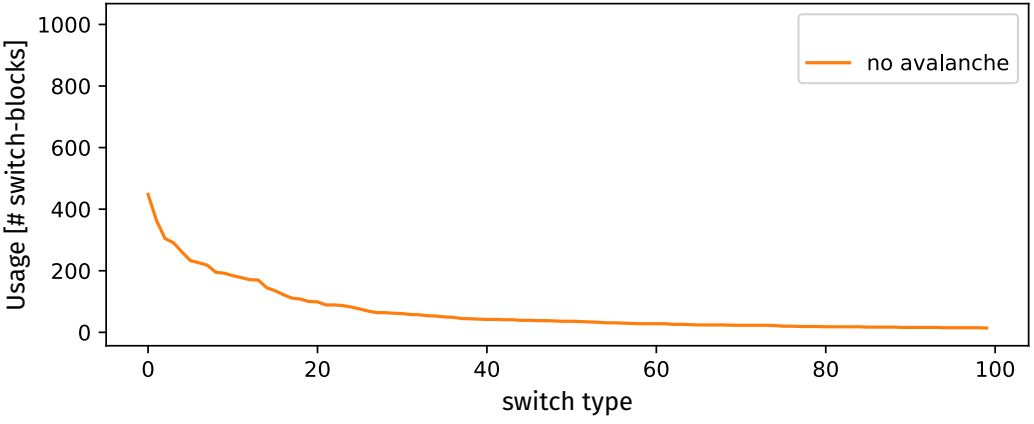
Circuit Routing with Pathfinder	SB-Pattern Design with Avalanche Costs
Spreads the routes over more wire **instances** $\Longrightarrow$ congestion-free routing	Concentrates the routes on fewer switch **types**

Avalanche Costs: Similarity with Congestion Negotiation

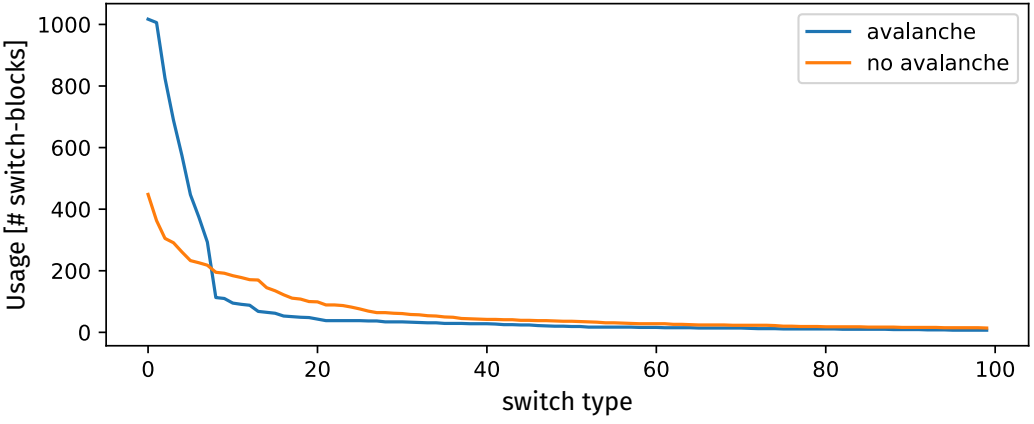
Signals negotiate which one will give up its desired nodes

Circuit Routing with Pathfinder	SB-Pattern Design with Avalanche Costs
Spreads the routes over more wire **instances**	Concentrates the routes on fewer switch **types**
$\Longrightarrow$ congestion-free routing	$\Longrightarrow$ optimizes the switch-pattern

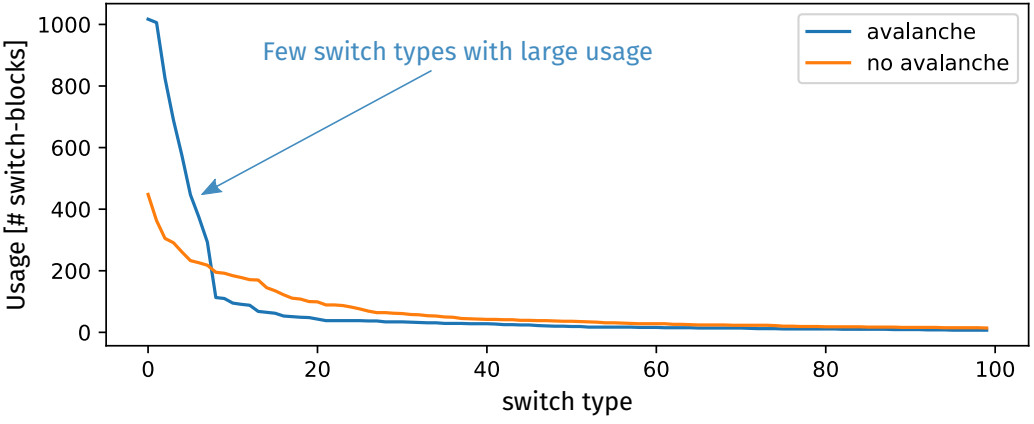
Avalanche Costs: Effects



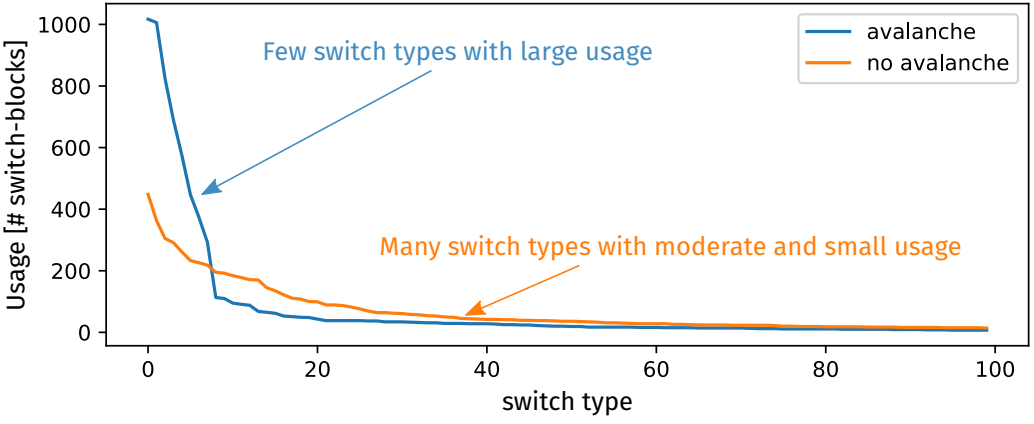
Avalanche Costs: Effects



Avalanche Costs: Effects



Avalanche Costs: Effects



The Complete Algorithm

Very similar to Simple Greedy:

1. Set the cost of all switch types not yet taken to some small cost ϵ.

2. Route all circuits.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.
4. Take all switch types with $Usage > Usage_{max}/\theta, \theta > 1$.
5. Set the cost of all taken switch types to 0.
6. If there are newly taken switch types, go to step 2.

The Complete Algorithm

Very similar to Simple Greedy:

1. Set the cost of all switch types not yet taken to ~~some small cost ϵ.~~ their starting avalanche cost.
2. Route all circuits.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.
4. Take all switch types with $Usage > Usage_{max}/\theta, \theta > 1$.
5. Set the cost of all taken switch types to 0.
6. If there are newly taken switch types, go to step 2.

The Complete Algorithm

Very similar to Simple Greedy:

1. Set the cost of all switch types not yet taken to ~~some small cost ϵ.~~ their starting avalanche cost.
2. Route all circuits updating avalanche costs.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.
4. Take all switch types with $Usage > Usage_{max}/\theta, \theta > 1$.
5. Set the cost of all taken switch types to 0.
6. If there are newly taken switch types, go to step 2.

The Complete Algorithm

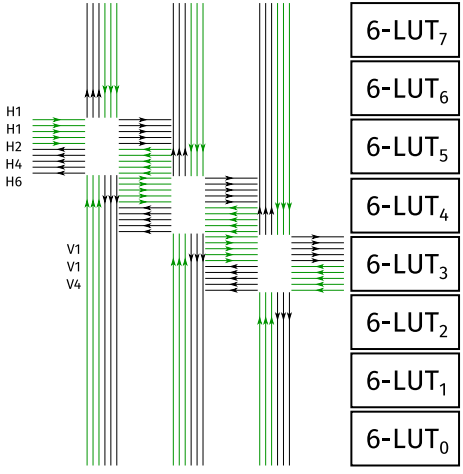
Very similar to Simple Greedy:

1. Set the cost of all switch types not yet taken to ~~some small cost ϵ.~~ their starting avalanche cost.
2. Route all circuits updating avalanche costs.
3. Find the maximum *Usage* of all switch types, $Usage_{max}$.
4. Take all switch types with $Usage > Usage_{max}/\theta, \theta > 1$.
5. Set the cost of all taken switch types to 0.
6. If there are newly taken switch types, go to step 2.

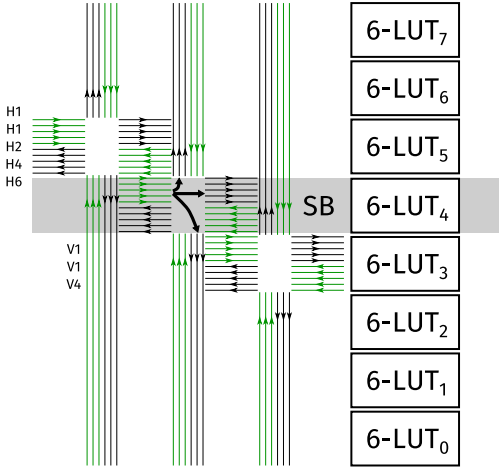
More details in the paper

Experimental Results

Experimental Setup

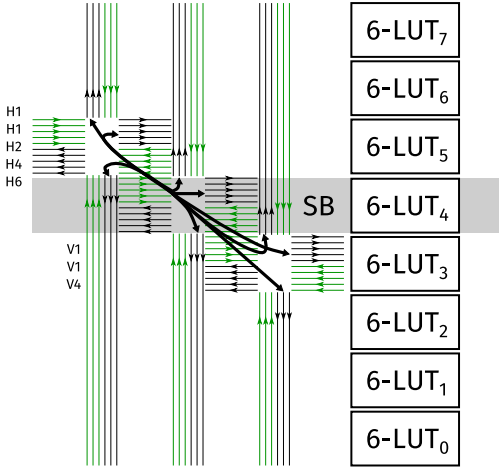


Experimental Setup



SB defined at LUT level

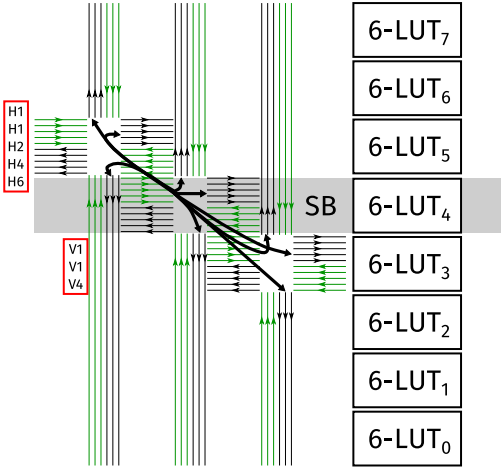
Experimental Setup



SB defined at LUT level

Switches allowed between adjacent LUTs

Experimental Setup



SB defined at LUT level

Switches allowed between adjacent LUTs

564 potential switch types

Comparison with Greedy

	avalanche			greedy		
#iterations	63			228		
#switches	93			438		
	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]
H1	5	5	14.5	31	25	23.1
H2	5	5	17.8	28	28	31.6
H4	8	7	25.9	21	27	43.2
H6	6	6	34.9	19	25	59.6
V1	7	7	22.2	38	31	35.5
V4	5	8	71.7	12	27	97.5
W(tile)	6816 nm			8904 nm		
CPD	1.40 ns			1.71 ns		

Comparison with Greedy

	avalanche			greedy		
#iterations	63			228		
#switches	93			438		
	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]
H1	5	5	14.5	31	25	23.1
H2	5	5	17.8	28	28	31.6
H4	8	7	25.9	21	27	43.2
H6	6	6	34.9	19	25	59.6
V1	7	7	22.2	38	31	35.5
V4	5	8	71.7	12	27	97.5
W(tile)	6816 nm			8904 nm		
CPD	1.40 ns			1.71 ns		

Comparison with Greedy

	avalanche			greedy		
#iterations	63			228		
#switches	93			438		
	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]
H1	5	5	14.5	31	25	23.1
H2	5	5	17.8	28	28	31.6
H4	8	7	25.9	21	27	43.2
H6	6	6	34.9	19	25	59.6
V1	7	7	22.2	38	31	35.5
V4	5	8	71.7	12	27	97.5
W(tile)	6816 nm			8904 nm		
CPD	1.40 ns			1.71 ns		

Comparison with Greedy

	avalanche			greedy		
#iterations	63			228		
#switches	93			438		
	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]
H1	5	5	14.5	31	25	23.1
H2	5	5	17.8	28	28	31.6
H4	8	7	25.9	21	27	43.2
H6	6	6	34.9	19	25	59.6
V1	7	7	22.2	38	31	35.5
V4	5	8	71.7	12	27	97.5
W(tile)	6816 nm			8904 nm		
CPD	1.40 ns			1.71 ns		

Comparison with Greedy

	avalanche			greedy			truncated greedy		
#iterations	63			228			62		
#switches	93			438			92		
	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]
H1	5	5	14.5	31	25	23.1	6	4	14.3
H2	5	5	17.8	28	28	31.6	7	6	18.4
H4	8	7	25.9	21	27	43.2	4	7	26.4
H6	6	6	34.9	19	25	59.6	2	7	35.9
V1	7	7	22.2	38	31	35.5	10	7	22.0
V4	5	8	71.7	12	27	97.5	2	5	67.8
W(tile)	6816 nm			8904 nm			7368 nm		
CPD	1.40 ns			1.71 ns			1.41 ns		

Comparison with Greedy

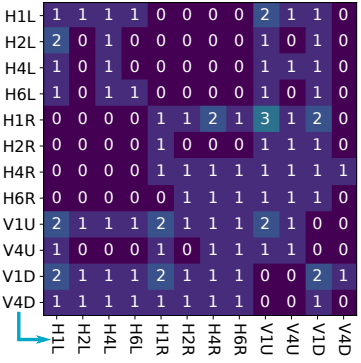
	avalanche			greedy			truncated greedy		
#iterations	63			228			62		
#switches	93			438			92		
	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]
H1	5	5	14.5	31	25	23.1	6	4	14.3
H2	5	5	17.8	28	28	31.6	7	6	18.4
H4	8	7	25.9	21	27	43.2	4	7	26.4
H6	6	6	34.9	19	25	59.6	2	7	35.9
V1	7	7	22.2	38	31	35.5	10	7	22.0
V4	5	8	71.7	12	27	97.5	2	5	67.8
W(tile)	6816 nm			8904 nm			7368 nm		
CPD	1.40 ns			1.71 ns			1.41 ns		

Comparison with Greedy

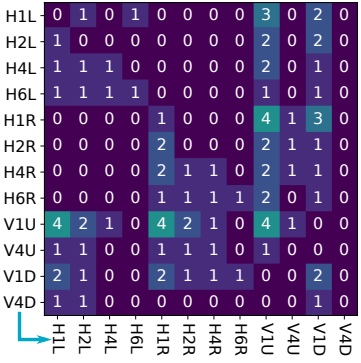
	avalanche			greedy			truncated greedy		
#iterations	63			228			62		
#switches	93			438			92		
	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]
H1	5	5	14.5	31	25	23.1	6	4	14.3
H2	5	5	17.8	28	28	31.6	7	6	18.4
H4	8	7	25.9	21	27	43.2	4	7	26.4
H6	6	6	34.9	19	25	59.6	2	7	35.9
V1	7	7	22.2	38	31	35.5	10	7	22.0
V4	5	8	71.7	12	27	97.5	2	5	67.8
W(tile)	6816 nm			8904 nm			7368 nm		
CPD	1.40 ns			1.71 ns			1.41 ns		

Avalanche vs Greedy: Switch Selection Choices

avalanche

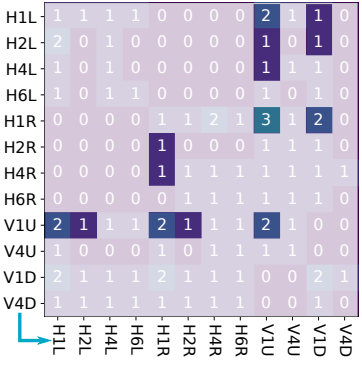


truncated greedy

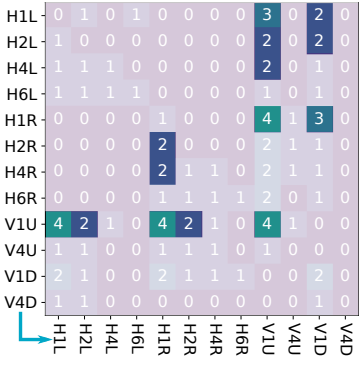


Avalanche vs Greedy: Switch Selection Choices

avalanche

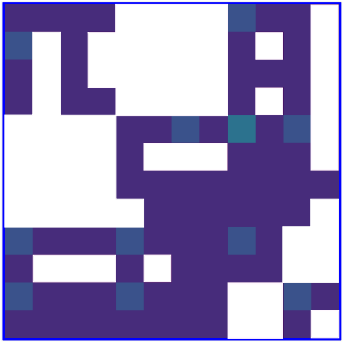


truncated greedy

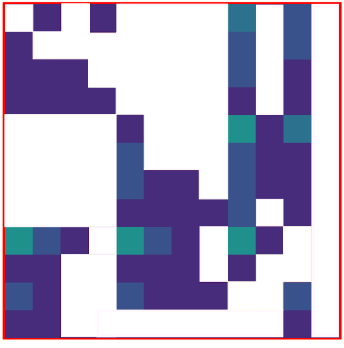


Avalanche vs Greedy: Switch Selection Choices

avalanche



truncated greedy



Avalanche vs Greedy: Routability

10 Gnl circuits

Rent's exponent = 0.7

10k LUT

avalanche	147	145	57	73	56	71	82	59	65	74
trunc. greedy	—	—	—	—	278	—	—	—	149	—

Avalanche vs Greedy: Routability

10 Gnl circuits

Rent's exponent = 0.7

10k LUT

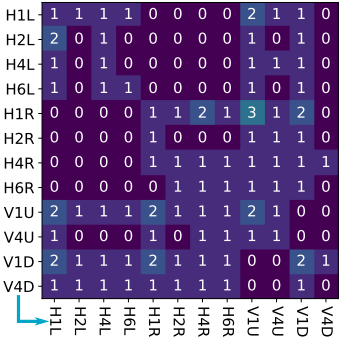
avalanche	147	145	57	73	56	71	82	59	65	74
trunc. greedy	—	—	—	—	278	—	—	—	149	—

Comparison with Simulated Annealing

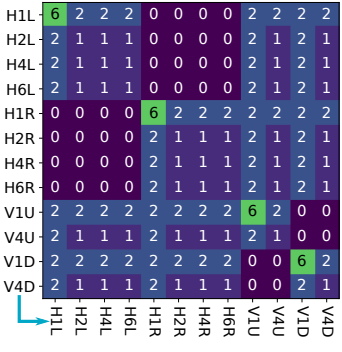
Inspired by M. Lin, J. Wawrzynek, and A. El Gamal,
“Exploring FPGA routing architecture stochastically”, TCAD’10

Comparison with Simulated Annealing: Starting Pattern

avalanche



ISFPGA'21



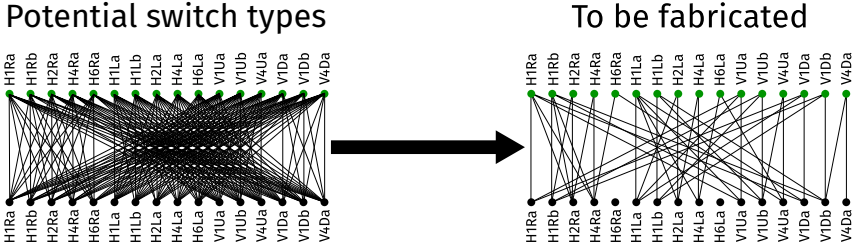
[1] S. Nikolić, F. Catthoor, Z. Tőkei, and P. Ienne,
“Global is the New Local: FPGA Architecture at 5nm and Beyond”, FPGA’21

Comparison with Simulated Annealing: Setup

- Each move is an exclusion/inclusion of one of the 564 switch types

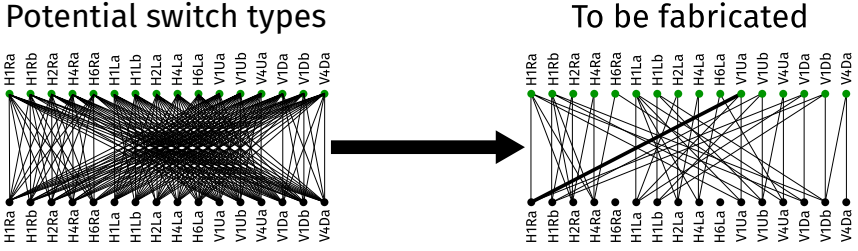
Comparison with Simulated Annealing: Setup

- Each move is an exclusion/inclusion of one of the 564 switch types



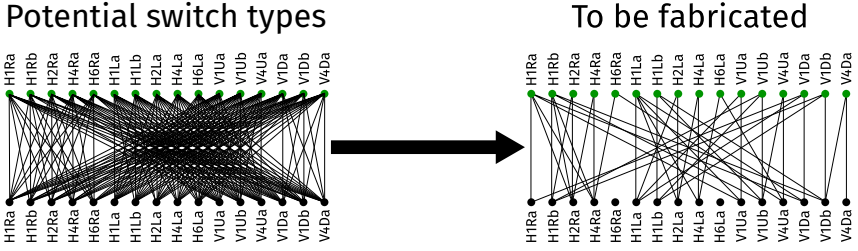
Comparison with Simulated Annealing: Setup

- Each move is an exclusion/inclusion of one of the 564 switch types



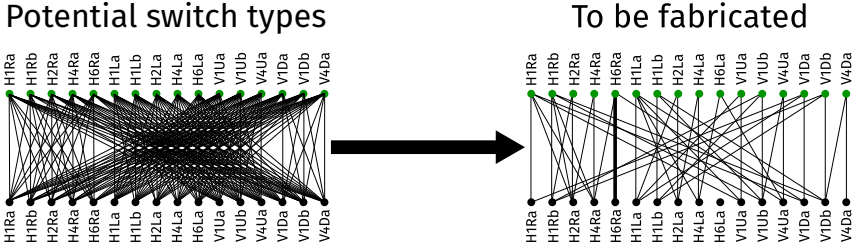
Comparison with Simulated Annealing: Setup

- Each move is an exclusion/inclusion of one of the 564 switch types



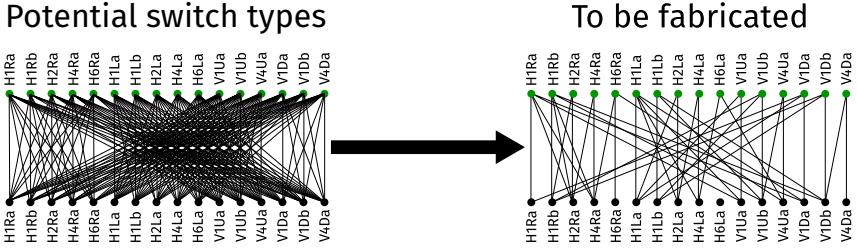
Comparison with Simulated Annealing: Setup

- Each move is an exclusion/inclusion of one of the 564 switch types



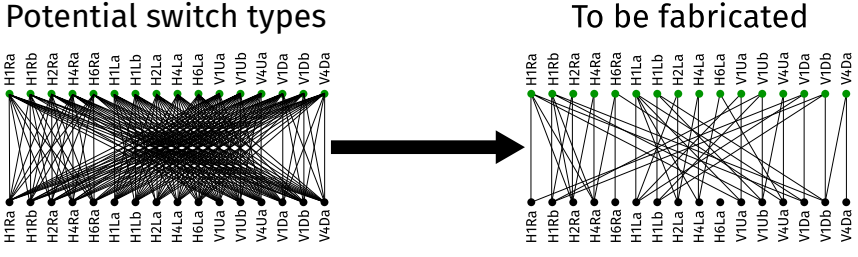
Comparison with Simulated Annealing: Setup

- Each move is an exclusion/inclusion of one of the 564 switch types



Comparison with Simulated Annealing: Setup

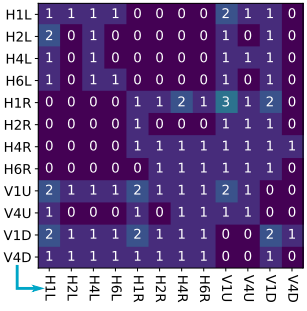
- Each move is an exclusion/inclusion of one of the 564 switch types



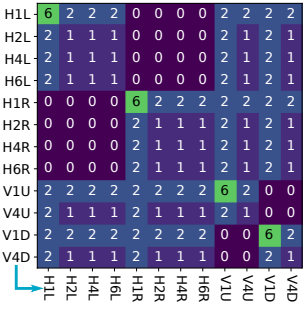
- cost function = f(CPD, tile area)

Comparison with Simulated Annealing: Outcome

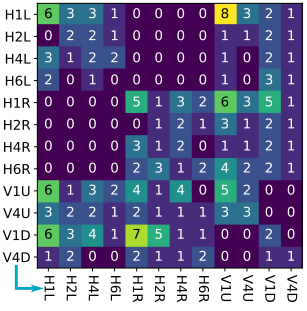
avalanche



ISFPGA'21



annealed



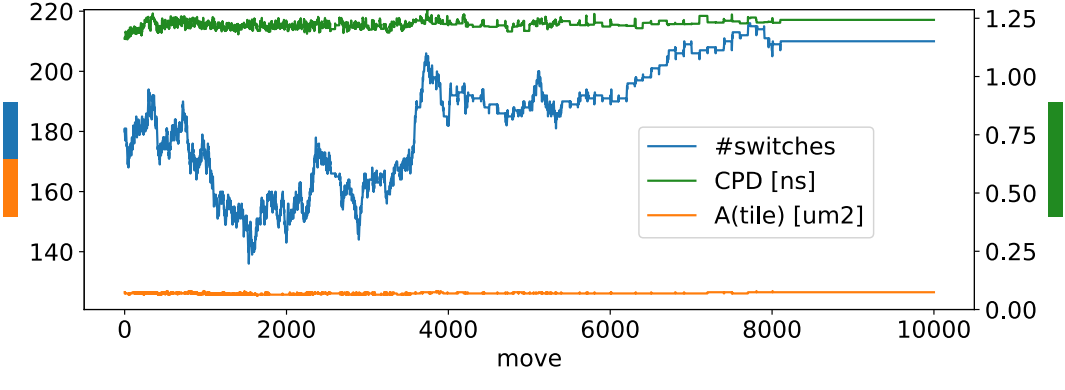
+ 30 switches

Comparison with Simulated Annealing: Outcome

	avalanche			initial			annealed		
#switches	93			180			210		
	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]	fi_{avg}	fo_{avg}	t_{avg} [ps]
H1	5	5	14.5	10	10	16.0	13	13	19.6
H2	5	5	17.8	11	11	21.3	14	11	24.1
H4	8	7	25.9	11	11	30.8	16	12	32.1
H6	6	6	34.9	11	11	43.1	9	13	47.3
V1	7	7	22.2	12	12	24.6	14	15	29.2
V4	5	8	71.7	13	13	74.3	13	15	86.8
W(tile)	6816 nm			7464 nm			7488 nm		
CPD	1.40 ns			1.46 ns			1.55 ns		

+ 10.7%

Simulated Annealing: Convergence



Conclusions and Future Work

FPGA routers can efficiently explore switch-block patterns

Future Work

Avalanche costs can be attributed to any node in any graph

Future Work

Avalanche costs can be attributed to any node in any graph

$\Longrightarrow$ use them to explore the entire routing architecture at once

Thank you for attention

https://github.com/EPFL-LAP/fpl21-avalanche