FastPass: Fast Pin Access Analysis with Incremental SAT Solving

Fangzhou Wang, Jinwei Liu, and Evangeline F.Y. Young

CSE Dept., The Chinese University of Hong Kong

March 27, 2023

香港中文大學
The Chinese University of Hong Kong

Outline

Outline

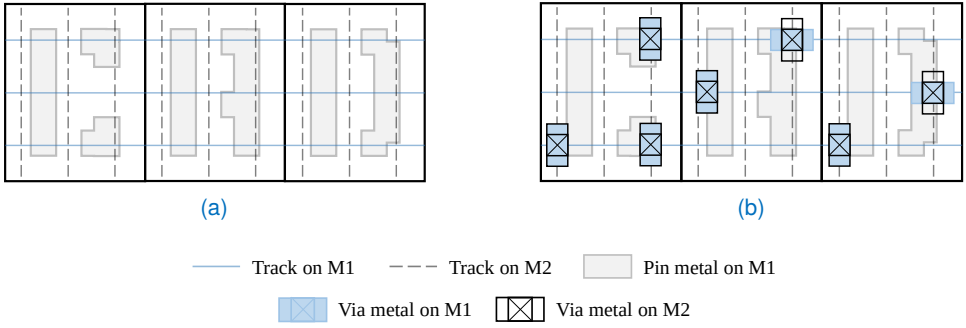
Introduction

Pin access is one of the most complicated sub-problems in VLSI routing.

- The pin shapes are complex polygons comprised of multiple rectangles.
- Over millions of pins might need to be analyzed simultaneously.
- The pin access solution of one pin should avoid causing violations with existing metals.
- The pin access solutions of different pins might interfere with each other.

Introduction

Here is an example pin access solution for a single cell instance.



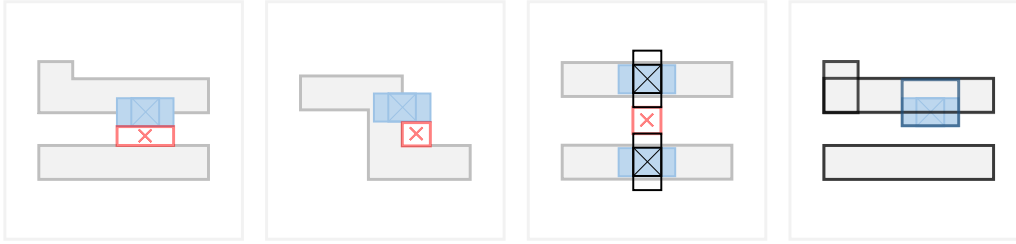
Outline

Problem Formulation

The Pin Access Problem:

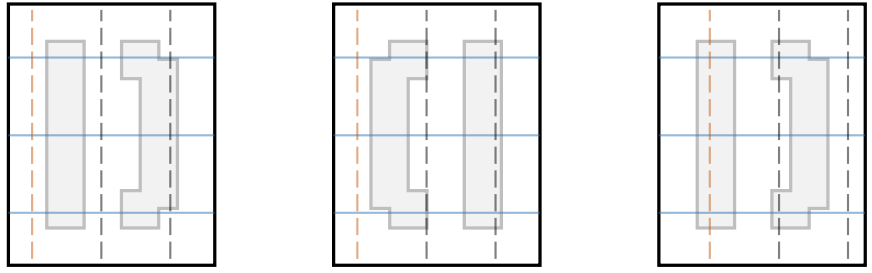
Given a design (with tracks, cell instances, nets, etc.), find a *routing scheme* such that each net pin is connected to a nearby grid point using *routes* (containing wires and vias), such that no *routes* will cause design rule violations with fixed metals or each other.

Design Rule Violations



Design Rule Check. (a) A PRL spacing violation between a routed via's metal and the fixed metal of a different pin. (b) A PRL spacing violation between a routed via's metal and the metal of the pin it attempts to access. (c) DRC violations between two routed vias' metals. (d) The maximal rectangles decomposed from fig. 1d.

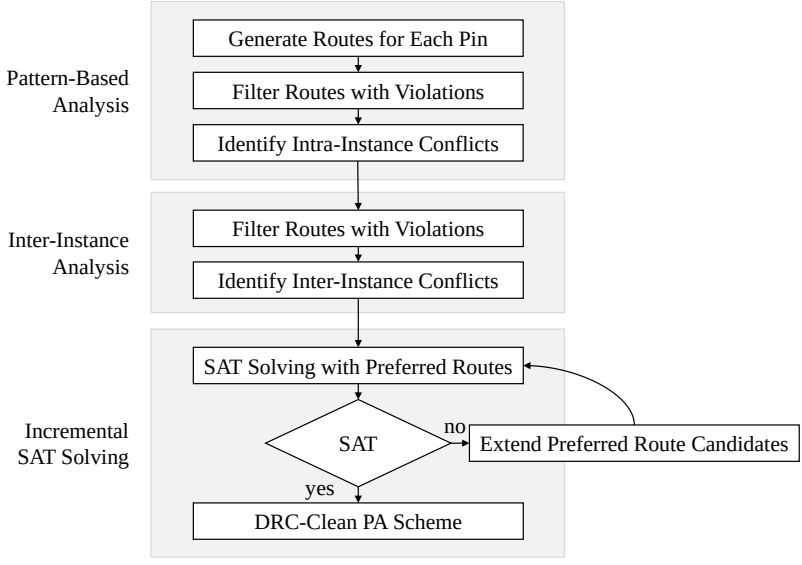
Instance Patterns



Instance Patterns. (a) An instance of the same instance pattern as the rightmost instance in Figure 1a. (b) An instance of a different instance pattern due to different orientation. (c) An instance of a different instance pattern due to different track offset.

Outline

Overview


Pattern-Based Analysis
Generate Routes for Each Pin
Filter Routes with Violations
Identify Intra-Instance Conflicts
Inter-Instance Analysis
Filter Routes with Violations
Identify Inter-Instance Conflicts
Incremental SAT Solving
SAT Solving with Preferred Routes
SAT
no
Extend Preferred Route Candidates
yes
DRC-Clean PA Scheme

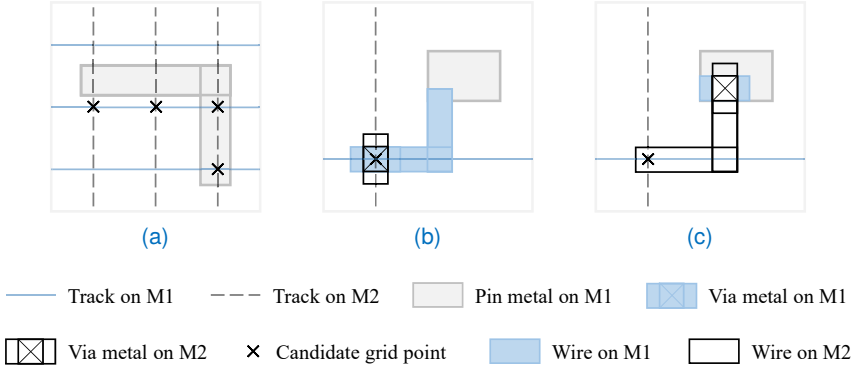
Candidate Routes Generation

We define a candidate *route* for a pin p as a potential set of routing segments that can connect p to a grid point on M2.
To generate *route* candidates for each pin:

- We first find a set of access grid points for each pin.
- For each access grid point, we will generate two kinds of *routes* using each via type.

Candidate Routes Generation

Two kinds of *routes*:



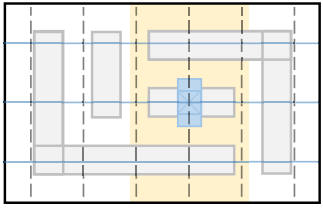
Candidate Routes Generation. (a) Pin access grid points. (b) Pin access *route* with an on-grid via. (c) Pin access *route* with an off-grid via.

Instance Pattern-Based Analysis

Goals:

- Filter *routes* that cause violations with the fixed metals inside the cell instance.
- Construct intra-instance conflict graph between *routes*.

Instance Pattern-Based Analysis



Intra-instance Design Rule Checking.

Algorithm Intra-Instance Design Rule Checking

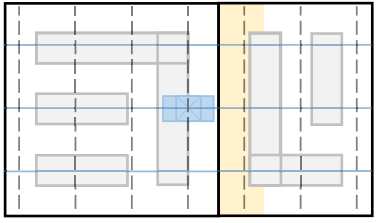
1: Sort all routed rectangles in non-decreasing order of x: $r_1, r_2, ... r_n$.
2: Sort all fixed rectangles in non-decreasing order of x: $R_1, R_2, ... R_N$.
3: $A \leftarrow \emptyset, j \leftarrow 1$
4: **for** $i = 1, 2, ..., n$ **do**
5: Shift the active region for routed rectangle r_i
6: **while** R_j has overlap with the active region **do**
7: $A \leftarrow A + \{R_j\}$
8: $j \leftarrow j + 1$
9: **for** each fixed rectangle $R' \in A$ **do**
10: **if** R' falls to the left of the active region **then**
11: $A \leftarrow A - \{R'\}$
12: **else**
13: Check violations between r_i and R'

Inter-Instance Analysis

Goals:

- Filter routes that cause violations with fixed metals in the neighboring cell instances.
- Construct inter-instance conflict graph between *routes* in different instances.

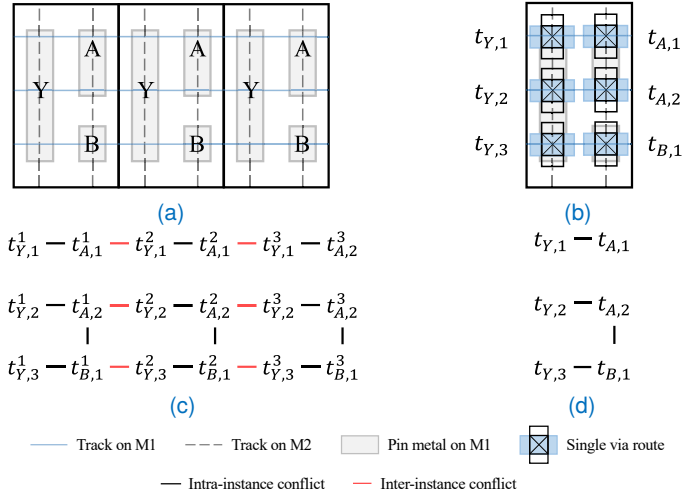
Inter-Instance Analysis



Inter-Instance Design Rule Checking

A *route* near the boundary of an instance only needs to be checked with the fixed metals and *routes* within a small active region (as colored yellow) near the boundary of its neighboring instance.

Inter-Instance Analysis



Conflict Graph. (a) Three identical cell instances in a row. (b) Possible routes for the pins in the instance. (c) Complete conflict graph. (d) Intra-instance conflict graph.

Route Selection by Increment SAT Solving

Goals:

- Select a *route* for each pin such that no *route* will have conflicts with another *route* or an existing fixed metal.
- If possible, optimize certain metrics such as the number of out-of-guide routes, HPWL, etc.

Boolean Formulation

$$\mathcal{P} \equiv \bigwedge_{p_i \in P} \bigvee_{t_{i,j} \in T_i} s_{i,j},$$

$$\mathcal{C} \equiv \bigwedge_{(t_{i_1,j_1}, t_{i_2,j_2}) \in E} (\neg\, s_{i_1,j_1} \vee \neg\, s_{i_2,j_2}).$$

Notations:

- P: the set of all pins.
- T_i: the set of candidate routes for pin p_i.
- $t_{i,j}$: the j^{th} candidate route for pin p_i.
- $s_{i,j}$: if route $t_{i,j}$ is selected.
- E: the set of all conflict edges.

Incremental SAT Solving

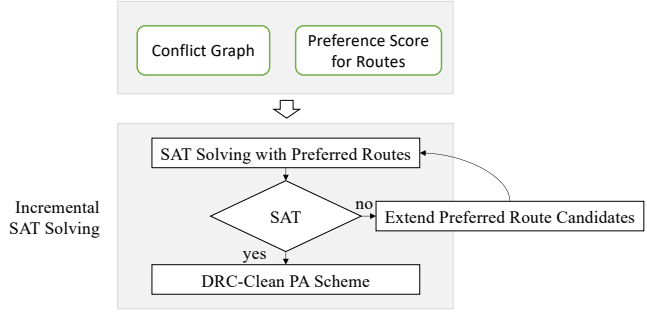
To prioritize the use of the candidate *routes* that have a better chance to give a better routing solution, we define the cost tuple $cost(t_{i,j}) = \langle c_1, c_2 \rangle$.

$$c_1 = \begin{cases} 1, & \text{if } ap(t_{i,j}) \text{ is out-of-guide,} \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

$$c_2 = L1Dist\big(ap(t_{i,j}), net(p_i).center\big), \tag{2}$$

Incremental SAT Solving

- We start the SAT solving with only the best *route* enabled for each pin (by using assumptions to disable other routes).
- If the solver reports SAT, we can stop the process and convert the assignment to an optimized pin access scheme.
- Otherwise, we will enable one more candidate *route* for each pin that caused UNSAT.



Outline

Experimental Results

Table: Benchmark and Runtime Statistics

Benchmarks	Tech (nm)	# Cell Instances	# Instance Patterns	# Pins	# Candidate Routes	# Conflict Edges	TOP[1] Time (s)	FastPass Time (s)	Speedup
ispd18_test1	45	8,879	182	17,202	90,601	4,928	6.3	**0.2**	38.3×
ispd18_test2	45	35,913	222	158,741	788,660	44,399	10.6	**1.0**	10.7×
ispd18_test3	45	35,973	227	159,579	798,128	44,348	13.6	**1.0**	14.1×
ispd18_test4	32	72,094	2,725	318,121	4,258,475	106,053	117.7	**5.4**	21.8×
ispd18_test5	32	71,954	2,733	318,059	3,492,551	7,554,606	125.7	**7.6**	16.6×
ispd18_test6	32	107,919	2,886	475,429	5,140,864	11,072,916	146.2	**10.2**	14.3×
ispd18_test7	32	179,865	148	793,129	8,442,950	16,884,133	98.3	**12.1**	8.1×
ispd18_test8	32	191,987	150	793,129	8,443,220	16,902,913	115.4	**12.6**	9.1×
ispd18_test9	32	192,911	136	791,761	8,416,299	16,907,027	72.9	**12.5**	5.9×
ispd18_test10	32	290,386	144	811,761	8,625,282	17,393,590	106.0	**13.7**	7.7×
Avg.	-	-	-	-	-	-	81.3	**7.6**	14.7×

[1]Kahng, Andrew B., Lutong Wang, and Bangqi Xu. "The tao of PAO: Anatomy of a pin access oracle for detailed routing." 2020 57th ACM/IEEE Design Automation Conference (DAC). IEEE, 2020.

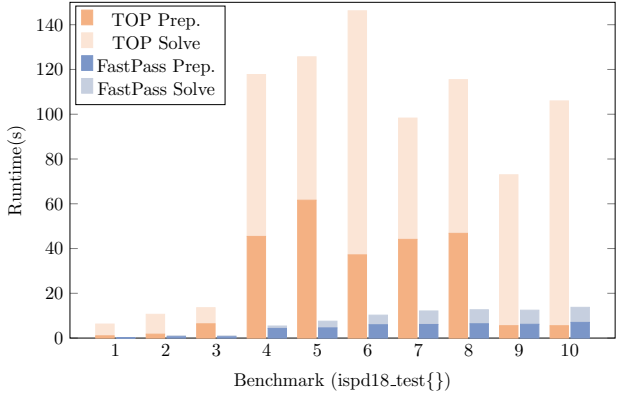
Effectiveness of Incremental SAT Solving

Table: Route Selection Results with/without Incremental SAT Solving.

Benchmarks	TOP		SAT-Unsorted		SAT-Sorted		Incremental SAT	
	# OFG *	HPWL	# OFG *	HPWL	# OFG *	HPWL	# OFG *	HPWL
ispd18_test1	860	62,754	860	62,772	33	61,532	8	61,507
ispd18_test2	7,188	1,338,360	8,888	1,338,622	1,010	1,322,727	688	1,322,142
ispd18_test3	6,992	1,457,780	7,606	1,458,023	1,595	1,442,733	1,092	1,442,262
ispd18_test4	50,207	2,158,830	38,004	2,158,717	2,132	2,124,095	1,810	2,123,646
ispd18_test5	46,445	2,164,850	46,148	2,164,636	3,389	2,132,656	2,093	2,131,595
ispd18_test6	73,750	2,942,230	73,964	2,942,051	5,422	2,894,805	3,483	2,893,202
ispd18_test7	107,177	5,054,720	97,597	5,054,326	2,806	4,970,566	1,444	4,968,595
ispd18_test8	108,038	5,076,690	98,554	5,076,310	2,874	4,992,606	1,434	4,990,678
ispd18_test9	130,448	4,495,240	122,556	4,494,891	4,088	4,411,584	2,637	4,409,740
ispd18_test10	136,333	5,621,630	127,737	5,621,370	4,433	5,537,226	2,785	5,535,221
geomean ratio	32.386	1.016	31.433	1.016	1.739	1.000	1.000	1.000

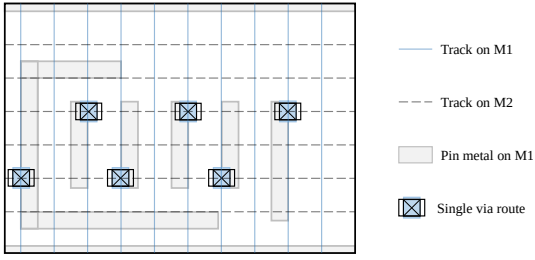
* OFG: Out-of-guide access grid points. The unit for HPWL is µm.

Runtime Decomposition



Runtime decomposition of TOP and FastPass.

For Advanced Nodes



Pin access result for an AOI221 cell in *mor1kx*

- we synthesize an open-source processor IP core *mor1kx*(80K instances, 515 instance patterns) with the ASAP 7nm library.
- With simple adaptation, it takes FastPass 4.1 seconds to generate DRC-clean pin access result.

Outline

Future Works

- We are trying to integrate the pin access analysis framework into a detailed router to further validate its effectiveness.
- We are also working on a dynamic pin access analysis flow based on FastPass, which is capable of making adjustments after seeing the actual routing result.

Thank You