FastPass: Fast Pin Access Analysis with Incremental SAT Solving

Fangzhou Wang*
fzwang@cse.cuhk.edu.hk
CSE Department
The Chinese University of Hong Kong
Hong Kong

Jinwei Liu*
jwliu@cse.cuhk.edu.hk
CSE Department
The Chinese University of Hong Kong
Hong Kong

Evangeline F. Y. Young
fyyoung@cse.cuhk.edu.hk
CSE Department
The Chinese University of Hong Kong
Hong Kong

ABSTRACT

Pin access analysis is a critical step in detailed routing. With complicated design rules and pin shapes, efficient and accurate pin accessibility evaluation is desirable in many physical design scenarios. To this end, we present FastPass, a fast and robust pin access analysis framework, which first generates design rule checking (DRC)-clean pin access route candidates for each pin, pre-computes incompatible pairs of routes, and then uses incremental SAT solving to find an optimized pin access scheme. Experimental results on the ISPD 2018 benchmarks show that FastPass produces DRC-clean pin access schemes for all cases while being 14.7× faster than the known best pin access analysis framework on average.

CCS CONCEPTS

• **Hardware → Wire routing**.

KEYWORDS

Physical design, detailed routing, pin access, boolean satisfiability

ACM Reference Format:
Fangzhou Wang, Jinwei Liu, and Evangeline F. Y. Young. 2023. FastPass: Fast Pin Access Analysis with Incremental SAT Solving. In *Proceedings of the 2023 International Symposium on Physical Design (ISPD '23), March 26–29, 2023, Virtual Event, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3569052.3571879

1 INTRODUCTION

With the scaling of technology nodes and higher levels of integration, standard cell pin access has become non-trivial with complicated design rules and off-grid pin shapes on lower metal layers [14].

Under such extreme circumstances, as an early step in the detailed routing flow, pin access analysis will be responsible for planning violation-free paths from pin shapes to some easily accessible grid points for each net and pin, such that in the later path searching stage, routing can end at some on-track locations without worrying about possible conflicts between pin access segments (i.e., metal wires and vias) and fixed metals (e.g., blockages, pin shapes, etc.).

Being a vital part of the detailed routing process that enables design rule checking (DRC)-clean routing solutions, a good pin access analysis framework (PAAF) can play important roles in other physical design scenarios too, including but not limited to detailed placement, global routing, and even placement and routing (P&R) co-optimization [3, 7, 9]. Ding et al. [3] propose a detailed placement refinement approach to improve pin accessibility, where the pin accessibility is modeled by a cost function, which does not consider specific design rules and can become more accurate with a pin access engine. TritonRoute-WXL [9] has a pin access engine [8] integrated to provide global routing with a better estimation of the routing resources consumed by pin access. The work of [7] describes an in-route placement refinement approach that conducts exact pin access analysis and greatly accelerates DRC convergence.

Various pin access approaches have emerged in recent years. Ozdal [15] proposes a multicommodity flow model to find escape routes for pins in dense clusters. Nieberg [14] proposes to first generate a set of design rule conforming paths and then search for conflict-free path sets by branch-and-bound. Xu et al. [16] propose to guide regular routing with standard cell pin access planning for better routability under self-aligned double patterning constraints. Dr. CU [10] uses L-shape routes and off-track vias to access hard-to-access pins. Recently, Kahng et al. [8] present a design rule-aware PAAF, which uses dynamic programming (DP) for both intra-cell and inter-cell pin access planning. We refer to the framework as TOP in the rest of this paper. An assumption is made for their DP-based algorithm that only adjacent pins in terms of x-coordinates could have conflicts, which does not always hold. Thus, each generated inter-cell pin access pattern must be thoroughly validated by a DRC engine, which significantly slows down the process.

Addressed by Kahng et al. [8], efficiency is considered one of the most important aspects of a PAAF since placement can change frequently during physical design steps like detailed placement, sizing, and buffering, where a large amount of inter-cell pin access analysis is required. Therefore, in this work, we present a boolean satisfiability (SAT)-based framework named FastPass for efficient and correct-by-construction pin access scheme generation. Our main contributions are summarized as follows.

- We introduce a pre-processing flow, which consists of instance pattern-based analysis and inter-instance analysis, to efficiently generate DRC-clean pin access route candidates for each pin and pre-compute inter-route conflicts.
- A novel sweep line-style algorithm is introduced for quick design rule checking and inter-route conflict detection, which makes the pre-processing flow highly efficient.

*Both authors contributed equally to this work

ISPD '23, March 26–29, 2023, Virtual Event, USA
© 2023 Association for Computing Machinery.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record will be published in *Proceedings of the 2023 International Symposium on Physical Design (ISPD '23), March 26–29, 2023, Virtual Event, USA*, https://doi.org/10.1145/3569052.3571879.

- We propose an SAT encoding for the conflict-free route selection problem and adopt the incremental SAT solving technique [13] to find a pin access scheme that is not only DRC-clean but also optimized towards custom objectives.
- We conduct experiments on the ISPD 2018 detailed routing contest benchmark suite [12]. Experimental results show that our proposed framework can produce DRC-clean pin access schemes for all cases while being on average 14.7× faster than the state-of-the-art PAAF (TOP) [8]. We also test FastPass on a 7nm design to demonstrate its scalability on advanced technology nodes.

The rest of the paper is organized as follows. In Section 2, we will define the problem and introduce some preliminary concepts. We introduce our methodology in Section 3 and discuss our experimental results in Section 4.

2 PRELIMINARIES

2.1 Problem Formulation

We formally define the problem of DRC-clean pin access scheme generation as follows.

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 DRC violations with fixed metals or each other.

Figure 1a shows a toy example with three simplified cell instances, and Figure 1b gives an example of a DRC-clean pin access scheme. The difficulty of this problem lies in: (1) The complex pin shapes must be analyzed to rule out routes having DRC violations with them; (2) The routes for different pins must be decided simultaneously to avoid DRC conflicts between them.

2.2 Instance Patterns

Similar to [8], we group the cell instances into instance patterns to avoid redundant analysis on instances with the same pattern. Two instances share the same pattern if they have the same master cell type, orientation, and track offset (the distance from the first in-cell track to the left boundary of the instance). For example, the instance in Figure 2a is considered of the same pattern as the rightmost instance in Figure 1, but the instances in Figure 2b and Figure 2c are not, due to different orientation and track offset respectively.

2.3 Design Rule Check

In this work, we mainly consider four types of DRC violations described in the ISPD 2018 detailed routing contest, namely metal short, parallel run length (PRL) spacing, end-of-line (EOL) spacing, and cut spacing [12]. In pin access, DRC violations mainly come from two sources, (a) violations between a routed segment (wires or vias) and a fixed metal, and (b) violations between different routed segments. Figure 3a shows a PRL spacing violation between a routed via and a different pin's metal on M1. Figure 3b shows another PRL spacing violation between a routed via and the metal of the pin it attempts to access. Figure 3c, on the other hand, shows an EOL spacing violation between two routed via's metals on M2.

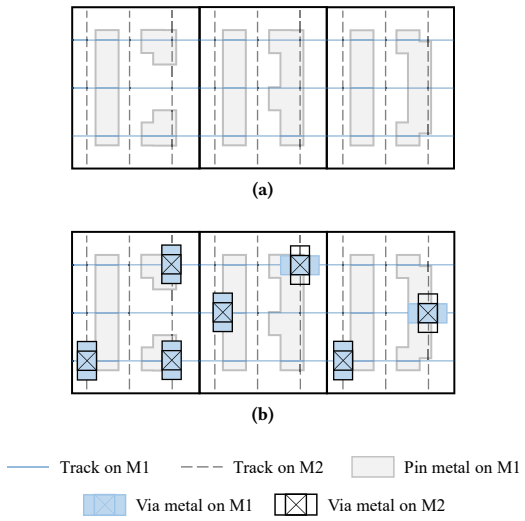

Figure 1: An Example Pin Access Problem.

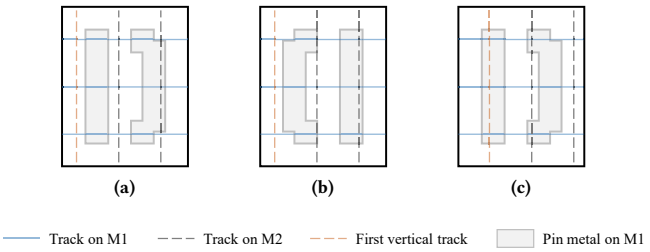

Figure 2: Instance Patterns. (a) An instance of the same instance pattern as the rightmost instance in Figure 1a. (b) An instance of a different instance pattern due to different orientation. (c) An instance of a different instance pattern due to different track offset.

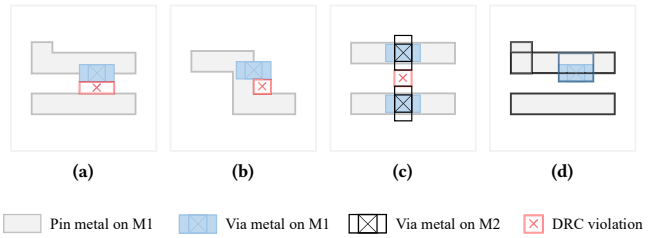

Figure 3: Design Rule Check. (a) A PRL spacing violation between a routed via's metal and the fixed metal of a different pin. (b) A PRL spacing violation between a routed via's metal and the metal of the pin it attempts to access. (c) DRC violations between two routed vias' metals. (d) The maximal rectangles decomposed from fig. 3a.

Moreover, almost all the design rules should be checked among maximal rectangles, i.e., the unique rectangles within a metal polygon, the width and height of which can not be expanded anymore.

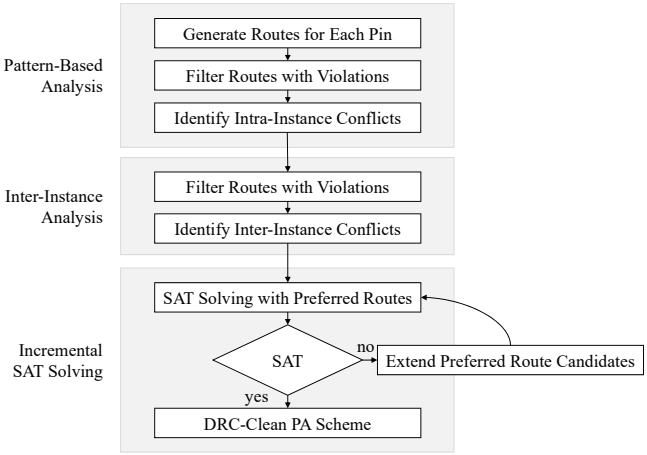

Figure 4: Overall Flow.

For example, the upper pin metal along with the via metal in Figure 3a can be decomposed into three maximal rectangles as in Figure 3d. The lower pin metal is a maximal rectangle itself. We call the maximal rectangles with black outline in Figure 3d fixed rectangles, as they are solely from fixed metals. We call the maximal rectangle with a blue outline a routed rectangle, as it is a newly formed maximal rectangle including some routed segments. Extracting the maximal rectangles of metal polygons is necessary because the calculation of the spacing needed for many design rules rely on the characteristics of the maximal rectangles, e.g., maximal width and maximal parallel run.

3 METHODOLOGY

3.1 Overview

The overall flow of the proposed algorithm is illustrated in Figure 4. To begin with, a pattern-based analysis is performed. For each instance pattern, a bunch of candidate routes is generated for each pin. A route for a pin connects the pin to a nearby grid point by a via and possibly a few wires. The candidate routes are filtered for the first time in pattern-based analysis to remove the routes that will cause DRC violations with the fixed metals inside the cell instance. The conflicts between routes for different pins in the same cell instance are identified to form an *intra-instance conflict graph*.

For inter-instance analysis, the candidate routes are filtered again to remove the routes that will cause DRC violations with the fixed metals in neighboring instances. The conflicts between routes in different cells are identified to form an *inter-instance conflict graph*.

Lastly, incremental SAT solving will be used to find an optimized DRC-clean pin access scheme. Each route for a pin is assigned a cost according to how likely it will bring more difficulty to detailed routing. To prioritize the preferred routes, only the route with the lowest cost for each pin is enabled at the beginning. If no solution can be found, the preferred candidate routes are extended progressively until a solution is found.

3.2 Instance Pattern-Based Analysis

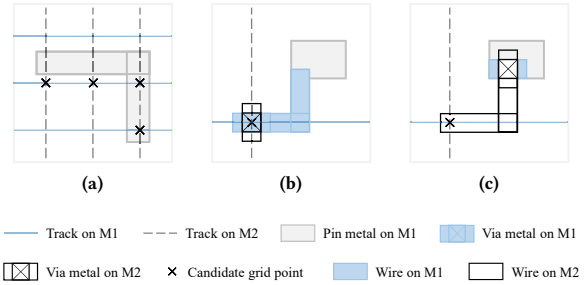

Figure 5: Candidate Routes Generation. (a) Pin access grid points. (b) Pin access route with an on-grid via. (c) Pin access route with an off-grid via.

3.2.1 Candidate Routes Generation. To analyze a unique instance pattern, we first find a set of access grid points on M2 for each pin. For each maximal rectangle of a pin shape, we include all the grid points it contains into the set GP. If it does not contain any grid point, we will include the grid points nearest to the rectangle into GP. For instance, four grid points are found for the pin shape having two maximal rectangles in Figure 5a.

After that, we generate two kinds of routes for each grid point ($ap \in GP$) using each via type. Note that we will only consider the via types whose metal on M2 shares the same direction of the tracks on M2. For the first kind of routes, we use a via to connect a grid point on M2 to M1 and then use two short wire segments on M1 to connect the center of the via to the nearest pin metal as shown in Figure 5b. The wire segments can be saved if the center of the via is already inside the pin metal. For the second kind of routes, we use wires on M2 to route from the grid point to a position over the pin metal so that a via can be dropped there and the M1 metal of the via is completely inside the pin metal as shown in Figure 5c.

In general, the first kind of routes is preferred as it uses the minimum routing resources on M2, but some pins are simply not accessible using the first kind, especially when the layout on M1 is too compact.

After generating the candidate routes, we examine all the routes in an instance pattern together to filter those having DRC violations with the fixed metals. To perform this violation checking, we will have to decompose all fixed metals and routed metals into maximal rectangles as discussed in Section 2.3. For brevity, we use rectangle to refer to maximal rectangle in the following paragraphs. A common approach to check DRC violations related to a rectangle r is to search for all the other rectangles within a certain distance from r using R-tree [6]. However, R-tree can be very slow when the number of rectangles increases and similar queries are conducted multiple times. Thus we adopt a completely different approach to check DRC violations.

Our algorithm utilizes a fact that the cell instances have a limited height, and a routed rectangle will cause violations with only a few fixed rectangles within a small distance in the x direction (horizontal) as the yellow region illustrated in Figure 6a. We call such a region the active region of a rectangle. The width of an active region can be calculated with the width of the rectangle and the maximum spacing requirement.

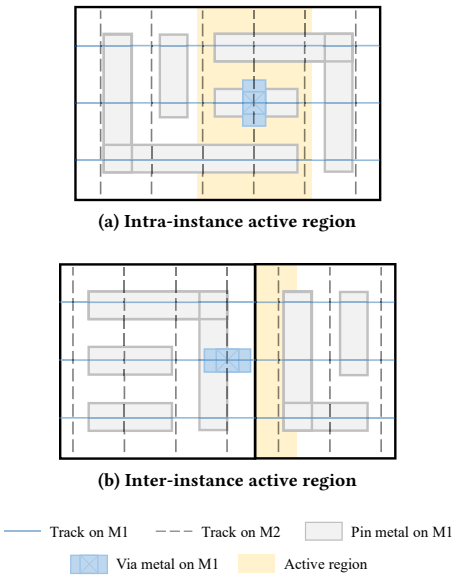

Figure 6: Intra-Instance and Inter-Instance Design Rule Checking. A routed rectangle will cause violations with only a few fixed rectangles within the active region.

Algorithm 1 Intra-Instance DRC Violation Check

```
Sort all routed rectangles in non-decreasing order of x: r_1, r_2, ... r_n.
Sort all fixed rectangles in non-decreasing order of x: R_1, R_2, ... R_N.
A ← ∅, j ← 1
for i = 1, 2, ..., n do
    Shift the active region for routed rectangle r_i
    while R_j has overlap with the active region do
        A ← A + {R_j}
        j ← j + 1
    for each fixed rectangle R' ∈ A do
        if R' falls to the left of the active region then
            A ← A − {R'}
        else
            Check violations between r_i and R'
```

With this fact, we check the DRC violations for all routes together by a sweep-line algorithm. The algorithm for checking the DRC violations between the routed rectangles and the fixed rectangles can be described as in Algorithm 1. We first sort the routed rectangles and fixed rectangles in non-decreasing order of their x coordinates respectively. We will then initialize the active set, i.e. the set of fixed rectangles within the active region, as an empty set ($A \leftarrow \emptyset$). The active region is initially outside the left boundary of the instance at the beginning. For each routed rectangle from left to right, we will shift the left and right boundary of the active region. The active set is updated by adding the fixed rectangles entering from the right and removing those leaving to the left of the active region. For each routed rectangle, we will only check it with the fixed rectangles within the corresponding active region. The routes having a rectangle with DRC violations will be filtered.

Assume that the number of fixed rectangles is N, the number of routed rectangles is n and the maximum number of routed rectangles in the active region does not exceed m, the time complexity of the above algorithm excluding the sorting step will be $O(N + nm)$. $O(N)$ time is needed to include and remove each fixed rectangle from the active set once, and $O(nm)$ time is for checking violations between each routed rectangle and the fixed rectangles in the corresponding active set. Note that N, n, and m are all small in a single cell instance, thus Algorithm 1 can be performed very efficiently.

3.2.2 Intra-Instance Conflict Graph. After filtering the routes having violations with the fixed metals, the conflicts (violations) between routes for different pins in the same instance will also be checked to build an *intra-instance conflict graph*. Similar to violation checking with fixed metals, a routed rectangle only needs to be checked with routed rectangles of other pins within the active region. The checking algorithm is similar to Algorithm 1, except that the active set will be used to maintain other routed rectangles in the active region instead of fixed rectangles. After the checking, an *intra-instance conflict graph* can be constructed. In the graph, each route is a vertex, and there will be an edge between two routes if they are from different pins and cannot be selected simultaneously.

3.3 Inter-Instance Analysis and Complete Conflict Graph

After instance pattern-based analysis, we get a bunch of routes and an *intra-instance conflict graph* for each cell instance. These routes will all be compatible with the fixed metals of the cell instance they belong to, and the *intra-instance conflict graphs* will tell the compatibility between routes in the same instance. Nonetheless, a route near the left or right boundary of an instance still has a chance to conflict with a fixed metal or another route in a neighboring instance, like the situation in Figure 6b. Fortunately, similar to intra-instance violation checking, the routes near the boundary of an instance only need to be checked with the fixed metals and routes within a small active region (as colored yellow in Figure 6b) near the boundary of its neighboring instance. All routes near the left or right boundary of an instance are checked in this way to filter those incompatible with the fixed metals in the neighboring instance, and edges will be added to the *inter-instance conflict graph*.

For better illustration, a toy example is given in Figure 7. The conflict graph involving real standard cells can be much more complicated than that. Figure 7a shows three cell instances of the same instance pattern in a row. After instance pattern-based analysis, we get six routes, each of which has a single via as shown in Figure 7b, and an *intra-instance conflict graph* as shown in Figure 7d. We use $t_{X,i}$ to denote the i^{th} route for pin X. After inter-instance analysis, suppose none of the existing routes are filtered, we will get the complete conflict graph shown in Figure 7c after adding the inter-instance edges in red color ($t^j_{X,i}$ denotes the i^{th} route for pin X of the j^{th} instance).

3.4 Route Selection by Incremental SAT Solving

With DRC-clean candidate routes and the complete conflict graph, we propose an incremental SAT-based approach for route selection,

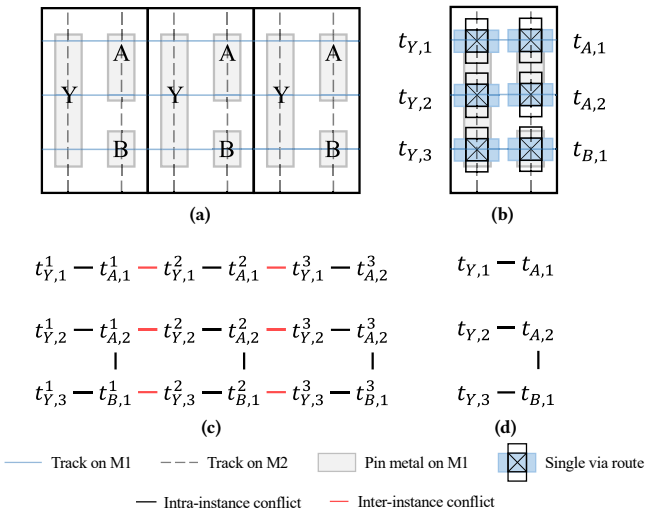

Figure 7: Conflict Graph. (a) Three identical cell instances in a row. (b) Possible routes for the pins in the instance. (c) Complete conflict graph. (d) Intra-instance conflict graph.

such that each net pin is assigned one route to form an optimal or close to optimal pin access scheme without any inter-route conflict.

3.4.1 Division into Independent Subproblems. It is observed that the route selection problem for the whole design can be divided into many independent subproblems, each of which covers only a small set of pins and can be quickly solved. We perform depth-first search (DFS) on the conflict graph to identify all the connected components and treat each of them as a single problem instance. To avoid cases where a pin's candidate routes appear in multiple subproblems, we add edges between routes from the same pin during DFS. The originally large problem can thus be solved efficiently. In the rest of this section, we focus on one subproblem for clarity.

3.4.2 Problem Definition with Notations. Consider the set of pins P, we denote T_i as the set of candidate routes for pin p_i and let $t_{i,j} \in T_i$ be the j^{th} candidate route of pin p_i. With the conflict graph $G = (V, E)$, a bijection (one-to-one mapping) exists between the set of routes and the set of vertices V. With an abuse of notation, for a route $t_{i,j}$, we denote its corresponding vertex in V as $t_{i,j}$, too.

To indicate our preference on different routes, each route $t_{i,j}$ has a cost tuple $cost(t_{i,j}) = \langle c_1, c_2 \rangle$, which will be calculated as follows.

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

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

where $ap(t_{i,j})$ means the access grid point of route $t_{i,j}$ and $net(p_i)$ represents the net to which pin p_i belongs. Note that the global route guides are a set of regions in which detailed routing is preferred. Therefore, by having c_1, we first consider routes whose grid points lie in guides. With the term c_2, we prefer to have routes that are closer to the bounding box center of the net, which can lead to a shorter total half-perimeter wire length (HWPL).

We assume that in each T_i, routes have been sorted in lexicographic order (e.g., $\langle 0, 500 \rangle < \langle 0, 1000 \rangle < \langle 1, 200 \rangle$). In the route selection problem, we want to select a set of routes AS, such that

(1) for each pin $p_i \in P$, there must be a route $t_{i,k} \in AS$;
(2) for all $t_{i_1,j_1} \neq t_{i_2,j_2} \in AS$, we require that the two routes do not conflict with each other, i.e., $(t_{i_1,j_1}, t_{i_2,j_2}) \notin E$;
(3) we want to use preferred routes (i.e., the first few candidates) as much as possible.

3.4.3 Boolean Formulation for Route Selection. For this problem, we propose a neat Boolean formula $\mathcal{F}$ with two types of constraints,

$$\mathcal{F} \equiv \mathcal{P} \wedge \mathcal{C}. \tag{3}$$

In the above equation, $\mathcal{P}$ enforces that at least one route should be chosen for each pin. $\mathcal{C}$ encodes from the conflict graph that two incompatible routes cannot be both selected at the same time. With a set of Boolean variables S, where $s_{i,j} \in S$ is asserted true if route $t_{i,j}$ is selected, constraints $\mathcal{P}$ and $\mathcal{C}$ are formulated as follows.

$$\mathcal{P} \equiv \bigwedge_{p_i \in P} \bigvee_{t_{i,j} \in T_i} s_{i,j}, \tag{4}$$

$$\mathcal{C} \equiv \bigwedge_{(t_{i_1,j_1}, t_{i_2,j_2}) \in E} (\neg\, s_{i_1,j_1} \vee \neg\, s_{i_2,j_2}). \tag{5}$$

By feeding $\mathcal{F}$ into an SAT solver, we can easily get a solution which corresponds to a DRC-clean and conflict-free pin access scheme if there exists one. Moreover, in case multiple access schemes are needed for better detailed routability, we can simply add the negation of the current variable assignment to $\mathcal{F}$ as a new clause and conduct SAT solving again to get a different solution.

However, as an SAT solver does not consider any optimization goal, the resulting pin access scheme may contain a large number of access grid points that are out-of-guide or relatively far away from the centers of nets, which is undesirable for detailed routing. To address this problem, we propose an incremental SAT solving flow, which can produce a solution that is optimal or close to optimal.

3.4.4 Incremental SAT Solving. Introduced in Minisat [4], *incremental SAT solving with assumptions* allows us to pass a set of assumptions (in the form of unit clauses) that holds for a single invocation of the SAT solver. With this feature, users can (i) enable/disable certain clauses or (ii) pre-assign some variables according to the needs [13]. As a result, incremental SAT solving can be much more powerful than general SAT solving.

When the solver is called under assumptions, it either (i) reports satisfiable (SAT) and produces a solution that satisfies all the assumptions or (ii) reports unsatisfiable (UNSAT). In the latter situation, the solver will return a subset of assumptions that are sufficient to make the current invocation UNSAT [5], which can be used to build new assumptions for the next incremental run.

3.4.5 Incremental SAT Flow for Route Selection. Consider Equation (3), the SAT solver has a good chance of picking less preferred routes because such assignments exist in the search space. With assumptions, we can manually adjust the search space of an SAT problem instance. For example, by having the assumption $\neg s_{i,j}$, we can eliminate feasible solutions where $s_{i,j}$ is assigned 1. The usage of route $t_{i,j}$ is thus disabled.

Table 1: Benchmark and Runtime Statistics

Benchmarks	Tech (nm)	# Cell Instances	# Instance Patterns	# Pins	# Candidate Routes	# Conflict Edges	TOP 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×

As shown in Figure 4, we propose the following incremental SAT solving approach, which can be considered as an iterative process in which we dynamically adjust the search space of an SAT problem.

(1) We start the SAT solving with only the best route enabled for each pin (by using assumptions to disable other routes).
(2) If the solver reports SAT, we can stop the process and convert the assignment to an optimized pin access scheme.
(3) Otherwise, there must be some insolvable conflicts between the routes in the current search space. Therefore, we will find a set of pins P' that need to be extended based on the returned assumptions and enable one more candidate route for each pin in P'. In this way, we increase the chance to get a feasible solution in the next incremental run while introducing as few suboptimal routes as possible.

Note that if the solver reports UNSAT when no assumptions are passed to the solver (i.e., no routes are disabled), we can conclude that no feasible solutions exist for the route selection problem.

Compared with normal SAT solving, the proposed incremental flow can lead to several incremental runs of an SAT solver, each of which has a relatively small search space. Therefore, in practice, the latter one will not be slower though the SAT solver can be invoked several times. Most importantly, by gradually enlarging the search space, we can finally get an optimal or close to optimal solution with an SAT solver only, instead of resorting to more time-consuming methods like MAX-SAT or integer linear programming.

4 EXPERIMENTAL RESULTS

We implement FastPass in C++ language, and MiniSat 2.2 [4] is used for SAT solving. All the experiments are conducted on a Linux server with a 2.90 GHz Intel Xeon CPU. To demonstrate the effectiveness and scalability of our proposed PAAF, we conduct a series of experiments on the ISPD 2018 benchmark suite [12], which contains realistic industrial designs with up to 290K standard cell instances and 182K nets. The statistics of the benchmarks can be found in Table 1. The same table also shows the statistics of the conflict graph for each design constructed by FastPass. For most designs, FastPass generates around ten candidate routes for each pin on average, and the number of conflict edges found is about two times the number of candidate routes.

4.1 Comparison with TOP

We first compare our framework with the known best PAAF - TOP[1] [8]. As both frameworks can be easily parallelized, a single thread is used in this experiment for a fair comparison. Quantitative results are shown on the right-hand side of Table 1. Both TOP and FastPass can achieve DRC-clean pin access schemes with no failed pins. Moreover, it can be observed that our proposed framework is on average 14.7× faster than TOP, which shows that FastPass can be highly preferred in scenarios where placement changes frequently and pin access analysis needs to be called repeatedly.

4.2 Runtime Decomposition

Both the runtime of TOP and FastPass can be roughly divided into two parts - preparation time and solving time. For preparation, the unique instance patterns are analyzed, and DRC-clean access points and routes are generated. For solving, a DRC-clean pin access scheme is generated by DP in TOP and by incremental SAT in FastPass. The actual runtime decomposition of the two algorithms is shown in Figure 8. It is shown that our preparation with sweep-line design rule checking is much faster than TOP for most cases. Though SAT-based methods are usually considered slower, our solving is extremely fast due to the fact that most of the independent subproblems are relatively small. Besides, since our pin access scheme is correct by construction, no additional DRC checking is required compared to the DP-based algorithm in TOP.

4.3 Effectiveness of Incremental SAT Solving

As discussed in Section 3.4, we can resort to incremental SAT solving to optimize the quality of the pin access scheme, rather than merely select a legal route for each pin. In this work, we use two metrics to evaluate the quality of a pin access scheme - the number of out-of-guide access grid points (OFG) and the total half-perimeter wire length (HPWL) of the nets. Since detailed routers often prioritize path searching within the route guides, too many OFGs are likely to prolong the path searching process and even worsen the routing

[1]The source code is available at https://github.com/ABKGroup/TritonRoute-WXL.

Table 2: 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.

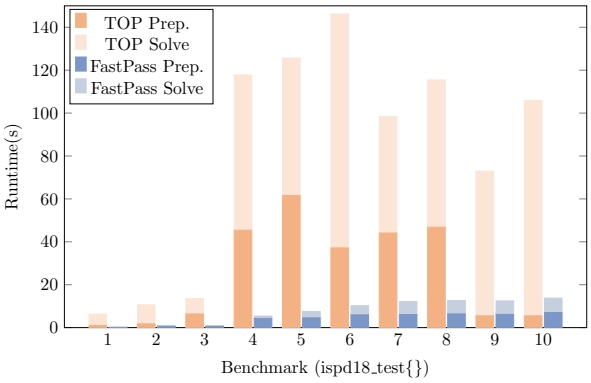

Figure 8: Runtime decomposition of TOP and FastPass.

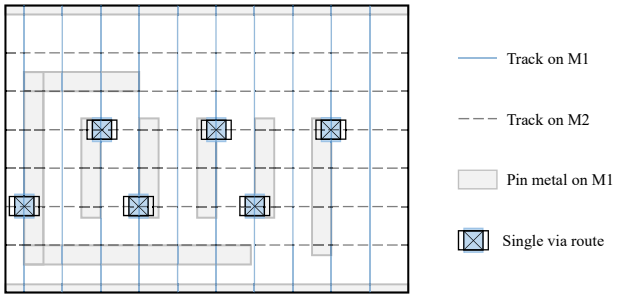

Figure 9: Pin access result for an AOI221 cell in *mor1kx* [1].

quality. The total HPWL, on the other hand, determines the lower bound of the total wire length, and thus could be optimized as well.

In Table 2, we compare the quality of the pin access scheme found by TOP and FastPass. For FastPass, we compare three different strategies. For the first strategy (SAT-Unsorted), we will feed the candidate routes into the SAT solver without sorting them beforehand. With this strategy, FastPass generates pin access schemes of similar quality with respect to the number of OFGs and total HPWL. For the second strategy (SAT-Sorted), we will sort the candidate routes in increasing order of their costs as discussed in Section 3.4.2. Since the SAT solver tends to select the option it is first given, sorting beforehand helps to reduce the number of OFGs by around 94.5% and reduce the total HPWL by around 1.6%. For the last strategy, we will use incremental SAT for route selection. Experimental results show that it further reduces the number of OFGs by around 42.5% compared to SAT-Sorted.

4.4 Supporting Advanced Technology Nodes

To verify the effectiveness of our approach on advanced technology nodes, we synthesize an open-source processor IP core *mor1kx* [1] (80K instances, 515 instance patterns) with the ASAP 7nm library [2]. With simple adaptation[2], it takes FastPass 4.1 seconds to generate DRC-clean pin access result. Figure 9 shows the pin access scheme for an AOI221 cell from *mor1kx*.

5 CONCLUSIONS

In this paper, we present FastPass, a highly efficient pin access analysis framework, including a sweep-line style algorithm for efficiently design rule checking, an incremental SAT-based algorithm for correct-by-construction route selection. Experimental results show that FastPass can produce DRC-clean pin access schemes for all benchmarks in the ISPD 2018 detailed routing contest, with merely 6.8% the running time of the state-of-the-art approach.

In the future, we plan to integrate FastPass into a detailed router to further validate its effectiveness. Besides, we will also support more complex design rules in the ISPD 2019 benchmark suite [11]. It will be interesting to model design rules that involve more than two objects with a variation of our current approach.

ACKNOWLEDGMENTS

The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 14209320).

[2]Necessary modifications are made to FastPass to deal with different design rules in the advanced technology node. For example, we add the design rule check for the end-of-line keepout spacing rule and rule out all routes with off-grid routing segments.

REFERENCES

[1] [n.d.]. mor1kx - an OpenRISC processor IP core. https://github.com/openrisc/mor1kx.

[2] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. 2016. ASAP7: A 7-nm finFET predictive process design kit. *Microelectronics Journal* 53 (2016), 105–115.

[3] Yixiao Ding, Chris Chu, and Wai-Kei Mak. 2017. Pin accessibility-driven detailed placement refinement. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*. 133–140.

[4] Niklas Eén and Niklas Sörensson. 2003. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*. Springer, 502–518.

[5] Henri Fraisse and Dinesh Gaitonde. 2018. A SAT-based timing driven place and route flow for critical soft IP. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 8–87.

[6] Antonin Guttman. [n.d.]. R-trees: A dynamic index structure for spatial searching. In *SIGMOD, 1984*. 47–57.

[7] Andrew B Kahng, Jian Kuang, Wen-Hao Liu, and Bangqi Xu. 2021. In-Route Pin Access-Driven Placement Refinement for Improved Detailed Routing Convergence. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 3 (2021), 784–788.

[8] Andrew B Kahng, Lutong Wang, and Bangqi Xu. 2020. The tao of PAO: Anatomy of a pin access oracle for detailed routing. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[9] Andrew B Kahng, Lutong Wang, and Bangqi Xu. 2021. TritonRoute-WXL: The Open-Source Router With Integrated DRC Engine. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 4 (2021), 1076–1089.

[10] Haocheng Li, Gengjie Chen, Bentian Jiang, Jingsong Chen, and Evangeline FY Young. 2019. Dr. CU 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–7.

[11] Wen-Hao Liu, Stefanus Mantik, Wing-Kai Chow, Yixiao Ding, Amin Farshidi, and Gracieli Posser. 2019. ISPD 2019 initial detailed routing contest and benchmark with advanced routing rules. In *Proceedings of the 2019 International Symposium on Physical Design*. 147–151.

[12] Stefanus Mantik, Gracieli Posser, Wing-Kai Chow, Yixiao Ding, and Wen-Hao Liu. 2018. ISPD 2018 initial detailed routing contest and benchmarks. In *Proceedings of the 2018 International Symposium on Physical Design*. 140–143.

[13] Alexander Nadel and Vadim Ryvchin. 2012. Efficient SAT solving under assumptions. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 242–255.

[14] Tim Nieberg. 2011. Gridless pin access in detailed routing. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 170–175.

[15] Muhammet Mustafa Ozdal. 2009. Detailed-routing algorithms for dense pin clusters in integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 3 (2009), 340–349.

[16] Xiaoqing Xu, Bei Yu, Jhih-Rong Gao, Che-Lun Hsu, and David Z Pan. 2016. PARR: Pin-access planning and regular routing for self-aligned double patterning. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 21, 3 (2016), 1–21.