This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3066528, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

In-Route Pin Access-Driven Placement Refinement for Improved Detailed Routing Convergence

Andrew B. Kahng, Jian Kuang, Wen-Hao Liu and Bangqi Xu

***Abstract*—Pin access is increasingly important in advanced nodes. Neighboring or cell-boundary pins can have degraded pin accessibility, causing design rule violations (DRCs) during routing, which are runtime-expensive to resolve. Conventional physical design tool flow uses pessimistic and/or inaccurate understanding of pin access during the placement stage and keeps the location of cells fixed during routing. This can leave pin access issues unsolvable and block further routing solution improvement. The timeliness of our present work is confirmed by the recent ICCAD-2020 CAD Contest, Problem B formulation from Synopsys, Inc. [11]. The organizers give a succinct motivation for what we study – to eliminate preserved margins and misalignment issues from conventional placement models. In this work, we develop an *in-route*, pin access-driven local placement refinement. Experiments across industry designs in a wide range of advanced technology nodes confirm that our optimization can significantly improve routing convergence (i.e., subsequent detailed routing runtime and initial detailed routing DRCs). Our optimization can reduce congestion and wirelength without timing degradation.**

I. Introduction

In advanced technology nodes, detailed routing is a critical challenge. With smaller feature sizes, more and more complex design rules are introduced with each new technology enablement. Such complex design rules make detailed routing evermore challenging. This is especially true with respect to *pin access*, where the detailed router aims to achieve DRC-clean connection to complicated pin shapes with comprehension of not only design rules, but intra-cell and inter-cell pin shape interactions as well.

Pin accessibility, which measures ease or difficulty of pin access, is an important measurement of routability. Pin accessibility assessment and modeling have been widely studied in recent works. Xu et al. [9] propose a dynamic hit point scoring strategy to dynamically assess pin accessibility based on the number of pin access points described in [8]. Seo et al. [6] propose a metric (i.e., Inaccessibility of Cell) to describe pin accessibility considering the number of access points and interference among access points based on a pre-defined threshold. Ding et al. [2] define a pin access region for each standard cell pin and propose a pin access penalty function based on distance and visibility between two pin access regions. Yu et al. [10] propose a deep learning-based pin pattern recognition methodology for pin accessibility prediction and optimization. In this work, we adopt the concept of *pin access pattern*, i.e., combination of pin access points, as in [4].

Many recent works focus on pin access-aware detailed placement optimizations. [2] proposes a two-phase pin accessibility-driven detailed placement refinement. The first phase performs cell flipping and swapping adjacent cells, and the second stage performs cell movement. Taghavi et al. [7] propose a local congestion metric considering local pin accessibility and apply a suite of detailed placement techniques, MILOR, to mitigate local congestion. Chow et al. [1] propose a two-step global-local move detailed placement algorithm to minimize wirelength considering cell density. In summary, these works attempt to optimize pin accessibility during placement stage with modeled pin accessibility information – in similar fashion as in a conventional physical design tool flow.[1]

In a conventional physical design tool flow, the placement tool uses models including cell density, pin density, etc. to estimate the pin accessibility. However, such estimation can be inaccurate, if not misleading. Figure 1(a) and Figure 1(b) have the same placement solution attributes (i.e., same cell and pin density) while the relative locations between pin shapes and track locations differ. The pin accessibility in Figure 1(a) is better than the pin accessibility in Figure 1(b) because of more on-grid access points, especially for pin A near the left cell boundary. A placement solution with poor pin accessibility, or poor pin accessibility correlation with routing, can cause a considerable amount of initial detailed routing design rule violations (DRCs).[2] Although detailed routers have the capability to iteratively fix DRCs, a large number of initial detailed routing DRCs can lead to (i) many iterations and long runtimes needed for DRC convergence, if the detailed router can converge on DRC at all; and (ii) longer routed wirelength due to detours needed to fix DRCs.

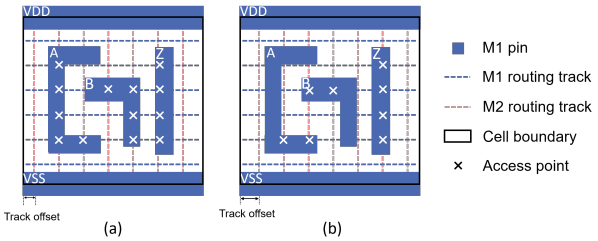

Fig. 1: Pin access points with different routing track-placement site offsets.

In this work, we propose an *in-route*, pin access-driven local placement refinement using a hybrid wirelength model with timing awareness (i.e., an objective function with weighted sum of timing and wirelength components). With application of our optimization, a leading commercial P&R tool's detailed routing runtime can be significantly reduced as we attain improved initial detailed routing DRCs and final routed wirelength without timing degradation. To our knowledge, ours is the first detailed placement framework that comprehends *exact pin access*, precisely as the detailed router understands this. Our main contributions are summarized as follows.

- We propose a dynamic programming (DP) based approach to minimize a cost function that is aware of pin accessibility, wirelength and timing while also considering EEQ cell swapping for advanced technology nodes.
- We propose a pin accessibility model that comprehends interactions between neighboring standard cell instances.

[1]We adopt the widely-studied ordered-row placement in this work for scalability consideration.

[2]According to [5], the detailed routing can be divided into two steps. The initial detailed routing step handles the major routing rules; then, the detailed routing refinement step fixes the remaining complicated DRCs.

- With integration of our optimization, the commercial P&R tool's detailed routing runtime can be reduced by up to 31.82% (avg. 15.06%). Moreover, our optimization results in up to 10.13% initial detailed routing DRC reduction, across 19 industry designs using various technology nodes.

The remainder of this paper is organized as follows. Section II describes our problem formulation. Section III details our pin access-driven detailed placement optimization flow. Section IV presents our experimental setup and results. Section V gives conclusions.

II. Pin Access-Driven Placement Optimization

We now present our problem statement and formulation for pin access-driven detailed placement optimization.

Pin Access-Driven Optimization Problem. *Given an initial legalized placement with pin access, perturb the placement to minimize overall cost.*

Inputs: A legal initial placement with pin access and a cost function considering pin accessibility, wirelength and timing.
Output: Optimized detailed placement with minimized overall cost (including pin accessibility).
Constraints: Maximum displacement range and placement legality.

We make the following observation and assumption with respect to this problem statement.

Observation. *Each cell row can be separated from each other from its neighboring cell rows in terms of pin accessibility.*

In a technology library that contains only single-height standard cells, the observation is obvious since the pin access points are well separated by VDD/VSS power rails inside the standard cell. For technology libraries that contain multi-height standard cells, the observation follows from the fact that shapes of standard cells are usually large. Hence, the pin accessibility inside the multi-height cell is usually decent, and instances of multi-height cells are unlikely to have pin access conflicts with their neighboring cell instances.

Assumption. *The locations of multi-height cell and clock-related instances are fixed.*

In advanced technology nodes, there are cells that span more than one standard cell row. Such multi-height cells usually are complex functional cells (e.g., flip-flops). Displacement of flip-flops in a well-optimized placement solution can introduce dramatic degradation in timing, especially because flip-flops are on critical timing paths. Similarly, in a well-optimized design, clock buffers are placed to satisfy various clock-related constraints (e.g., clock skew).

A. Notations

Table I shows the notations we use in our formulation. For each cell instance c_i, **cell instance index** i indicates it is the i^{th} left-to-right cell instance in the cell row from the initial placement. We use C to denote the set of cell instances in a row of the initial placement.

In order to honor the initial placement solution, we define a **displacement range** with x_Δ, in units of placement sites, to limit the amount of placement perturbation. We use x_i to denote the absolute x-coordinate, in units of placement sites, of c_i in the initial placement. Therefore, the absolute x-coordinate of c_i in the final placement solution must be within $[x_i - x_\Delta, x_i + x_\Delta]$. We use the variable $l \in [-x_\Delta, x_\Delta]$ to describe the **displacement** of a cell instance from its initial placement.

TABLE I: Notations.

Notation	Meaning
C	set of cells in a row of the initial placement
c_i	i^{th} cell in the left-to-right ordered initial placement, where i is *the cell index*
$[-x_\Delta, x_\Delta]$	displacement range
x_i	absolute x-coordinate of c_i in the initial placement, in units of placement sites
l	displacement of a cell from its location in the initial placement, in units of placement sites
$d[i][l]$	dynamic programming table entry indicating that the i^{th} cell is placed with displacement l from its initial location

We use a node array $d[i][l]$ as the underlying structure for a dynamic programming recurrence. Each node represents a unique placement location of cell instance c_i with displacement l. The information contained in each node is summarized in Table II. $nodeCost$ indicates the cost of placing the cell instance at the location implied by the node displacement index l. $pathCost$ is the lowest accumulated path cost from the first cell instance of the row to the current node. $pattern$ contains the information of the pre-selected best pin access pattern for the current node. Note that although pin access pattern, in theory, can be one dimension of the dynamic programming, we pre-select the pin access pattern for a cell instance at each possible location defined by displacement range in order to reduce runtime. We detail the pin access selection procedure in Section III. $prevNode$ points to the previous cell instance node with the lowest path cost.

TABLE II: Dynamic programming node notations.

Notation	Meaning
$nodeCost$	cost of placing the cell instance at the location indicated by the node
$pathCost$	lowest accumulated path cost to the current node
$pattern$	pre-selected pin access pattern for the current node
$prevNode$	pointer to the previous node with lowest path cost

B. Dynamic Programming (DP) Formulation

In our DP formulation, cell instances of a given cell row are placed sequentially from left to right in the same order as in the initial placement. Algorithm 1 describes our DP-based, pin access-driven detailed placement optimization procedure. Lines 2–3 populate DP nodes with pre-selected pin access patterns. Procedure $getPattern$, as described in Algorithm 2, pre-selects the pin access pattern for cell instance c_i with displacement l. Lines 4–5 initialize the pathCost of each node. Lines 6-16 describe the core part of the DP algorithm. Starting with the first cell instance, the algorithm sequentially places cell instances based on the recursive relation as described in Lines 8–14. Lines 9–11 prune placement solutions with overlapped cells. Procedure $cost$ computes the placement cost between partial placement solutions described by $d[i][l]$ and $d[i+1][l']$. Lines 17–21 find and return the solution with the lowest overall cost. The overall runtime of Algorithm 1 scales as $\mathcal{O}(|C|x_\Delta^2)$ but is negligible in practice (see Table V below).

The procedure $cost(\begin{smallmatrix} i+1,l' \\ i,l \end{smallmatrix})$ calculates the cost of placing c_{i+1} with displacement l' as a weighted sum of (i) wirelength cost $cost_{WL}$ and (ii) pin access cost $cost_{PA}$ as shown in Equation 1. We use a wirelength weighting factor $\alpha \in [0, 1]$ to balance the tradeoff between wirelength and pin accessibility.

$$cost(\begin{smallmatrix} i+1,l' \\ i,l \end{smallmatrix}) = \alpha \cdot cost_{WL}(i+1, l') + (1-\alpha) \cdot cost_{PA}(\begin{smallmatrix} i+1,l' \\ i,l \end{smallmatrix}) \quad (1)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3066528, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Algorithm 1 Dynamic programming

1: **Inputs**: dynamic programming array d, track patterns *tps*, access patterns array *APs*
2: **Initialize pin access pattern for all nodes** ($l \in [-x_\Delta, x_\Delta]$)
3: $d[i][l].pattern \leftarrow getPattern(APs[i], tps, d[i][l])(0 < i \leq |C-1|)$
4: **Initialize cost for all nodes**
5: $d[0][l].pathCost \leftarrow 0$, $d[i][l].pathCost \leftarrow +\infty, (0 < i < |C|)$
6: **for all** $i = 0$ to $|C| - 1$ **do**
7: **for all** $d[i][l].pathCost \neq +\infty$ **do**
8: **for all** $l' \in [-x_\Delta, x_\Delta]$ **do**
9: **if** isOverlap($d[i][l]$, $d[i+1][l']$) **then**
10: continue
11: **end if**
12: $t \leftarrow d[i][l].pathCost + cost(^{i+1,l'}_{i,l})$
13: $d[i+1][l'].pathCost \leftarrow \min(d[i+1][l'].pathCost, t)$
14: **end for**
15: **end for**
16: **end for**
17: $finalCost \leftarrow \infty$
18: **for all** $d[|C|-1|][l]$ **do**
19: $finalCost \leftarrow \min(d[|C-1|][l].pathCost, finalCost)$
20: **end for**
21: **Return** $finalCost$

III. Flow

We now describe the overall flow of our pin access-driven placement optimization, along with key elements including pin access pattern selection and cost function calculations.

A. Pin Access Pattern Selection

In our work, DRC-clean pin access patterns comprehending all pin shapes are an input from the detailed router's integrated pin access analysis engine. In order to enable efficient and accurate wirelength calculation, we perform pin access selection prior to our *in-route* detailed placement optimization. In advanced, especially sub-7nm, technology nodes, electrically-equivalent (EEQ) cell swapping is necessary for cell instance movement in order to ensure alignment between colored routing tracks and pin shapes. Therefore, for each DP node with displacement $l \neq 0$, we pre-calculate the best EEQ cell to use for the given placement site prior to pin access pattern selection.

For all DP nodes representing cell instances at original location (i.e., l=0), we preserve the pre-defined access patterns which are already optimized for wirelength and pin accessibility. For all DP nodes with displacement $l \neq 0$, we perform the pin access pattern selection as described in Algorithm 2.

The main goal of Algorithm 2 is to preemptively avoid potential design rule violations between access patterns from neighboring cell instances.[3] Line 3 sorts the access patterns based on their on-gridness with respect to the routing tracks. The one with the most on-gridness is less likely to conflict with its neighboring cells. Line 4 initializes the best access pattern to the access pattern with the most on-grid access points. Lines 5–15 iterate through all access patterns. Lines 6–7 check whether the current DP node (i.e., $d[i][l]$), which represents the placement of the current instance, overlaps with the original placement solution of the next instance (i.e., $d[i+1][0]$). If the two nodes overlap, we return the access pattern with the most on-grid access points. Lines 9–10 check whether the current access pattern conflicts with the access pattern of the next instance at its original location. Lines 11–13 return the best access pattern that does not conflict with the access pattern of the next instance.

[3]We only check the right neighbor of an instance as DP propagates from left to right, since conflicts between neighboring access patterns are symmetric.

Algorithm 2 Pin access pattern selection *getPattern*

1: **Inputs**: access patterns *aps*, track patterns *tps*, dynamic programming node $d[i][l]$
2: **Output**: best access pattern ap_{best} for $l \neq 0$
3: aps = sort(aps, tps)
4: ap_{best} = $aps[0]$
5: **for all** access pattern $ap \in aps$ **do**
6: **if** isOverlap($d[i][l]$, $d[i+1][0]$) **then**
7: break
8: **else**
9: **if** isConflict(ap, $d[i+1][0].pattern$) **then**
10: continue
11: **else**
12: ap_{best} = ap
13: break
14: **end if**
15: **end if**
16: **end for**
17: **return** ap_{best}

B. Cost Functions

As mentioned in Section II, we consider two cost components in our formulation (i.e., $cost_{WL}$ and $cost_{PA}$).

Wirelength cost. The wirelength cost $cost_{WL}$ for a node $d[i][l]$ depends only on the placement of the cell instance c_i itself. The wirelength cost is calculated based on Equation 2.

$$cost_{WL}(i,l) = \sum_{pin \in c_i} (m(pin,l) - m(pin,0)) \quad (2)$$

In order to prevent timing degradation, we use a hybrid wirelength metric, as shown in Equation 3, with timing awareness where (i) $dist_{CG}$ denotes distance to the center of gravity of pins of the net which the given pin belongs to, and (ii) $dist_{driver}$ denotes distance to the driver pin of the net which the given pin belongs to.

$$m(pin,l) = min(dist_{CG}(pin,l), dist_{driver}(pin,l)) \quad (3)$$

Pin access cost. The pin access cost $cost_{PA}(^{i+1,l'}_{i,l})$ is calculated based on the boundary pin cost $cost_{BP}$ between cells c_i and c_{i+1}, having corresponding displacements l and l', as shown in Equation 4. For c_i and c_{i+1}, pin access conflicts could occur between the pins near the cell boundaries. We use a threshold distance $dist_{thres}$ to query the boundary pins and check potential conflicts between boundary pins. Figure 2 illustrates the boundary pins defined by $dist_{thres}$.

$$cost_{PA}(^{i+1,l'}_{i,l}) = cost_{BP}(^{i+1,l'}_{i,l}) - cost_{BP}(^{i+1,0}_{i,0}) \quad (4)$$

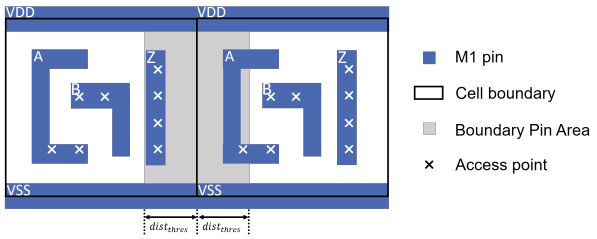

Fig. 2: Illustration of boundary pins with threshold distance $dist_{thres}$.

We calculate the boundary pin cost using Equation 5 where $BP_R(i)$ (resp. $BP_L(i+1)$) indicates the right boundary pins of c_i (resp. left boundary pins of c_{i+1}).

$$cost_{BP}(^{i+1,l'}_{i,l}) = \sum_{\substack{pin_1 \in BP_R(i) \\ pin_2 \in BP_L(i+1)}} cost_{pin(pin_1,pin_2)} \quad (5)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3066528, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

We calculate the pairwise pin access cost for the boundary pin pairs from c_i and c_{i+1} using Equation 6, where *PAP(pin)* represents the set of pin access points of a boundary pin and $dist$ calculates the distance between two boundary pins.

$$cost_{pin}(pin_1, pin_2) = \frac{|PAP(pin1)| \cdot |PAP(pin2)|}{dist(pin_1, pin_2)} \quad (6)$$

C. Overall Flow

Figure 3 illustrates the difference between the conventional routing flow and the routing flow with our proposed placement optimization. The red box implements what we refer to as *in-route* detailed placement refinement. We take an initial placement solution and pin access patterns as inputs. We then select the access pattern for each DP node. Next, based on the (already well-optimized) placement solution, we use DP to further optimize pin accessibility and wirelength.

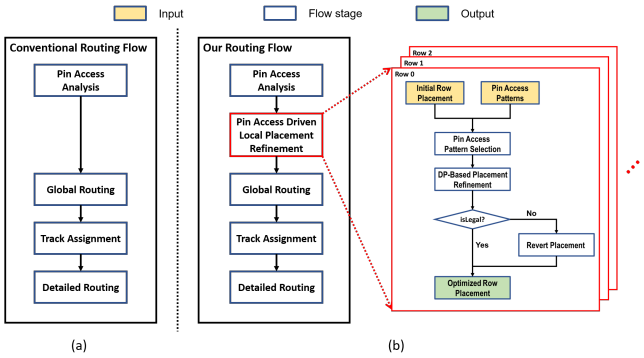

Fig. 3: Illustrations of (a) conventional routing flow, and (b) our proposed routing flow with *in-route* pin access-driven detailed placement refinement.

In advanced technology nodes, placement legality constraints are much more extensive and complex than cell non-overlapping (e.g., cell edge spacing constraint). Such constraints can cause placement violations between non-neighboring cell instances (e.g., c_{i-1} and c_{i+1}). Therefore, although we utilize the placement API from the commercial tool to check placement legality between neighboring cell instances, for each cell row, we perform row-based placement legality check after optimization and revert our changes to the row placement if any placement violation is observed.

IV. Experiments

We integrate our *in-route* detailed placement optimization with a commercial tool and perform experiments in foundry technology nodes from $32nm$ to sub-$5nm$. We apply our detailed placement optimization to 19 industry designs. Design information is summarized in Table III. We are not able to provide details of the design12 - design19 benchmarks due to product confidentiality constraints for these sub-$5nm$ designs. All experiments are performed on Intel Xeon servers. Note that the placement solutions of these designs are already well-optimized by a leading commercial tool that considers pin accessibility in a node-specific manner during placement. All results are reported by the commercial tool.

A. Study of Displacement Constraint

To assess the impact of displacement (Disp.) constraint x_Δ, we sweep x_Δ from 0 to 4 (step size 1) for a sub-$5nm$ design. Table IV shows the difference in WNS, initial detailed routing violation count (#init. DRC), final routed wirelength (RWL) and percentage over-congested GCells (Overcon). With $x_\Delta \geq 2$, the timing, RWL and Overcon start to degrade versus $x_\Delta = 1$. This reflects imperfect correlation of DP cost with routing and timing outcomes: max displacement range $x_\Delta = 1$ provides enough solution space for our optimization while honoring the already well-optimized original placement. We thus apply $x_\Delta = 1$ in all experiments below.

TABLE III: Benchmark information (omitting sub-$5nm$ Design12-19 cases).

Benchmark	#stdcell	#macro	#net	Util. (%)	Node
design1	119998	0	131514	78.385	$7nm$
design2	547180	20	566347	39.035	$7nm$
design3	521409	60	533777	36.371	$16nm$
design4	100005	0	111392	54.814	$16nm$
design5	213065	8	233781	38.244	$20nm$
design6	1790828	192	1811059	35.593	$20nm$
design7	815974	90	889974	64.043	$28nm$
design8	159429	0	197441	57.424	$28nm$
design9	337524	45	394336	65.887	$28nm$
design10	91667	0	101903	55.064	$28nm$
design11	43798	0	48171	54.855	$32nm$
design12-19	–				sub-$5nm$

TABLE IV: Experimental results for displacement range constraint.

Disp.	WNS (ns)	#init. DRC	RWL (μm)	Overcon (%)
0	**-0.10**	241373	2952508	3.68
1	**-0.10**	239828	**2950170**	**3.63**
2	-0.12	238523	2954262	3.71
3	-0.11	**237391**	2953569	3.71
4	-0.11	241209	2957398	3.79

B. Main Results

We apply our pin access-driven dynamic programming-based optimization to all design blocks in Table III. We use a WL weighting factor $\alpha = 0.01$ based on our empirical studies. Table V and Table VI present experimental results, comparing the routing solutions based on (i) placement solution optimized by the commercial tool, and (ii) placement solution after further optimization by our methodology. Table V gives the comparison of timing, including worst negative slack (WNS) and total negative slack (TNS), total power, optimization CPU time, and subsequent detailed routing CPU time. Table VI gives the comparison of initial detailed routing violation count (#init. DRC), final detailed routing violation count (#final DRC), routed wirelength, via count (#via) and percentage of over-congested GCell (Overcon) across all metal layers.

Table V confirms that the runtime of our optimization is negligible as compared to the runtime of detailed routing. Notably, the subsequent runtime of detailed routing is reduced by up to 31.82% (avg. 15.06%) with our *in-route* detailed placement optimization. With the negligible optimization runtime, we also reduce WNS by up to 52ps (avg. 10ps), and reduce TNS by up to 759.13ns (avg. 60.01ns). Moreover, we reduce total power by up to 0.15% (avg. 0.03%).

Based on Table VI, we observe that, for #init. DRC, we achieve improvement in 14 out of 19 testcases. We reduce #init. DRC up to 10.13% (avg. 1.28%), with similar #final DRC. For routed wirelength and via count, we achieve improvement in most testcases. We reduce routed wirelength by up to 0.20% (avg. 0.04%), and reduce via count by up to 0.42% (avg. 0.09%). For congestion, we reduce the percentage of over-congested GCell by up to 0.35% (avg. 0.06%). Note that these improvements are achieved over final, well-optimized placement solutions from a leading commercial tool.

TABLE V: Comparison of worst negative slack (WNS), total negative slack (TNS), total power, optimization CPU time (Runtime) and detailed routing CPU time (DR runtime) between commercial tool (Comm.) and our work (Ours). Positive reduction values = improvements.

Benchmark	**Metrics**									
	WNS (ns)		**TNS** (ns)		**Total power** (mW)		**Runtime** (s)		**DR Runtime** (s)	
	Comm.	Ours	Comm.	Ours	Comm.	Ours	Comm.	Ours	Comm.	Ours
design1	**-0.161**	-0.174	-28.62	**-28.48**	20.56	**20.53**	–	13	10456	**7717**
design2	-0.201	**-0.197**	-14.45	**-13.34**	85.00	**84.97**	–	40	27609	**21833**
design3	-0.563	**-0.511**	-904.25	**-856.72**	188.50	**188.40**	–	91	34934	**23817**
design4	-0.122	**-0.087**	**-3.68**	-5.39	18.27	**18.27**	–	13	7590	**5696**
design5	-0.092	**-0.088**	-35.63	**-29.75**	288.40	**288.30**	–	16	8561	**6611**
design6	**-0.478**	-0.487	-1665.20	**-1367.50**	**957.50**	957.60	–	226	88134	**81295**
design7	-0.221	**-0.172**	-1405.20	**-646.07**	439.10	**438.90**	–	103	33822	**25147**
design8	-0.077	**-0.063**	-48.92	**-41.53**	45.39	**45.37**	–	14	8191	**6893**
design9	**-0.083**	-0.090	-65.97	**-65.31**	95.76	**95.74**	–	34	16969	**14436**
design10	-0.133	**-0.119**	-44.76	**-40.34**	28.17	**28.17**	–	5	4120	**3474**
design11	-0.085	**-0.080**	-17.54	**-15.71**	15.09	**15.08**	–	2	2014	**1499**
sub-5nm										
design12	-0.007	**-0.007**	-0.043	**-0.035**	84.2	**84.2**	–	35	1127	**1003**
design13	-0.063	**-0.054**	-18.1	**-16.934**	180.4	**180.4**	–	348	12556	**11417**
design14	-0.1	**-0.1**	-69.151	**-57.525**	198.1	**198.0**	–	245	9898	**9596**
design15	-0.048	**-0.042**	**-50.496**	-50.828	265.0	**265.0**	–	346	8768	**8395**
design16	-0.103	**-0.077**	-55.018	**-51.064**	294.0	**294.0**	–	372	11055	**9825**
design17	**-0.033**	-0.036	-9.567	**-8.192**	87.1	**87.1**	–	200	47758	**47153**
design18	-0.025	**-0.025**	-1.973	**-1.747**	89.5	**89.5**	–	178	17627	**16922**
design19	-0.038	**-0.033**	**-13.684**	-15.549	121.5	**121.5**	–	196	15960	**14332**
Avg. reduction (units)	0.01		60.01		–		–		–	
Avg. reduction (%)	–		–		0.03%		–		15.06%	

TABLE VI: Comparison of initial detailed routing violation count (#init. DRC), final detailed routing violation count (#final DRC), wirelength, via count (#via) and over-congested GCell (Overcon) between commercial tool (Comm.) and our work (Ours). Positive reduction values = improvements.

Benchmark	**Metrics**									
	#init. DRC		**#final DRC**		**Wirelength** (μm)		**#via**		**Overcon** (%)	
	Comm.	Ours	Comm.	Ours	Comm.	Ours	Comm.	Ours	Comm.	Ours
design1	2098	**2087**	7	**4**	1245216	**1242721**	1073763	**1069850**	0.07	**0.06**
design2	**4212**	4398	**1**	4	7059722	**7054360**	4765333	**4762265**	0.39	**0.39**
design3	9265	**9243**	**27**	31	9611743	**9605399**	5377754	**5377121**	0.19	**0.19**
design4	**1019**	1094	4	**1**	1231962	**1231019**	995369	**994380**	0.96	**0.96**
design5	123	**116**	18	**17**	5451472	**5446375**	1736060	**1733824**	0.06	**0.06**
design6	7122	**6838**	**60**	65	**93548447**	93584424	**20093121**	20099553	0.43	**0.41**
design7	10592	**10270**	**63**	71	16514279	**16503297**	6774719	**6747758**	0.01	**0.01**
design8	3714	**3660**	7	**3**	3416199	**3413156**	1546361	**1544693**	0.09	**0.09**
design9	1692	**1658**	32	**27**	9703209	**9697284**	**3063067**	3063454	0.01	**0.01**
design10	185	**171**	48	**42**	**1775344**	1775584	**702774**	703469	0.69	**0.66**
design11	79	**71**	**2**	3	656508	**655495**	341956	**340532**	0.42	**0.39**
sub-5nm										
design12	8721	**8829**	58	**55**	612379	**612304**	**672426**	673215	0.15	**0.15**
design13	204963	**203199**	21	**21**	2799052	**2798876**	**3979621**	3982364	8.78	**8.67**
design14	241373	**239828**	**15**	27	2952508	**2950170**	3934269	**3929686**	3.68	**3.63**
design15	176544	**176019**	12	**12**	**2972888**	2973352	**4099268**	4099386	7.82	**7.61**
design16	**235245**	235487	22	**22**	3181943	**3180857**	4176552	**4175833**	4.45	**4.42**
design17	194647	**192867**	1001	**908**	**894068**	895508	1999585	**1992910**	12.23	**11.88**
design18	**158588**	158639	158	**127**	896906	**896627**	1917938	**1917663**	3.79	**3.66**
design19	150608	**150389**	101	**90**	**943526**	944317	**2021737**	2022820	8.97	**8.76**
Avg. reduction (units)	–		6.68		–		–		0.06	
Avg. reduction (%)	1.28%		–		0.04%		0.09%		–	

V. Conclusions

In this work, we present an *in-route* dynamic programming-based pin access-driven detailed placement optimization methodology to significantly reduce the detailed routing runtime, with noticeable benefits in initial detailed routing DRC count, timing, and routed wirelength. We show that with integration of our *in-route* placement optimization, the detailed routing runtime can be reduced by up to 31.82% (avg. 15.06%) with up to 10.1% (avg. 1.28%) reduction in initial detailed routing DRCs across a wide spectrum of industry designs and technology nodes. Our ongoing research directions include (i) DRC-driven detailed placement refinement; and (ii) a more comprehensive timing-aware optimization flow considering different cell timing criticality characteristics.

References

[1] W.-K. Chow, J. Kuang, X. He, W. Cai and E. F. Y. Young, "Cell Density-Driven Detailed Placement with Displacement Constraint", *Proc. ISPD*, 2014, pp. 3-10.

[2] Y. Ding, C. Chu and W.-K. Mak, "Pin Accessibility-Driven Detailed Placement Refinement", *Proc. ISPD*, 2017, pp. 133-140.

[3] C. Han, A. B. Kahng, L. Wang and B. Xu, "Enhanced Optimal Multi-Row Detailed Placement for Neighbor Diffusion Effect Mitigation in Sub-10nm VLSI", *IEEE Trans. on CAD* 38(9) (2019), pp. 1703-1716.

[4] A. B. Kahng, L. Wang and B. Xu, "The Tao of PAO: Anatomy of a Pin Access Oracle for Detailed Routing", *Proc. DAC*, 2020, to appear.

[5] S. Mantik, G. Posser, W.-K. Chow, Y. Ding and W.-H. Liu, "ISPD2018 Initial Detailed Routing Contest and Benchmarks", *Proc. ISPD*, 2018, pp. 140-143.

[6] J. Seo, J. Jung, S. Kim and Y. Shin, "Pin Accessibility-Driven Cell Layout Redesign and Placement Optimization", *Proc. DAC*, 2017, pp. 54:1-54:6.

[7] T. Taghavi, C. Alpert, A. Huber, Z. Li, G.-J. Nam and S. Ramji, "New Placement Prediction and Mitigation Techniques for Local Routing Congestion", *Proc. ICCAD*, 2010, pp. 621-624.

[8] X. Xu, B. Cline, G. Yeric, B. Yu and D. Z. Pan, "Self-Aligned Double Pattering Aware Pin Access and Standard Cell Layout Co-Optimization", *IEEE Trans. on CAD* 34(5) (2015), pp. 699-712.

[9] X. Xu, B. Yu, J.-R. Gao, C.-L. Hsu and D. Z. Pan, "PARR: Pin Access Planning and Regular Routing for Self-Aligned Double Patterning", *ACM Trans. on DAES* 21(3) (2016), pp. 42:1-42:21.

[10] T.-C. Yu, S.-Y. Fang, H.-S. Chiu, K.-S. Hu, P. H.-Y. Tai, C. C.-F. Shen and H. Sheng, "Pin Accessibility Prediction and Optimization with Deep Learning-based Pin Pattern Recognition", *Proc. DAC*, 2019, pp. 220:1-220:6.

[11] K.-S. Hu and M.-J. Yang, "Problem B: Routing with Cell Movement", *ICCAD-2020 CAD Contest*, http://iccad-contest.org/2020/