Revisiting Inherent Noise Floors for Interconnect Prediction

Tuck-Boon Chan[1], Andrew. B. Kahng[2] and Mingyu Woo[2]

[1]Qualcomm, San Diego, CA, USA
[2]UC San Diego, La Jolla, CA, USA

ABSTRACT

Today's synthesis, placement and routing (SP&R) tools routinely handle millions of instances. Accurate prediction of outcomes is needed to avoid long wasted runtimes from, e.g., unroutable floorplans or placements. However, tool outputs have inherent noise that implies a lower bound on prediction error [10] [7]. The goal of interconnect prediction naturally raises a question of "How accurate can interconnect prediction be?" In this work, we revisit the topic of inherent noise and "chaos" in IC implementation flows, to characterize current *noise floors* on interconnect prediction. We study effects on commercial P&R tool outcomes of such previously-identified noise sources as reordering and renaming in instance cells, nets, and master cells. We also perform studies for macro placement, by slightly shifting the location of macro placement blockages in the center of the layout floorplan. We find that recent commercial tool versions still show significant routed wirelength noise of up to 7% when netlist reordering is applied, and 11.5% when macro placement blockages are shifted. Finally, we also raise the question of "How should predictions be used?" by showing example scenarios where advance knowledge of physical design outcomes can potentially worsen noise and predictability.

ACM Reference Format:
Tuck-Boon Chan[1], Andrew. B. Kahng[2] and Mingyu Woo[2]. 2020. Revisiting Inherent Noise Floors for Interconnect Prediction. In *System-Level Interconnect - Problems and Pathfinding Workshop (SLIP '20), November 5, 2020, San Diego, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3414622.3431907

1 INTRODUCTION

With the slowdown of classical scaling, it is more important than ever for industry design organizations to achieve improved design quality with reduced design schedule. Toward this goal, a key lever is *prediction*: what will be the power, performance and area quality of the design outcome, if the implementation flow is allowed to continue? An accurate predictor can enable more design space exploration earlier in the design process – e.g., at SOC architecture, floorplan or RTL design – since designers can prune solution paths that are hopeless, and free up design resources to pursue more promising paths. In this way, prediction leads to both quality and schedule benefits.

The goal of interconnect prediction naturally raises a question of "How accurate can interconnect prediction be?" [8] and [9] point out that modern tools that internally chain many heuristic combinatorial optimizations exhibit the chaotic behaviors reported in [7], especially when pushed to achieve the best possible solution quality. Earlier work of [10] observed inherent noise in P&R tools: different solutions with a large range of solution quality are produced for isomorphic inputs that vary only in the names of cell instances, nets, and cell masters.

Furthermore, the goal of interconnect prediction (and prediction of layout and physical design attributes such as area, timing or power in general) raises a second question of "How should predictions be used?" Useful, or actionable, predictions in IC design tend to be of a constructive nature. For example, a prediction of "total routed wirelength will be 3km" or "3.3GHz is the maximum achievable clock period within a 1W power budget" is not useful without an associated implementation tool recipe (e.g., a Tcl runscript) that will make the prediction come true. In another scenario, acting upon a prediction in the obvious way may actually be harmful to the final design quality. Below, we show that this is a very real risk of prediction.

In this paper, we make the following contributions. (1) We experimentally study the *noise floor* on interconnect prediction accuracy, using recent releases of leading commercial P&R tools and a commercial sub-20nm foundry technology with commercial cell library and IPs. (2) We reproduce earlier studies of noise [10] and "chaos" [7]. We find that new noise sources (e.g., reordering signal nets in the input Verilog netlist) affect solution quality today. Furthermore, previously identified noise sources (e.g., renaming cell masters, nets and/or instances in the netlist) still affect the solution. (3) We also observe a new form of chaotic behavior in the tool, where the input perturbation is a slight movement of a placement blockage in the middle of the layout region, and the solution shows high instability with respect to this blockage's exact location (see Figure 3). (4) We show examples where naive use of information from prediction can harm solution quality, and propose this as a key consideration for future research on prediction of IC design implementation.

The remainder of this paper is organized as follows. Section 2 reviews several related works, including the key works [10] and [7] whose experiments we revisit in this work. Section 3 describes experiments that revisit the tool noise studies of [10]. Section 4 describes experiments that revisit the "chaos" studies of [7]. Section 5 adds studies of the border between chaotic and stable behaviors in macro placements, as a function of relatively small placement blockages placed near the middle of the layout region. Section 6 shows examples of risk in the use of prediction information, i.e., how a partial prediction can harm solution quality. We conclude the paper in Section 7.

2 RELATED WORKS

Heuristics for difficult optimizations in VLSI CAD often return locally optimum solutions. The distribution of solution qualities seen in local minima for graph bisection and traveling salesperson problems, as well as 'globally-convex' structure of the set of local

SLIP '20, November 5, 2020, San Diego, CA, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8106-2/20/11…$15.00
https://doi.org/10.1145/3414622.3431907

minima, has been studied by, e.g., [1]. For a given problem, the solution quality distribution seen over local minima will change according to the strength of the heuristic [4]. Predictions of solution quality, i.e., the learning of models for optimization outcomes, must therefore deal with the existence of noise [11].

The work of [10] studied noise sources in the context of Cadence QPlace and WRoute tools. The phenomenon of tool noise due to naming and ordering of cells/nets in a netlist was previously noted in works of Hartoog [5], Harlow and Brglez [6], and Bodapati and Najm [2]. [10] gives a taxonomy of tool behavior criteria (monotonicity, smoothness, scaling) as well as a taxonomy of perturbations (randomness, ordering and naming, library richness, constraints, and geometric properties) that do not affect the correctness (i.e., well-formed nature) of solutions. Experiments are performed to assess tool monotonicity, effect of random seeds, netlist ordering, random renaming of cells and nets, and random cell renaming while preserving hierarchy. Further studies of noise *additivity* and the potential for exploiting noise through best-of-k multi-start approaches, are also given.

Jeong et al. [7] also studied noise sources across synthesis, placement and routing (SP&R), and across tools from multiple EDA vendors. This work also addressed "chaos", the non-smoothness of output metrics with respect to small input changes such as picosecond-scale changes to timing constraints (clock period, clock uncertainty and IO delay). Experiments in [7] showed that a 1ps change in timing constraints could cause up to 16.4% variation in the area of the post-synthesis gate netlist. A method was proposed to find the input parameters to which solution quality is most sensitive, and to empirically determine the optimal number of runs k needed to obtain a robust "best-of-k" solution in practice.

Motivated by noise and chaos in SP&R tools, Kahng et al. [9] study multi-armed bandit (MAB) formulation and propose an adaptive sampling strategy under license, schedule, area and frequency constraints. Adaptive sampling in the MAB context embodies the exploration versus exploitation tradeoff inherent in search and optimization – in this case, automatic (no-human-in-the-loop) optimization of the design process to obtain a high-quality final design solution.

3 REVISITING TOOL NOISE

In this section, we revisit the taxonomy of P&R noise sources from [10]. In [10], the term *noise source* connotes a perturbation of the input that is not expected to change the underlying optimization instance, and hence should not change the tool solution. We perform experiments using recent releases of two leading commercial P&R tools, that are selected from among three major tools (Cadence Innovus, Synopsys IC Compiler II, and Mentor Olympus-SoC), but report anonymized results to comply with EULA restrictions. (We use the names "P&R_1" and "P&R_2" consistently to refer to these tools, but do not disclose the mapping of names to tools.) Experiments are performed with a commercial 14nm foundry technology, with multiple-VT 9-track standard cells and generated memories from a leading third-party IP provider. All timing metrics (worst negative slack (WNS), total negative slack (TNS)) are reported for worst-case analysis in the same (unnamed) corner for this enablement. Table 1 lists testcases used in our study.[1]

[1]Sources. (1) AES from [12], (2) JPEG from [13], (3) SweRV_wrapper from [14], and (4) BlackParrot from [15].

Table 1: Testcases used in our experiments.

Design	**ClkPer** (ns)	**IO Delay** (ns)	**WNS** (ns)	**TNS** (ns)	Num Macros	Num Insts
AES	0.300	0.000	-0.235	-56.502	0	14837
JPEG	0.400	0.000	-0.038	-8.939	0	60127
SweRV_wrapper	0.500	0.000	-0.204	-252.882	28	109692
BlackParrot	0.800	0.760	-0.128	-21.676	49	314393

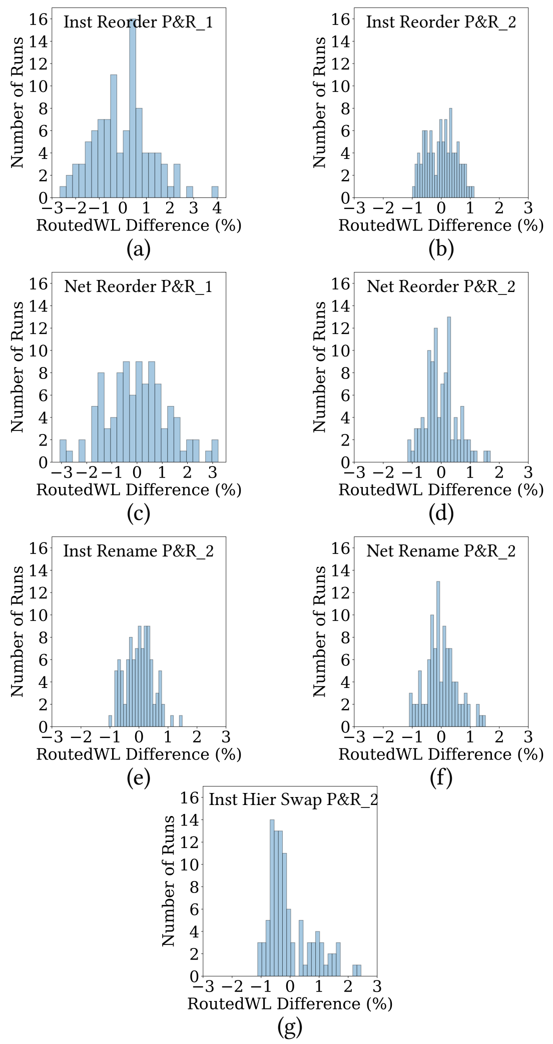

Figure 1: Noise from reordering, renaming, and hierarchical instance swap. Routed wirelength distributions from *AES* (P&R_1 tool) over 100 runs, where 0% corresponds to mean wirelength over all runs. Results are for (a) instance reordering in P&R_1 and (b) inst reordering in P&R_2; (c) net reordering in P&R_1 and (d) net reordering in P&R_2; (e) inst renaming in P&R_2, (f) net renaming in P&R_2, and (g) hierarchical swapping in P&R_2. A 7% spread in routed WL is seen from instance reordering and 2.5% spread in routed WL is seen from instance renaming. Note that P&R_1 does not show any difference from inst renaming, net renaming, or hierarchical swapping.

3.1 Monotonicity and Random Seeds

Sections 3.1 and 5.1 of [10] define and execute a monotonicity test, and demonstrate how a user-determined tool effort level affects solution quality. In recent P&R tools, such fine-grained control over effort level is now unavailable. Similarly, Sections 4.1 and 5.2 of [10] define and execute a test for sensitivity to user-defined random seeds. However, in recent P&R tools, seeding is not available.

3.2 Ordering and Naming

Sections 4.2 and 5.3, 5.4 and 5.5 of [10] define and execute several tests involving ordering, naming, and hierarchy perturbation in the input design data.

3.2.1 Renaming instances, nets and master cells. [10] reported that Cadence P&R tools showed routed wirelength variation of up to 7% when renaming is applied. We follow the experimental procedure described in [10]. We rename the instance names as "INST_XXX", net names as "NET_XXX" and master cell names as "MASTER_XXX", where XXX is a random number between 1 and #INSTs, #NETs, and #MASTERs, respectively. When cell instances or nets are renamed, we change the gate-level Verilog that is input to P&R accordingly. When master cells are renamed, we change Verilog, cell LEFs, and cell Liberty models accordingly.

We find that one vendor's P&R tool P&R_1 is unaffected by instance, net and master renaming. However, the other vendor tool, P&R_2, gives different results when names are changed in the netlist. This noise effect is shown in Figures 1(e) and (f). Up to 2.5% variation in routed wirelength is seen across 100 runs with the *AES* testcase. The authors of [10] also study perturbation of the design hierarchy (Sections 4.2, 5.5 of their paper), that swaps two instance names in the same hierarchical-level modules. We find that P&R_1 is unaffected by such perturbation, as one would expect from being unaffected by renaming. P&R_2 results do change with hierarchy perturbation, as presented in Figure 1(g). The magnitude of total wirelength variation reported in [10] for this type of noise source was 12%, which is larger than what we observe for the modern tool.

3.2.2 Reordering instances and nets. To assess the impact of instance and net ordering, we shuffle `wire` statements for given nets and reorder instance declaration lines inside the input gate-level Verilog. Shuffling of master cell declarations is subsumed by the shuffling of cell instance declarations, since the input gate-level Verilog has an instance declaration on each line. We do not separately study reordering of master cell declarations.

We generate 100 different gate-level Verilogs by shuffling the instance declaration orders using NumPy with different seeds. Similarly, we generate 100 different gate-level Verilogs by shuffling net declarations. We run the P&R tools on each instance- or net-shuffled Verilog. For the P&R_1 tool, the routed wirelength distributions with shuffling for the AES (aes_cipher_top) design are shown in Figure 1. The tool shows up to 7% noise. For P&R_2, the observed wirelength variation is 2.5%. The magnitude of total wirelength variation reported in [10] for this type of noise source was 7%, which is similar to what we observe for the modern tools.[2]

3.3 Another Noise Source: Floorplan Symmetry

We also consider a noise source that is absent from previous studies, namely, whether an optimization such as macro placement or standard-cell place-and-route is affected by *symmetries.* For example, in modern technologies, it is possible to mirror the entire layout about the Y-axis, obtaining an equally manufacturable layout with identical timing and wirelength metrics. We have examined whether the macro placement step in modern tools shows noise in outcomes due to such symmetry.

We study the behavior of the P&R_1 and P&R_2 tools on the SweRV_wrapper design, when all of the macros are fixed in advance. Table 2 shows routed wirelength after all of the macros are fixed in advance as in original, left-right flipped, up-down flipped, and left-right-up-down flipped. P&R_1 and P&R_2 show 0.6% and 3.6% variations in routed wirelength, respectively.

[2]We have separately examined the effect of reordering macro definitions inside each LEF file (using a python parser and "shuffle"), and changing the order in which multiple LEF files are read into the P&R tool. For P&R_1, no change in tool outcome is observed.

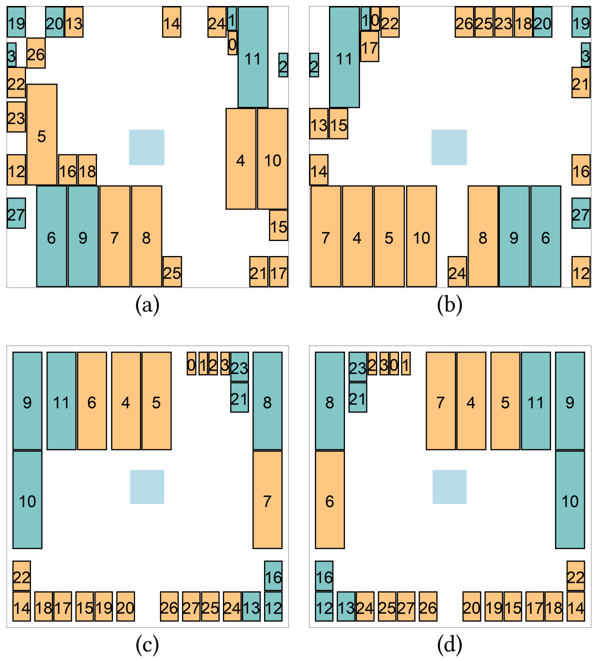

Figure 2: Test of symmetry noise in macro placement, using the SweRV_wrapper design. The center light blue square is a fixed macro placement blockage. Fixed macros are shown in darker blue and movable macros are shown in orange. (a) Some macros in lower-left, upper-left, upper-right regions are fixed, while remaining macros are movable in P&R_1 tool. (b) Same as (a), but mirrored about the Y-axis. (c) Some macros in lower-right, upper-left, upper-right regions are fixed, while remaining macros are movable in P&R_2 tool. (d) Same as (c), but mirrored about the Y-axis.

Table 2: Routed wirelength distribution on SWeRV_wrapper design when all macros are fixed in advance.

Flipped Info	Routed Wirelength (μm)	
	P&R_1	P&R_2
Original	**1720095**	1752283
LR Flipped	1728234	1775584
UD Flipped	**1731809**	**1731447**
LRUD Flipped	1730830	**1794888**

We further study the behavior of P&R_1 using the same design, where three macros at corners of a placed floorplan are selected randomly and fixed (dark blue instances in Figure 2). We then mirror the fixed layout about the Y-axis and determine whether the tool solution will also be mirrored. Figure 2 shows that very different solutions result from the two mirrored pre-placements.[3]

Furthermore, the routed wirelength of (a) is 9.7% greater than that of (b), while the routed wirelength of (c) is 0.1% greater than that of (d). This type of noise (in the P&R_1 tool) suggests simple heuristics, as noted in [10] [7]: e.g., run place-and-route twice (mirrored and non-mirrored), and return the better solution. Mirroring of site maps (N, FS row orientations) and pre-placements about the X-axis to induce noise in outcomes may also be possible.

4 REVISITING TOOL "CHAOS"

In this section, we revisit the concept of tool "chaos" [7], where very small changes to inputs (clock period, I/O delay, and clock uncertainty) are observed to cause large changes to outputs.

In replicating studies of [7], we use the designs and clock settings as in Table 1. For synthesis tools, Table 3 shows post-synthesis WNS and area outcomes of the Syn_1 and Syn_2 tools. Here, the two tools are selected from among three major tools (Synopsys Design Compiler, Cadence Genus, and Mentor Oasys-RTL). To further anonymize, we consistently refer to one of these tools as "Syn_1" and the other as "Syn_2". Both Syn_1 and Syn_2 show small post-synthesis WNS effects from small perturbations, particularly clock period and clock uncertainty. The Syn_2 tool shows near-deterministic results with respect to I/O delay perturbation in most cases.

Our studies of chaotic effects in P&R show that small changes in timing constraints, as well as perturbations of aspect ratio and utilization, can cause changes in post-routed results. Table 4 shows the post-routed outcomes for WNS and TNS (self-reported) of the P&R_1 and P&R_2 tools. One discovery is that P&R_1 has very deterministic results when run multiple times with exactly the same settings, i.e., zero perturbations of clock period perturbation by +0 ps, clock uncertainty by +0 ps, I/O delay by +0 ps, aspect ratio by +0.00, and utilization by +0 % show deterministic results on most of the designs. In contrast, P&R_2 generates quite noisy results when run multiple times with the same settings. Further, we find that the P&R_2 tool exhibits extremely large "chaos" in its outcomes.[4] Arguably, chaotic effects in place-and-route are *larger* than what was seen a decade ago. This suggests a greater challenge for interconnect prediction today than in the past.

5 CHAOTIC TOOL BEHAVIOR IN MACRO PLACEMENT

We further study a type of chaotic behavior involving designs with large numbers of macros (e.g., SRAMs and register files). This type of design is increasingly relevant in modern IC products. Today's commercial place-and-route tools offer automated macro placement capability, typically for use in early design planning steps. Macro placements strongly affect final layout metrics, including

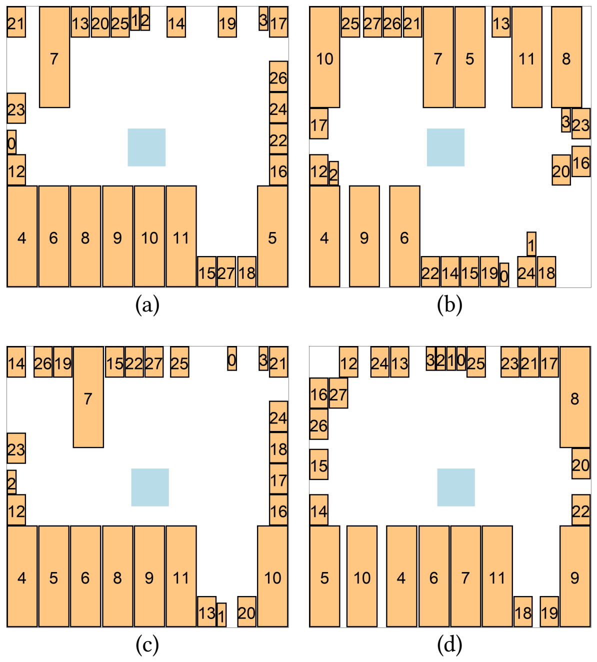

Figure 3: Visualized solutions when a relatively small macro placement blockage (light blue square) is shifted slightly in the *SweRV_wrapper* implementation. The orange rectangles are movable macros. Four example P&R_1 macro placements are shown. The macro placement blockage is 64um × 64um. From its original position, the blockage is (a) located at center (baseline), (b) 6um to the left, (c) 6um to the right, and (d) 12um to the right. These correspond respectively to Rows 4, 10, 12, 13 of Table 5.

timing, wirelength, routability (number of post-route DRC violations), and power. The commercial macro placement follows the conventional strategy of pushing macros to corners and sides of the layout, leaving a relatively unobstructed region for standard-cell place-and-route. Such a strategy can be seen in such academic works as Chen et al. [3], whose MP-tree algorithm leaves empty space at the center of the layout to maintain routability. In practice, a physical designer normally defines the macro placement blockage at the center of the layout during auto-macro placement, in order to preserve the region for place-and-route.

In our study, we first run auto-macro placement repeatedly and confirm that the outcomes are identical in each run. This indicates that the auto-macro placer produces a deterministic output for a fixed input. However, we observe chaotic behavior when the small macro placement blockage in the center region of the block is shifted slightly. Figure 3 shows example outcomes for *SweRV_wrapper* and the P&R_1 tool, in which macro placements are drastically different when a small fixed macro placement blockage (light blue square) is shifted slightly. Details of 13 macro placement outcomes (post P&R) with slight changes in the location of a fixed 64um × 64um macro placement blockage (i.e., original location, and shifts of {left, up, right, down} by {6, 12, 18} um) are given in Table 5. Results in Table 5 show that the different implementations with slight changes in the macro placement blockage location have up to -3%/+11.5% difference in wirelength compared to the baseline implementation. This is again a challenge for interconnect prediction.

[3]Note that in (a) and (c), the dark-blue fixed macros are in either N, FN, S, FS orientation; in (b) and (d), they are flipped individually (i.e., N <-> FN, S <-> FS). All orange (unfixed) macros are allowed to be placed freely by the tool, with any orientation among (N, FN, S, FS).

[4]The worst TNS of -101.280ns for JPEG is not a typo. Indeed, most of the JPEG runs of P&R_2 return very reasonable results. A possible explanation of the outlier is that today's tools are known to "give up" mid-run if timing or routability looks incurable. This may have happened with the outlier run.

Table 3: Revisiting experiments of [7]. Chaotic behavior is studied in synthesis tools.

Parameter	Noise (Δ)	*AES*				*JPEG*			
		Syn_1		Syn_2		Syn_1		Syn_2	
		WNS (ns)	Area (μm^2)	WNS (ns)	Area (μm^2)	WNS (ns)	Area (μm^2)	WNS (ns)	Area (μm^2)
	-3 ps	**-0.137**	3349.422	**-0.065**	3873.663	-0.113	**15734.759**	**-0.088**	13780.086
	-2 ps	-0.133	3492.679	-0.061	3924.829	-0.113	15420.182	-0.084	13921.811
	-1 ps	-0.135	3446.472	-0.061	3962.004	-0.114	15621.580	-0.081	13814.640
Clock Period	+0 ps	**-0.130**	3490.099	-0.063	3863.825	**-0.117**	15433.850	-0.084	13896.732
	+1 ps	-0.131	3454.899	-0.055	4052.402	-0.109	15537.876	-0.078	13660.376
	+2 ps	-0.131	3393.452	-0.055	3985.350	-0.109	15523.643	**-0.074**	13769.401
	+3 ps	**-0.130**	3438.852	**-0.049**	4061.877	**-0.105**	15684.117	-0.081	13749.080
	-3 ps	**-0.130**	3438.852	-0.055	4061.877	**-0.105**	15684.117	-0.081	13747.749
	-2 ps	-0.131	3393.452	-0.055	3985.350	-0.109	15523.643	**-0.074**	13773.756
	-1 ps	-0.131	3454.899	**-0.049**	4052.402	-0.109	15537.876	-0.078	13662.795
Clock Uncertainty	+0 ps	**-0.130**	3490.099	-0.063	3863.825	**-0.117**	15433.850	-0.084	13896.732
	+1 ps	-0.135	3446.472	-0.059	3995.470	-0.114	15621.580	-0.081	13814.640
	+2 ps	-0.133	3492.679	-0.061	3924.829	-0.113	15420.182	-0.084	13921.206
	+3 ps	**-0.137**	3349.422	**-0.065**	3873.663	-0.113	15734.759	**-0.087**	13784.279
	-3 ps	-0.132	3465.423	**-0.063**	3863.825	**-0.124**	15436.834	**-0.084**	13896.732
	-2 ps	-0.131	3455.746	-0.063	3863.825	**-0.107**	15854.065	-0.084	13896.732
	-1 ps	**-0.130**	3489.454	-0.063	3863.825	-0.117	15376.636	-0.084	13896.732
IO Delay	+0 ps	**-0.130**	3490.099	-0.063	3863.825	-0.117	15433.850	-0.084	13896.732
	+1 ps	-0.132	3432.965	-0.063	3863.825	-0.114	15583.720	-0.084	13896.732
	+2 ps	-0.132	3453.690	-0.063	3863.825	-0.123	15419.134	-0.084	13896.691
	+3 ps	**-0.133**	3422.200	-0.063	3867.333	-0.121	15357.484	-0.084	13895.965
Best	-	-0.130	-	-0.049	-	-0.105	-	-0.074	-
Worst	-	-0.137	-	-0.065	-	-0.124	-	-0.088	-
Delta	-	0.007	-	0.016	-	0.019	-	0.014	-

Parameter	Noise (Δ)	*SweRV_wrapper*				*BlackParrot*			
		Syn_1		Syn_2		Syn_1		Syn_2	
		WNS (ns)	Area (μm^2)	WNS (ns)	Area (μm^2)	WNS (ns)	Area (μm^2)	WNS (ns)	Area (μm^2)
	-3 ps	-0.198	137864.035	-0.167	135813.803	-0.388	315843.656	-0.318	294107.103
	-2 ps	**-0.210**	137659.250	**-0.151**	136047.579	**-0.406**	315855.067	**-0.328**	294984.224
	-1 ps	-0.202	137777.629	-0.154	135833.318	-0.398	315836.076	-0.323	295421.898
Clock Period	+0 ps	-0.206	137688.079	**-0.174**	135649.741	-0.392	315798.498	-0.315	295225.983
	+1 ps	-0.198	137806.176	-0.173	135648.088	-0.399	315886.879	-0.321	295110.063
	+2 ps	-0.188	137973.464	-0.156	135976.333	-0.393	315773.338	-0.317	295342.467
	+3 ps	**-0.183**	137905.283	-0.167	135238.598	**-0.386**	316127.912	**-0.297**	294933.905
	-3 ps	**-0.184**	137866.495	**-0.154**	135619.945	-0.388	315751.041	**-0.297**	294933.905
	-2 ps	-0.188	137942.216	-0.160	135530.918	**-0.383**	315918.490	-0.317	295343.153
	-1 ps	-0.196	137797.144	-0.155	135902.628	-0.385	315788.902	-0.321	295110.063
Clock Uncertainty	+0 ps	-0.206	137688.079	**-0.174**	135649.741	-0.392	315798.498	-0.315	295225.983
	+1 ps	-0.198	137796.459	-0.153	135877.186	-0.395	315876.880	-0.323	295417.825
	+2 ps	-0.214	137618.890	-0.158	135612.123	**-0.402**	315839.745	**-0.328**	294983.901
	+3 ps	**-0.219**	137535.951	-0.159	135895.653	-0.394	315804.788	-0.318	294106.780
	-3 ps	**-0.186**	137925.362	**-0.174**	135649.741	-0.390	315790.877	**-0.315**	295225.983
	-2 ps	-0.191	137879.196	-0.174	135649.741	**-0.390**	315790.877	-0.315	295225.983
	-1 ps	-0.205	137652.799	-0.174	135649.741	-0.390	315790.152	-0.315	295225.983
IO Delay	+0 ps	-0.206	137688.079	-0.174	135649.741	-0.392	315798.498	-0.315	295225.983
	+1 ps	-0.199	137745.898	**-0.164**	135714.011	-0.393	315825.553	-0.315	295225.983
	+2 ps	-0.196	137862.422	-0.169	135590.592	**-0.394**	315850.793	-0.315	295225.983
	+3 ps	**-0.211**	137680.136	-0.174	135584.463	-0.393	315841.600	-0.315	295225.983
Best	-	-0.183	-	-0.151	-	-0.383	-	-0.297	-
Worst	-	-0.219	-	-0.174	-	-0.406	-	-0.328	-
Delta	-	0.036	-	0.023	-	0.023	-	0.031	-

6 BEYOND NOISE AND CHAOS: CAN PREDICTIONS BE HARMFUL?

We have also studied a second key question, namely, "How should predictions be used?" Recent years have seen tremendous energy devoted to machine learning for modeling and prediction of physical design. Yet, there may be contexts where use of predictions can cause, e.g., additional noise or chaos in tool outcomes. The message here, perhaps, is "Be careful what you ask for."

In this section, we show an example scenario where advance knowledge of the physical design outcome worsens noise and predictability. We select a subset of macros, pre-place them to locations in original macro placement output and let the P&R tool place remaining macros. This experiment mimics the use case where a partial solution (i.e., locations of some macros) is determined through prediction. Figure 4 shows example outcomes for *SweRV_wrapper* using P&R_1 and P&R_2 tools. Results show that macro placement outcomes vary when different subsets of macros are pre-placed. For example, post-P&R implementation wirelength of macro placement in Figure 4(b) is 1,772,586 μm or 3.5% longer compared to baseline macro placement in Figure 4(a). This experiment highlights that even though some locations of macros are known and pre-placed, there is still some uncertainty in final P&R wirelength. This chaotic behavior in P&R tools again puts a limit on achievable accuracy of interconnect predictions.

Epilogue. We add a final "epilogue" regarding the earlier comment, "be careful what you ask for" (in terms of predictions). A well-known challenge for prediction is to bridge the gap between the post-synthesis netlist and the post-P&R netlist which has undergone sizing, buffering, hold fix, and many other physical synthesis and optimization transforms. Physical designers and methodologists universally agree that bridging this gap will improve timing and routability convergence. We ask the question, "Is it helpful to know *exactly* what the final post-P&R netlist will be?" Then, a trivial experiment takes a final post-P&R(&Opt) netlist and feeds it back

Table 4: Revisiting experiments of [7]. Chaotic behavior is studied in P&R tools.

Parameter	Noise (Δ)	*AES*				*JPEG*			
		P&R_1		P&R_2		P&R_1		P&R_2	
		WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)
Clock Period	-3 ps	-0.232	-58.262	-0.230	-29.233	-0.064	-9.332	-0.026	-10.991
	-2 ps	-0.240	-58.440	-0.227	-29.745	-0.040	-12.319	-0.024	-8.659
	-1 ps	-0.229	-56.220	-0.223	-29.221	-0.051	-10.270	-0.034	-15.317
	+0 ps	-0.235	-56.502	-0.244	-30.112	-0.038	-8.939	-0.052	-10.733
	+1 ps	-0.236	-56.306	-0.226	-30.314	-0.039	-8.955	-0.030	-9.221
	+2 ps	-0.229	-56.717	-0.220	-29.523	-0.045	-13.505	-0.024	-8.248
	+3 ps	-0.231	-55.345	-0.230	-29.302	-0.034	-8.859	-0.026	-10.252
Clock Uncertainty	-3 ps	-0.234	-56.518	-0.224	-28.752	-0.033	-9.448	-0.031	-11.650
	-2 ps	-0.233	-56.770	-0.224	-29.634	-0.038	-14.466	-0.024	-8.400
	-1 ps	-0.240	-56.046	-0.232	-29.877	-0.040	-9.995	-0.033	-8.519
	+0 ps	-0.235	-56.502	-0.245	-30.663	-0.038	-8.939	-0.054	-11.020
	+1 ps	-0.235	-56.120	-0.224	-29.988	-0.041	-10.839	-0.031	-8.128
	+2 ps	-0.232	-58.406	-0.232	-29.638	-0.046	-11.917	-0.026	-7.406
	+3 ps	-0.238	-57.927	-0.234	-29.011	-0.042	-8.540	-0.031	-11.416
IO Delay	-3 ps	-0.236	-56.588	-0.230	-29.396	-0.041	-10.002	-0.033	-9.132
	-2 ps	-0.237	-56.136	-0.235	-32.322	-0.038	-10.015	-0.039	-9.614
	-1 ps	-0.235	-57.233	-0.220	-29.723	-0.042	-9.862	-0.031	-8.500
	+0 ps	-0.235	-56.502	-0.248	-30.461	-0.038	-8.939	-0.061	-11.636
	+1 ps	-0.228	-56.762	-0.230	-29.686	-0.039	-11.289	-0.032	-9.144
	+2 ps	-0.241	-56.067	-0.221	-28.934	-0.043	-9.516	-0.046	-10.431
	+3 ps	-0.232	-56.770	-0.223	-28.393	-0.044	-10.283	-0.036	-7.903
Aspect Ratio	-0.03	-0.243	-56.916	-0.229	-29.404	-0.039	-10.310	-0.027	-10.028
	-0.02	-0.229	-58.337	-0.222	-30.464	-0.029	-5.623	-0.028	-9.841
	-0.01	-0.235	-56.502	-0.225	-29.705	-0.039	-12.495	-0.070	-101.280
	+0.00	-0.235	-56.502	-0.234	-30.820	-0.038	-8.939	-0.054	-9.934
	+0.01	-0.228	-58.638	-0.226	-31.309	-0.038	-9.543	-0.038	-11.587
	+0.02	-0.233	-58.890	-0.228	-29.209	-0.038	-10.984	-0.035	-11.803
	+0.03	-0.250	-58.067	-0.240	-30.619	-0.041	-8.356	-0.031	-10.055
Placement Util	-3 %	-0.235	-57.330	-0.237	-29.374	-0.035	-10.812	-0.031	-9.896
	-2 %	-0.230	-57.392	-0.224	-28.910	-0.037	-7.755	-0.022	-8.901
	-1 %	-0.227	-56.387	-0.224	-29.526	-0.037	-8.477	-0.023	-7.767
	+0 %	-0.235	-56.502	-0.243	-30.571	-0.038	-8.939	-0.036	-9.819
	+1 %	-0.237	-57.808	-0.227	-30.787	-0.044	-11.394	-0.058	-10.013
	+2 %	-0.239	-59.136	-0.231	-30.060	-0.042	-12.405	-0.034	-10.882
	+3 %	-0.233	-56.937	-0.231	-30.093	-0.039	-9.086	-0.027	-12.468
Best	-	-0.227	-55.345	-0.220	-28.393	-0.029	-5.623	-0.022	-7.406
Worst	-	-0.250	-59.136	-0.248	-32.322	-0.064	-14.466	-0.070	-101.280
Delta	-	0.023	3.791	0.028	3.929	0.035	8.843	0.048	93.874

Parameter	Noise (Δ)	*SweRV_wrapper*				*BlackParrot*			
		P&R_1		P&R_2		P&R_1		P&R_2	
		WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)
Clock Period	-3 ps	-0.257	-339.301	-0.502	-278.424	-0.123	-15.197	-0.194	-60.090
	-2 ps	-0.225	-291.141	-0.793	-241.234	-0.126	-49.392	-0.107	-41.889
	-1 ps	-0.265	-336.346	-0.682	-280.573	-0.139	-18.283	-0.122	-64.540
	+0 ps	-0.204	-244.887	-0.566	-163.356	-0.128	-21.676	-0.121	-32.079
	+1 ps	-0.212	-287.519	-0.608	-205.182	-0.111	-10.723	-0.108	-45.347
	+2 ps	-0.224	-318.609	-0.710	-228.354	-0.159	-50.867	-0.167	-78.630
	+3 ps	-0.216	-245.109	-0.528	-232.646	-0.115	-7.891	-0.115	-32.865
Clock Uncertainty	-3 ps	-0.221	-347.686	-0.679	-211.716	-0.135	-118.041	-0.111	-31.143
	-2 ps	-0.210	-265.114	-0.619	-169.827	-0.104	-15.631	-0.117	-42.325
	-1 ps	-0.236	-285.039	-0.602	-259.697	-0.145	-69.956	-0.117	-48.378
	+0 ps	-0.204	-244.887	-0.580	-180.897	-0.128	-21.676	-0.119	-29.889
	+1 ps	-0.207	-332.116	-0.561	-290.598	-0.139	-24.162	-0.143	-32.522
	+2 ps	-0.254	-315.494	-0.620	-282.778	-0.123	-23.965	-0.154	-30.463
	+3 ps	-0.233	-328.919	-0.871	-252.655	-0.141	-11.075	-0.112	-39.416
IO Delay	-3 ps	-0.225	-279.308	-0.578	-224.977	-0.147	-13.286	-0.131	-50.011
	-2 ps	-0.242	-313.851	-0.451	-225.109	-0.147	-58.600	-0.111	-84.830
	-1 ps	-0.250	-332.225	-0.724	-215.154	-0.152	-44.178	-0.094	-28.919
	0 ps	-0.204	-244.887	-0.566	-170.578	-0.128	-21.278	-0.112	-61.921
	1 ps	-0.240	-291.155	-0.697	-179.544	-0.144	-50.005	-0.114	-36.972
	2 ps	-0.206	-289.353	-0.538	-240.698	-0.140	-31.646	-0.119	-44.766
	3 ps	-0.217	-292.142	-0.667	-331.423	-0.123	-26.904	-0.112	-53.538
Aspect Ratio	-0.03	-0.462	-311.995	-0.572	-218.755	-0.114	-47.975	-0.143	-77.539
	-0.02	-0.268	-338.009	-0.708	-202.611	-0.115	-37.853	-0.121	-40.504
	-0.01	-0.205	-229.347	-0.588	-230.526	-0.144	-30.131	-0.129	-35.700
	+0.00	-0.204	-244.887	-0.589	-169.131	-0.128	-20.879	-0.122	-68.511
	+0.01	-0.223	-273.615	-0.573	-263.628	-0.153	-52.868	-0.107	-27.486
	+0.02	-0.214	-244.578	-0.622	-161.388	-0.141	-22.935	-0.098	-36.148
	+0.03	-0.272	-231.320	-0.623	-255.692	-0.161	-34.448	-0.110	-42.620
Placement Util	-3 %	-0.200	-260.962	-0.625	-284.832	-0.186	-62.789	-0.110	-28.408
	-2 %	-0.197	-250.333	-0.634	-217.232	-0.150	-79.364	-0.112	-34.864
	-1 %	-0.228	-232.827	-0.637	-219.109	-0.119	-30.741	-0.137	-50.131
	+0 %	-0.204	-244.887	-0.575	-172.473	-0.128	-21.278	-0.131	-41.080
	+1 %	-0.208	-283.216	-0.672	-216.844	-0.221	-119.503	-0.141	-52.975
	+2 %	-0.246	-294.268	-0.691	-187.814	-0.241	-302.801	-0.131	-32.278
	+3 %	-0.282	-428.977	-0.709	-196.054	-0.235	-122.664	-0.112	-36.651
Best	-	-0.197	-229.347	-0.451	-161.388	-0.104	-7.891	-0.094	-27.486
Worst	-	-0.462	-428.977	-0.871	-331.423	-0.241	-302.801	-0.194	-84.830
Delta	-	0.265	199.63	0.420	170.035	0.137	294.91	0.100	57.344

Table 5: Overall post-routed result comparison with a shifted macro placement blockages on *SweRV_wrapper* design. P&R_1 is used. The bold font denotes min and max values.

Noise (Δ)		WNS (ns)	TNS (ns)	Wirelength (um)	DRC
shiftX (um)	shiftY (um)				
0	-18	-0.078	-12.579	1852592.927	268
	-12	-0.064	-6.356	1767687.207	322
	-6	-0.061	-9.672	1859084.593	155
	+0	-0.085	-11.143	1712514.599	115
	+6	**-0.050**	**-2.564**	**1660819.498**	322
	+12	-0.088	**-18.800**	**1909736.061**	616
	+18	-0.068	-8.538	1769723.536	280
-18	0	-0.087	-12.036	1711466.776	268
-12		**-0.050**	**-2.564**	**1660819.498**	322
-6		-0.085	-11.143	1712514.599	115
+0		-0.085	-11.143	1712514.599	115
+6		**-0.104**	-12.477	1839574.923	418
+12		-0.098	-14.756	1874447.234	> 1000
+18		-0.085	-9.786	1777746.608	103

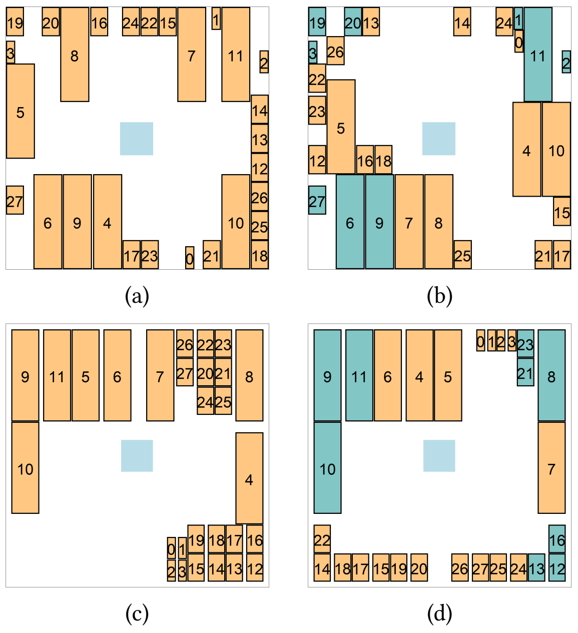

Figure 4: Visualized macro placement solutions when a subset of macros are pre-placed on *SweRV_wrapper* design. The center light blue square denotes macro placement blockages, and darker blue denotes fixed macros from original solutions. (a) original solution from P&R_1. (b) lower-left, upper-left, upper-right macros are fixed from (a). (c) original solution from P&R_2. (d) lower-right, upper-left, upper-right macros are fixed from (c).

to the P&R(&Opt) flow. The original P&R(&Opt) result serves as a constructive proof of the achievable QOR with this netlist. However, as shown in Table 6, the second run can have anywhere from 4% to 10% more routed wirelength than the first run.

7 CONCLUSION

In this work, we have revisited the studies of tool noise and chaos by [10] [7], toward determining "noise floors" for interconnect prediction accuracy. We find that additional sources of tool noise now exist beyond what had been identified in the works of a decade ago, and that today's major vendor tools show qualitatively different susceptibilities to noise sources. We further identify a new source of tool noise: symmetry in the floorplan definition. And, we observe very large chaotic effects on auto-macro placement from the location of a small fixed obstacle. Finally, we ask the question, "How should predictions be used?" and show example scenarios where advance knowledge of physical design outcomes can potentially worsen noise and predictability. We show a potentially harmful effect of knowing part of a macro placement solution in advance. And, we show a somewhat trivial example of harm from knowing the post-route netlist in advance of P&R. Each of these examples reinforces a new caveat for prediction: "Be careful what you ask for."

Table 6: P&R result comparison when post-route final netlist (.v) is fed back into the P&R(&Opt) flow, for the *AES* testcase. The target clock period is set as 450ps. In both tools, the wirelength is increased significantly in the second run, even though the result of the first run is a constructive proof of achievable QOR.

Tools	1st WNS(ns)	1st WL(um)	2nd WNS(ns)	2nd WL(um)
P&R_1	-0.061	101667.24	-0.050	112284.63
P&R_2	-0.064	110604.19	-0.075	114939.83

ACKNOWLEDGMENTS

Research at UCSD is supported by DARPA (HR0011-18-2-0032), the U.S. National Science Foundation (CCF-1564302), Samsung, Qualcomm, NXP Semiconductors, Mentor Graphics, and the C-DEN Center.

REFERENCES

[1] K. D. Boese, A. B. Kahng and S. Muddu, "A New Adaptive Multistart Technique for Combinatorial Global Optimizations", *Operations Research Letters* 16(2) (1994), pp. 101-113.
[2] S. Bodapati and F. N. Najm, "Pre-Layout Estimation of Individual Wire Lengths", *Proc. ACM Intl. Workshop on System-Level Interconnect Prediction*, 2000, pp. 93-98.
[3] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang and T.-Y. Liu, "MP-trees: A Packing-Based Macro Placement Algorithm for Modern Mixed-Size Designs", *IEEE Trans. on CAD* 27(9), 2008, pp. 1621-1634.
[4] L. Hagen and A. B. Kahng, "Combining Problem Reduction and Adaptive Multi-Start: A New Technique for Superior Iterative Partitioning", *IEEE Trans. on CAD* 16(7) (1997), pp. 709-717.
[5] M. R. Hartoog, "Analysis of Placement Procedures for VLSI Standard Cell Layout", *Proc. DAC*, 1986, pp. 314-319.
[6] J. E. Harlow and F. Brglez, "Design of Experiments in BDD Variable Ordering: Lessons Learned", *Proc. ICCAD*, 1998, pp. 646-652.
[7] K. Jeong and A. B. Kahng, "Methodology From Chaos in IC Implementation", *Proc. ISQED*, 2010, pp. 885-892.
[8] A. B. Kahng, "New Directions for Learning-Based IC Design Tools and Methodologies", *Proc. ASP-DAC*, 2018, pp. 405-410.
[9] A. B. Kahng, S. Kumar and T. Shah, "A No-Human-in-the-Loop Methodology Toward Optimal Utilization of EDA Tools and Flows", *DAC work in progress poster*, 2018. https://vlsicad.ucsd.edu/MAB/.
[10] A. B. Kahng and S. Mantik, "Measurement of Inherent Noise in EDA Tools", *Proc. ISQED*, 2002, pp. 206-211.
[11] M. Kearns, "Efficient Noise-tolerant Learning from Statistical Queries", *Proc. ACM STOC*, 1993, pp. 392-401.
[12] Rijndael IP Core, https://opencores.org/projects/aes_core, 2008.
[13] Video compression systems, https://opencores.org/projects/video_systems, 2008.
[14] SweRV RISC-V Core^{TM} 1.1 from Western Digital. https://github.com/westerndigitalcorporation/swerv_eh1, 2019.
[15] D. Petrisko, F. Gilani, M. Wyse, T. Jung, S. Davidson, P. Gao, C. Zhao, Z. Azad, S. Canakci, B. Veluri, T. Guarino, A. Joshi, M. Oskin and M. B. Taylor, "BlackParrot: An Agile Open Source RISC-V Multicore for Accelerator SoCs", *IEEE Micro*, July-August 2020, pp. 93-102.