Measurement of Inherent Noise in EDA Tools *

Andrew B. Kahng † and Stefanus Mantik

† UCSD CSE and ECE Departments, La Jolla, CA 92093-0114
UCLA Computer Science Department, Los Angeles, CA 90095-1596
abk@ucsd.edu, stefanus@cs.ucla.edu

Abstract

With advancing semiconductor technology and exponentially growing design complexities, predictability of design tools becomes an important part of a stable top-down design process. Prediction of individual tool solution quality enables designers to use tools to achieve best solutions within prescribed resources, thus reducing design cycle time. However, as EDA tools become more complex, they become less predictable. One factor in the loss of predictability is *inherent noise* in both algorithms and how the algorithms are invoked. In this work, we seek to identify sources of noise in EDA tools, and analyze the effects of these noise sources on design quality.

Our specific contributions are: (i) we propose new behavior criteria for tools with respect to the existence and management of noise; (ii) we compile and categorize possible perturbations in the tool use model or tool architecture that can be sources of noise; and (iii) we assess the behavior of industry place and route tools with respect to these criteria and noise sources. While the behavior criteria give some guidelines for and characterize the stability of tools, we are not recommending that tools be immune from input perturbations. Rather, the categorization of noise allows us to better understand how tools will or should behave; this may eventually enable improved tool predictors that consider inherent tool noise.

1 Introduction

To accommodate ever-increasing design complexity with shorter design cycle times, several avenues of research have been pursued, including elimination of design iterations, use of early estimations, methodologies such as design reuse, and design process instrumentation and optimization. One potential improvement that has become more important concerns *predictability* of the individual algorithm or design tool [6]. A prediction of the final solution quality from an EDA tool – before running that tool – can significantly reduce design time and enable a top-down predictive design methodology. Ideally, the prediction will prevent any run that leads to bad solutions, and will guide the user to choose a small set of runs (using the right tools and the right tool parameterizations) that will most likely return a high-quality solution.

Creating a prediction model for a tool requires understanding of the behavior of that tool. A typical EDA tool contains several algorithms that are used to perform design optimizations. Most design problems are NP-hard, and there are no exact methods for practical design instances. Therefore, heuristic approaches are used which return suboptimal solutions. These heuristics lead to *noise* that creates variability in solution quality.[1] Noise in EDA tools has been mostly ignored up to now, likely because it is assumed to have little effect on solution quality. However, as tools become more complex, the number of potential noise sources and the potential for noise in solution quality grows accordingly. If tools become unpredictable because of noise effects (i.e., noise creates variations in solution quality or solution structure which deviate unacceptably from predictions), a loss of design quality and productivity will result.

Besides adding variation to solution quality, tool noise can also change the effective solution space or "landscape". In other words, tool noise will make solution spaces less smooth, and hamper gradient-following or other methods for finding good operating points for tools. As a result, design methodologies can require longer runtimes to find acceptable solutions. On the other hand, variations in solution quality due to noise may have some positive effects. For example, such variations permit exploitation of multi-start and order statistics to get predictably good results after some number of design attempts. Noise also delivers variant solutions which can each be explored.

Our goal is to understand how tools behave in the presence of noise, so that we can create better predictive models that are noise-aware. In the remainder of this paper, we (i) propose behavior criteria for tools with respect to inherent noise and predictability, (ii) propose a taxonomy of possible use model and instance perturbations that correspond to noise sources, and (iii) experimentally characterize industry place and route tools according to the proposed behavior criteria. Specifically, Section 2 reviews related works on noise analyses in optimization algorithms. In Section 3 and in Section 4, we respectively present behavior criteria for tools and a taxonomy of possible perturbations (noise sources). Experiments are given in Section 5 with some additional analyses in Section 6, and we discuss ongoing and future research directions in Section 7.

2 Previous works

The presence of noise in heuristic algorithms has been realized by many researchers. For example, in [8], Yakowitz and Lugosi studied a formulation of random search in the presence of noise for the machine learning domain. The effect of noise has also been studied in such domains as stochastic approximation [7]. In the VLSI CAD domain, some early discoveries about noise in placement tools are presented in [5]. In this study, Hartoog showed the existence of solution variants obtained by reordering the cells and/or nets in the netlist. He used this source of variation to create a class of isomorphic circuits. These sets of isomorphic circuits were then used to compare two algorithms more effectively (compared to the traditional method that uses benchmarks with distinct circuits).

A similar analysis of variation due to ordering is also given by Harlow and Brglez [4]. They presented a class of isomorphic circuits created by randomizing the names and the node orders for the BDD representation. The use of noise from cell/net orderings and naming for creating *isomorphism classes* was extended by Ghosh [3], who explored the effect of random ordering on various VLSI design tasks including standard-cell placement and routing, FPGA placement and routing, partitioning, wire crossing minim
logic optimization, BDD variable ordering, and technolog

* This work was supported by a grant from Cadence Design Systems, Inc. and by the MARCO Gigascale Silicon Research Center.

[1] Given the existence of noise in algorithm implementations, designers often run their tools several times and choose the best of the resulting solutions. For each run, design attributes or assumed noise sources (e.g., initial seeds) are perturbed slightly to create variation.

Proceedings of the International Symposium on Quality Electronic Design (ISQED'02)
0-7695-1561-4/02 $17.00 © 2002 IEEE

ping. These variations are used to create isomorphism classes. In addition to the random ordering of cells/nets and the random namings, Ghosh also proposed circuit mutation as another method for generating *equivalent* circuits that belong to the same class. The circuit classes are determined by the "similarity" of such circuit attributes as number of crossings, fanout and fanin distributions, etc.

While the above-mentioned studies used the concept of noise to generate isomorphic circuits for tool benchmarking, Bodapati and Najm [1] analyzed noise in tools from a different perspective. Starting from the premise that noise due to cell/net ordering and naming has a negative effect on estimators, the authors of [1] proposed a pre-layout estimation model for individual wire length, and noted that the accuracy of their estimations are worsened by inherent tool noise (with respect to ordering and naming). The inaccuracy due to noise was quantified by comparing estimated values with actual values.

In summary, although several noise sources in EDA tools have been identified and have been used for specific purposes, the overall range of noise effects on design quality has not been well understood. In the following sections, we attempt to explore the concept of noise sources and effects more holistically.

3 Behavior Criteria for Tools

Most the EDA tools that use heuristic searches will have inherent noise. In this section, we propose several behavior criteria which EDA tools should, ideally, follow. These criteria essentially measure the stability of tools with respect to noise sources; these may be viewed as one set of potential quality metrics for EDA tools. The motivation for our behavior criteria is orthogonal to the analysis of noise: tool sensitivities should be understood and characterized, and behaviors *should be "sensible" even if noisy*.

3.1 Monotonicity

Tools should give *monotone* solutions with respect to input or operational parameters. In other words, expected solution qualities based on such parameters should be monotone. For example, if a tool has an option that controls the "granularity of the search step"[2], the solution quality should increase with decreasing search step size. Thus, the designer can expect a better but more time-consuming solution if he makes the search steps finer.

3.2 Smoothness

Tools should produce similar results for "very small" perturbations of their inputs. For example, if a buffer is added into the netlist, the resulting placement should be similar to the previous placement with the exception of the new buffer and its immediate neighborhood. This is one of the behaviors that is important for tools that have incremental optimization (ECO mode) capability. Some perturbations are more significant than others, e.g., changing a timing constraint by 1 ns may result in greater change to the solution than inserting a single buffer into the netlist.

3.3 Scaling

When the input design is scaled up/down with the same characteristics, the tool should be able to maintain the solution quality with the expected scale. For example, if a design is "doubled" with all design characteristics remaining the same (e.g., Rent parameter, fanout/fanin distributions, etc.), the expected solution should be about "twice" the original solution. Similarly, if we scale down the feature size on the same design without perturbing the netlist, the new expected solution should be similar to the original solution (with all quality measures scaled accordingly).

[2]In this example, we assume that the global optimization performed with a large step size will yield a fair solution with short runtime, while optimization with a small step size will give a better solution with longer runtime.

4 Taxonomy of Possible Perturbations

In this section, we propose a taxonomy of possible perturbations that may contribute to noise in EDA tool outcomes without compromising the legality of solutions.

4.1 Randomness

The first and most obvious type of perturbation relates to the use of randomization in the algorithm or tool. Typically, the random number generator (RNG) depends on an initial seed which determines the ensuing list of random numbers generated during the tool operation. Hence, we can generate noise by varying this seed value. In addition to the seed value, the implementation of the RNG can also be changed. This is done either by changing the library that implements the RNG on one machine or by running the same tool on different machine/operating system.[3]

4.2 Ordering and Naming

Some algorithms are order dependent, i.e., a different solution can be obtained if we change the order of the data. For example, a KL-FM netlist partitioning implementation [2] will search for the cell to be moved to a different partition based on the order of the cells in the gain bucket data structure. If there are several cells that have the same movement cost, the first cell in the "bucket" will most likely be chosen as the one that is to be moved. Therefore, any change in the ordering of the cells will eventually lead to possibly widely different solution qualities. Note that in the KL-FM example, the gain bucket data structure is often initially populated based on the order in which cells/nets appear in the input file.

Although some tools use the orderings that are defined in the input sources (data files), the majority of industry tools that we are familiar with apply some data reordering when the data are read into memory. Typically, the data are sorted according to some criteria, and this reordering lends some immunity to noise triggered by different ordering of the inputs. One of the keys that is typically used in the sorting phase is the name of the instances (e.g., cell names, net names, etc.). Thus, changing the names for the instances may produce different solution.

Finally, some tools exploit extra information that is encoded in the naming. For instance, designers usually encode functional hierarchy in the cell names; thus, it is possible to infer some relationship between two cells by looking at the longest common prefix of their names. Inferred hierarchy information might be used, e.g., by a placement tool to put cells that are purportedly in the same hierarchy closer together.[4] Therefore, changing the hierarchy for the input (if it exists) may also change the solution.

4.3 Coarseness and Richness of Libraries

Libraries are a noise source that can be perturbed by the designer. Examples includes the cell library and the timing library. In standard cell design, the richness of the cell library will affect the solution: synthesis, place and route with a large number of standard cells and many variants for each cell type will have a different (not necessarily better) solution than with a limited number of master cells in the library.

Timing libraries can also be perturbed to create noise. In a typical timing library, the timing specification of a cell is usually defined by a set of timing models that characterize delays between input pins and output pins, as well as slew times for output transitions. Common approaches are table-lookup based, with interpolating from values provided in the table. The dimensionality of the table is defined by the number of axes: a typical delay model is specified by a 2-dimensional table, with axes corresponding to output load capacitance and input slew time. Changing the table

[3]The RNG is dependent on the machine and/or operating system: the same random seed with the same RNG may create two different sequences of random numbers on two different platforms.

[4]This is not always a good thing. Classic sources of ill-advised inferenc repeaters, clock buffers, and generated memory blocks.

dimensionality will change the timing model, e.g., by making cell delays independent of input slews. The table size (i.e., the number of points and entries in the table) determines the accuracy of the timing model, and can also be adjusted.

4.4 Constraints

Besides design data and libraries, the designer can also perturb *constraints*. Design rules capture physical process limits that in general cannot be violated without risking incorrect solutions. Hence, tightening of design rules is the only available perturbation. Design constraints, on the other hand, are imposed by the designer in order to meet a certain performance, power or area target. For example, in order to meet the target clock speed, a designer may add a number of timing constraints on timing arcs. If these constraints are met, the resulting design will likely meet targets; however, if there are violations in the constraints, the design may still work. Perturbations such as relaxing or tightening constraints, or choosing different sets of constraints, will give different solutions.[5]

4.5 Geometric Properties

Our last type of perturbation concerns geometric properties of the design. Since cells and nets must be embedded in a geometrical layout, perturbations to the geometric context may also give rise to variations in the solution. There are many possible perturbations that belong to this category; for brevity, we mention only some examples.

The first perturbation in geometric space is the change in offsets (or locations) of design instances. There are many design instances for which offsets can be perturbed, such as cell sites, cell rows, routing tracks, global cell grids for global routing, etc.

Changes in orientations are a second possible perturbation in geometric space. Perturbations with respect to orientations include changes in site orientations, pin orientations, row directions, and preferred direction of routing layers.

A third possible perturbation is the scaling of the instances. Instance scaling includes changes in layout size, cell sizes, routing pitches, row sizes, etc. A variety of considerations such as numerical accuracy and rounding lead to noise in the solution quality as a result.

Finally, noise can also be introduced by adding additional blockages into the design (this may also be viewed as a type of added constraint). For example, if we intentionally add some obstructions, the solution will be affected as cells are no longer placeable at certain sites, or nets are no longer routable along certain tracks.

5 Example of Effects of Noise

In this section, we present an example of behavior criteria test for current tools and some examples of sources of noise with their effects on solution quality. We focus our experiments on the place and route domain, and assume that observed effects will have parallels in other domains. In all of our experiments, we run Cadence Design Systems placement (QPlace version 5.1.61.2) and routing (WRoute version 2.2.28.2) tools. Although the magnitude of the effects might be different in other tools, we believe that the effects themselves are similar. The industry designs that we use in our experiments are outlined in Table 1.

We first assess the monotonicity behavior of the place and route tools. Second, we give empirical analyses with respect to various types of noise sources: (i) random seeds, (ii) cell and net initial ordering, (iii) cell and net naming, and (iv) name hierarchy. The cell/net ordering and naming experiments are essentially confirmations of most of the previous works (Ghosh, Bodapati et al., etc.).

[5]Consider that there are many possible "compact SDF" covers in SDF-based flows, or many combinations of, e.g., skew and path timing constraints that achieve a given system clock period.

Design Name	No. Cells	No. Nets
Design01	276	314
Design02	3286	2902
Design03	6145	6006
Design04	7703	5657
Design05	9011	11962
Design06	12133	11828
Design07	12261	13245
Design08	12857	10880
Design09	20577	25634
Design10	25995	28603
Design11	35549	44121
Design12	57275	49582
Design13	85572	87390

Table 1: List of designs used in the experiments.

5.1 Monotonicity Test

In this experiment, we test the place and route tools for their behavior against their input parameters. QPlace has a parameter (*OptimizationLevel*) that is used to determine the strength of the optimization algorithm. Its value ranges from 1 to 10, where 1 indicates the weakest but fast optimization while 10 indicates the strongest but slow optimization.

Opt Level	1	2	3	4	5	6	7	8	9
QP WL	2.50	0.97	-0.20	-0.11	1.43	0.58	1.29	0.64	1.70
QP WWL	1.98	0.57	-0.24	-0.31	1.34	0.45	1.26	0.70	1.78
QP CPU	-59.73	-51.64	-40.43	-39.33	-31.53	-31.32	-17.29	-11.86	-6.73
WR WL	2.95	1.52	-0.29	0.07	1.59	0.92	0.89	0.94	1.52
# Vias	1.59	0.41	-0.08	-0.35	1.39	1.11	0.47	0.76	0.92
Total CPU	4.19	-6.77	-16.21	-15.16	-7.23	-10.57	-6.99	-3.75	-0.51

Table 2: Average percentage effect of OptLevel < 10 on solution quality and CPU time.

For each test cases, we run the tools 10 times with different optimization levels. Table 2 shows the average percentage effect of OptLevel values < 10 on solution quality and CPU time, with all results normalized against the results for OptLevel $= 10$. As seen from the table, the placement tool does not behave monotonically. Indeed, running the placement tool with its strongest optimization level does not guarantee the best solution; it appears preferable to use a weaker optimization level (= 3) that uses less CPU time.

5.2 Random Seeds

Given the heuristic nature of most VLSI algorithms, randomization and tie-breaking will nearly always play some role. Thus, noise in the *random number generator* (RNG) will also be a noise source for the corresponding algorithm. A RNG generates a (pseudorandom) sequence of numbers that is a function of the initial seed value. Changing the sequence of random numbers changes the sequence of decisions made within the algorithm, thus creating noise. In our

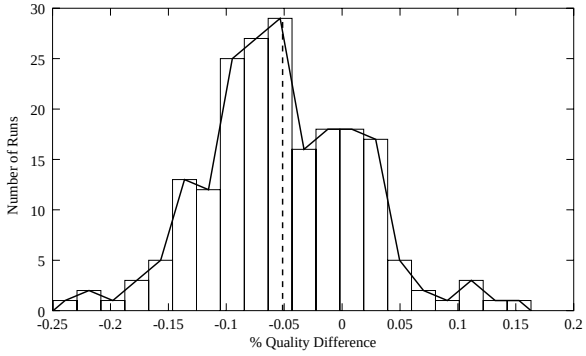


Figure 1: Frequency distribution (200 runs) of solution quality difference due to change in random seeds. The dashed line indicates the average solution quality over all runs.

experiment, we run the router 200 times on the same placement of Design06. For each run, we pick a unique random seed value and run both global and final routing with this seed value.

Figure 1 shows the frequency distribution of the percentage difference in solution quality with respect to the original solution (i.e., the routed solution using the default seed value). The solution quality is binned into 20 intervals, and the frequency plot is calculated based on these intervals. The dashed line on the plot indicates the difference of the average solution quality over all runs. Note that in this and all other figures, we equate "quality" with "cost" (e.g., routed wirelength cost), so that a negative change in "quality" indicates a better solution. From the figure, we can see that very slightly better solutions can be achieved just by varying the random seed values. The average solution quality of the multiple starts with different random seeds improve the quality by 0.05% for Design06.

5.3 Random Initial Ordering

The next experiment checks the effects of initial ordering of cells and nets. For each design, we reorder the cells and nets and run the place and route flow with the new ordering. We use the same (default) seed value for all runs to eliminate additional noise due to randomization. For all designs, the tools always give exactly the same solutions. This indicates that these tools perform an additional, canonical reordering after reading the data.

5.4 Random Naming

Since the physical ordering of cells and nets in the input files does not affect the solution, we extend the noise experiments to changes in naming. We apply three types of name change perturbation, for cells, nets and master cells.[6] For each perturbation (cell names, net names, and master cell names), we change the appropriate names in all designs into generic names such as `CELL014`, `NET042`, and `MCELL002` respectively. This is performed for all instances in the design, e.g., all cell names are changed into generic names, etc. Figure 2 shows the distributions of the percentage difference in solution quality (versus the original solution) for the three types of perturbations, for Design06. The data correspond to sets of 100 runs with each type of perturbation. From the figure, we can see that there is actually a higher probability to get better solutions with random naming; the magnitude of the variation is also nontrivial (e.g., net name perturbations lead to more than 7% spread in solution quality).

For some designs, hierarchy is encoded in the names. We perform additional experiments with random naming where the hierarchy is preserved. For example, if the original name is `/ABC/DDF/DEC/NAND15`, the replacement name may be `/ID0034/ID0067/ID0113/ID0226` where `ID####` are each randomly generated. Since master cell names do not have hierarchical data, we only perform the experiments on cell names and net names. Figure 3 shows the distributions of percentage difference in Design06 solution quality for cell name and net name perturbations that preserve hierarchy. We have performed additional experiments that combine changes in cell names with changes in net names, both with and without hierarchy preservation. We find that the dynamics of such noise combinations are not easy to elucidate; we defer this issue to Section 6.1 below.

5.5 Random Hierarchy

The last experiment that we present in this work tests reordering of the hierarchy of the cells. This experiment has some similarity with the previous name changing experiments. However, here we do not change the names: instead, we swap one existing cell name with another existing cell name. The resulting netlist will have the original hierarchical structure, but the assignment of individual instances within this hierarchy will be completely scrambled. Figure

[6]Names of pins in master cells could be the basis of a fourth type of name perturbation. However, since pin names are typically very simple (e.g., A, B, Y, etc.) we do not include them in our experiments.

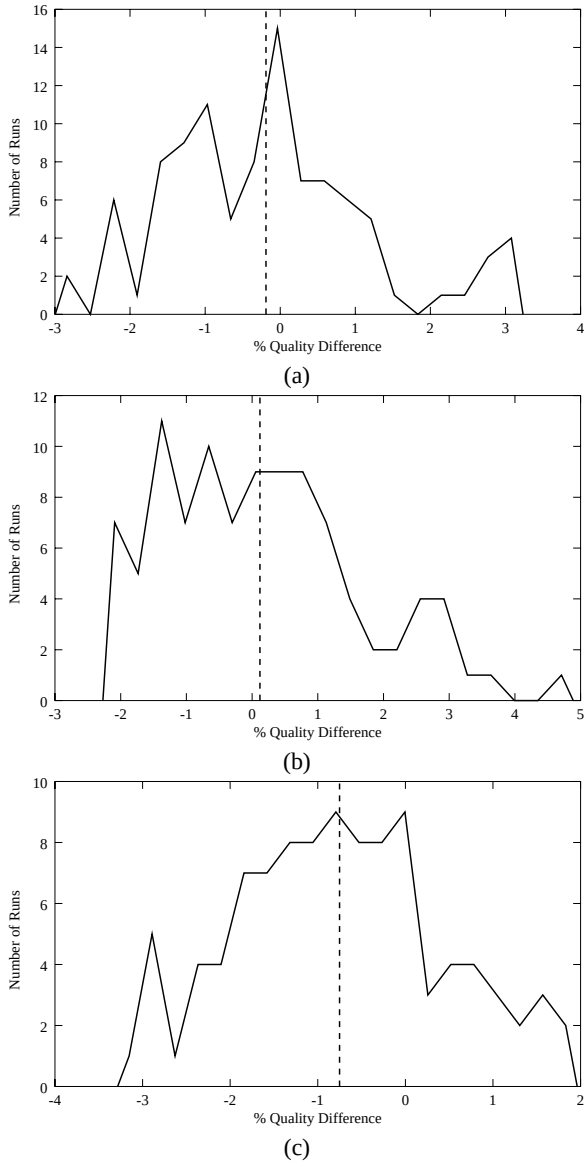


Figure 2: Frequency distributions over 100 runs of percentage difference in solution quality for Design06 (with respect to original solution) due to random naming perturbations: (a) cell names, (b) net names, and (c) master cell names.

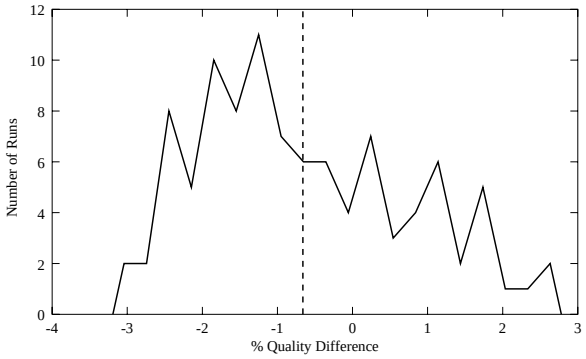
4 shows the distribution of solution quality difference due to this "hierarchy noise" for Design02. We can see that the original cell name hierarchy plays an important role in guiding the tools that we study. When the hierarchy information is corrupted, the solution quality degrades significantly (by up to 12%); on average, instance perturbation leads to a 5% loss of solution quality.

6 Noise Properties

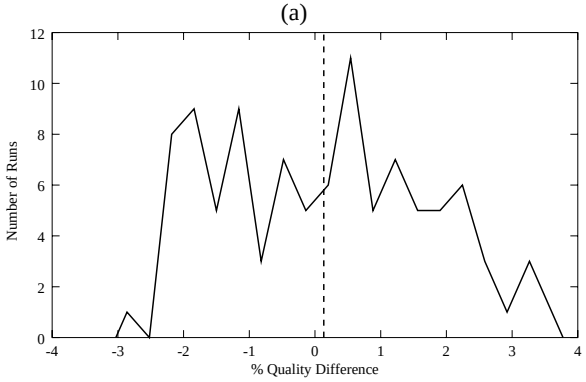
In this section, we briefly sketch two examples of how the basic noise measurements of the previous section may be extended. We first examine whether noise sources are "additive". We then consider aspects of solution quality distributions that may make certain noise sources more useful than others for design optimization.

6.1 Additivity of Noise

To find out whether the effects of various noise sources are additive, we test the effect of applying two types of name perturbations simultaneously. We use the randomization in cell naming ((the randomization in net naming (NR). For each noise sou



(a)



(b)

Figure 3: Frequency distributions of the percentage difference in solution quality for Design06 due to random naming perturbation that preserves hierarchy: (a) cell name perturbation, and (b) net name perturbation.

make 10 different noise generation runs with 10 different random seeds; there are 100 pairs of seeds for the first noise and for the second noise, and for each pair we apply the combined perturbation (CR-NR) to the design. Finally, for each solution obtained, we compare the difference from original solution quality with the differences induced by each noise perturbation (CR or NR) in isolation.

	NR									
CR	-1.59	-0.29	0.52	3.56	-0.72	-1.08	0.66	4.34	-2.78	1.58
0.83	0.84	1.81	1.03	2.40	2.90	-0.82	1.61	1.32	0.98	0.88
-0.57	-1.14	0.34	1.22	-0.20	0.42	-1.09	-0.34	1.41	-0.12	2.27
-0.85	0.57	-0.18	-0.46	-0.81	0.83	-0.97	0.70	-0.07	1.34	-0.35
-1.92	0.16	-0.69	1.07	1.38	0.86	0.84	-0.07	1.35	1.29	0.31
-0.37	-1.33	-1.00	1.06	0.98	1.26	-2.11	2.58	3.50	1.55	-1.81
1.74	-0.78	-0.72	1.10	-1.02	0.72	-0.82	3.00	0.09	1.39	0.52
-1.23	0.30	-1.10	1.77	1.46	0.77	1.39	-1.62	0.81	-1.04	0.09
-1.79	-0.00	0.73	1.96	1.84	0.72	2.59	-0.92	0.30	-0.17	-0.16
1.14	-0.69	-1.04	0.60	-0.84	1.39	0.51	0.10	-0.72	0.74	2.80
1.89	-0.23	1.02	-0.11	-0.06	0.93	2.20	1.51	1.12	0.42	1.10

Table 3: Comparisons between individual noise (random cell naming (CR) and random net naming (NR)) and the combination of those noises (CR-NR) for Design06. Entries are the percentage differences from original solution (without name change).

Table 3 lists the impacts of the individual perturbations (CR and NR) along with the impact of the combined perturbations (CR-NR) for Design06. The noises are not additive (i.e., CR-NR $\neq$ CR + NR), yet we see mildly positive correlation and rank correlation (0.1378 and 0.1579, respectively) between CR-NR and (CR + NR).

6.2 Potential for Exploiting Noise

After characterizing a given set of noise sources, it is natural to ask whether any of these noise sources are *useful* in design optimization. In this subsection, we examine whether noise can be exploited in the context of multi-start with a prescribed CPU resource. In this context, when the CPU budget is a single run, then we would invoke the noise source that has the best expected (i.e., mean) solution quality. However, if the CPU budget is, say, five runs, then we would invoke the noise source that has the best expected *minimum* solution cost over five runs: both mean and variance affect this decision.

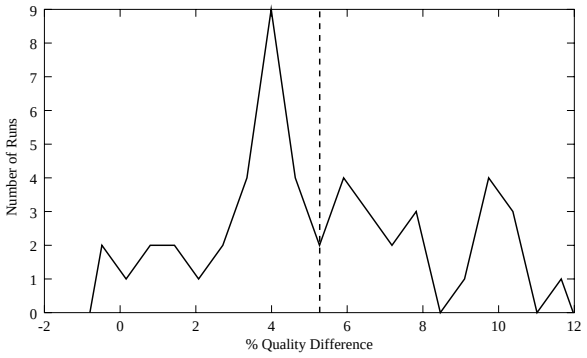


Figure 4: Frequency distribution of the difference in solution quality due to hierarchy noise. Data are collected over 50 runs of Design02. The dashed line indicates the average solution quality over all runs.

We study the utility of noise for the five variants of name-changing perturbations: random cell naming (CR), random cell naming while preserving hierarchy (CH), random net naming (NR), random net naming while preserving hierarchy (NH), and random master cell naming (MC). For each type of noise source, we randomly select sets of k solutions ($k = 1, 2, \ldots 30$) from 100 runs and record the best of the k solutions. We do this for 1000 sets of k solutions, and calculate the average solution (= expected best of k starts) over these 1000 trials.[7]

Table 4 shows the noise "quality" ranking for Design06. We see that the noise perturbation that preserves hierarchy almost always yields superior results for any given k-run CPU budget. However, if we only use one run, then changing the master cell name could be our best choice for exploiting noise in the place and route flow that we study. We observe that there are five distinct relative orderings of noise "utility" as k ranges from 1 to 30, depending on the CPU budget.

k	CR	CH	NR	NH	MC	k	CR	CH	NR	NH	MC
1	3	2	4	5	1	16	5	1	4	3	2
2	3	1	5	4	2	17	5	1	4	3	2
3	4	1	5	3	2	18	5	1	4	3	2
4	4	1	5	3	2	19	5	1	4	3	2
5	5	1	4	3	2	20	5	1	4	3	2
6	5	1	4	3	2	21	5	1	4	3	2
7	5	1	4	3	2	22	4	1	5	3	2
8	5	1	4	3	2	23	4	1	5	3	2
9	5	1	4	3	2	24	4	1	5	3	2
10	5	1	4	3	2	25	4	1	5	3	2
11	5	1	4	3	2	26	4	1	5	3	2
12	5	1	4	3	2	27	4	1	5	3	2
13	5	1	4	3	2	28	4	1	5	3	2
14	5	1	4	3	2	29	4	1	5	3	2
15	5	1	4	3	2	30	4	1	5	3	2

Table 4: Rank order (with respect to expected best-of-k solution quality) for five variants of name-changing perturbations, for $k = 1, \ldots, 30$ runs and the Design06 test case.

7 Conclusions

We have presented behavior criteria for EDA tools with respect to noise. These criteria allow potential future assessment of tool qualities with respect to noise. We also propose an initial taxonomy of perturbations that may introduce noise in design solutions. We believe that this taxonomy can help us better understand noise and its effects on design quality, e.g., we can build better estimators that consider the effects of noise. We have also presented results of several experiments that show effects of various noise sources on place-and-route solution qualities. The experiments indicate that

[7] If the number of possible solutions is less than 1000, e.g., the *best-of-1* case has only 100 possible solutions, then the maximum number of possible solutio instead of 1000.

noise impact can be quite large (ranges of up to 7% and 12% differences in solution quality were observed), and that noise may be usefully exploited in the multi-start context. Effects of noise are not additive, and it is unknown how different noise sources interact.

Numerous future directions can be followed. Our ongoing work includes an in-depth study and analysis of each source of noise to yield prediction models that include noise effects in their calculations. We also seek analyses of the relationships between different sources of noise, and the impact of noise sources on timing-driven solution quality. In addition, we are trying to uncover relationships between perturbation size and resulting changes in solution quality. Finally, we are studying the *composition* of noise between consecutive tools in the design flow (i.e., if each individual noise is modeled, we seek to model the composition of these noises).

References

[1] S. Bodapati and F. N. Najm, "Pre-Layout Estimation of Individual Wire Lengths", *Intl. Workshop on System-Level Interconnect Prediction*, April 2000, pp. 93-98.

[2] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Design and Implementation of Move-Based Heuristics for VLSI Hypergraph Partitioning", *ACM Journal of Experimental Algorithms* 5 (2000).

[3] D. Ghosh, "Generation of Tightly Controlled Equivalence Classes for Experimental Design of Heuristics for Graph-Based NP-hard Problems", *Ph.D. Dissertation*, Dept. of Electrical and Computer Engineering, North Carolina State University, May 2000.

[4] J. E. Harlow and F. Brglez, "Design of Experiments in BDD Variable Ordering: Lessons Learned", *Proc. Intl. Conference on Computer Aided Design*, November 1998, pp. 646-652.

[5] M. R. Hartoog, "Analysis of Placement Procedures for VLSI Standard Cell Layout", *Proc. Design Automation Conference*, July 1986, pp. 314-319.

[6] "International Technology Roadmap for Semiconductor", 2001.

[7] H. J. Kushner, "Asymptotic Global Behavior for Stochastic Approximation and Diffusions with Slowly Decreasing Noise Effects: Global Minimization via Monte Carlo", *SIAM Journal on Applied Mathematics*, vol. 47 (1), February 1987, pp. 169-185.

[8] S. Yakowitz and E. Lugosi, "Random Search in the Presence of Noise, with Application to Machine Learning", *SIAM Journal on Scientific and Statistical Computing*, vol.11 (4), July 1990, pp. 702-712.