On Robustness of EDA Tools

Jan Schmidt, Petr Fišer, Jiří Balcárek
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
schmidt@fit.cvut.cz, fiserp@fit.cvut.cz, balcaji2@fit.cvut.cz

***Abstract*— It is known that EDA tools produce results of different quality dependent on seemingly neutral details in the input. We bring further results in this direction, which show that the differences can impair any quantitative comparisons of the tools. To gain qualitative insight, we present a stochastic model of result quality based on Gaussian Mixtures. We show on three case studies how these models help to evaluate and improve EDA algorithms.**

Keywords- logic synthesis, randomness, robustness, stochastic models

I. INTRODUCTION

Present logic synthesis and optimization algorithms in Electronic Design Automation (EDA) [1], [2], [3], [4], [5] are heuristic. They do not guarantee optimum solutions, however they are supposed to produce solutions of sufficient quality. We will show that even this statement needs not be always true.

It has been shown that many synthesis processes are *not immune* to seemingly neutral details in the input, like the ordering of variables in the file header [6], [7] or small modifications in RTL statements [8]. In the case of the variables ordering [6], modification of only the source file header results in different result quality. Let us stress that the alternate inputs are *semantically equivalent*, e.g., they represent structurally equal circuits.

Authors of [8] recognized this problem for the first time and call the external bias as "*perturbations"* in the input description. They observed that the recent advancements in performance of logic synthesis tools are quantitatively comparable to the influence of these perturbations. Therefore, serious lack of robustness was reported.

The researched EDA tools are based on layered heuristics; iterative local search is the most common technique. Such techniques often depend on lexicographic order or other seemingly irrelevant aspects which nevertheless *bias* the result. As the tools are required to be deterministic, randomization cannot be used to achieve fairness. When it is done, as in [9], it appears to be an additional source of variance. The authors aptly call it "*CAD algorithm noise*". The algorithms themselves are then biased in an unknown way, which we call *internal bias.*

The vulnerability to external bias has, of course, internal causes and is of similar nature. Here, however, we distinguish the two because external bias can be manipulated without tool modification.

Although the heuristics are complex, for the purpose of our investigation a simple idea of an iterative local search is sufficient, as in Figure 1.

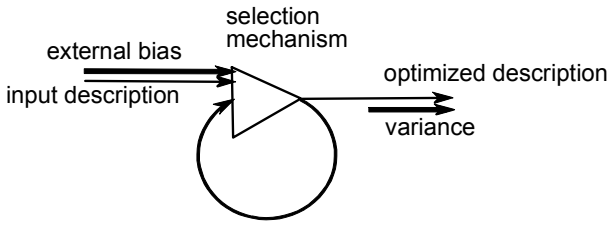

Figure 1. An iterative local search influenced by bias

The central idea of the presented paper, as well as [8], is to *manipulate* the bias more accurately to make it *random*. It enables the following:

- To *measure* the degree of influence of the bias and hence to *evaluate* the robustness of an algorithm, as in [8] and this paper.
- To construct a simple stochastic *model* of the influence, which is presented here, and thus to *understand* better the interaction between the tool and a task.
- To *randomize* a deterministic tool in order to achieve better results.

In Section II we experimentally analyze the influence of bias on the solution quality and present a stochastic model of the resulting distribution. This model enables us to state qualitative criteria for algorithms with acceptable quality. In Section III we describe three cases of algorithm evaluation and improvement. Sections IV and V conclude the paper.

II. EXPERIMENTAL OBSERVATIONS AND MODELING

Up to our knowledge, the first experimental evaluation of the influence of the external randomness, or perturbations in the source file, has been proposed in [8]. The authors have shown that modifying the source Verilog file, so that its functional equivalence is preserved, but some constructs are changed, yields synthesis results differing in units of percent (2.5% in area, 4% in slack, on average, 10% at most). This is approximately what the contemporary progress in improving the synthesis tools is. Therefore, they have rendered the experimental evaluation of tools inadequate, when performed in the form it has been performed until now. They also offered a tool introducing these perturbations into the source files, to help to perform relevant experimental evaluations.

A. Robustness of ABC

ABC [3], [4] is the current state-of-the-art academic logic synthesis and optimization tool. It comprises both combinational and sequential synthesis, mapping to standard cells or FPGA look-up tables (LUTs), verification, and many other features. It is controlled by *commands*. A sequence of commands constitutes a synthesis *script*. Each command implements a particular algorithm.

To briefly illustrate the lack of robustness in ABC, we will present results of only complete synthesis scripts:

- A script for standard cells mapping ("strash; dch; map"), which will be referred to as the "Gate mapping", and
- 4-LUT mapping ("strash; dch; if; mfs"), called "LUT mapping".

These scripts comprise of all the algorithms implemented for the suggested state-of-the-art synthesis. For details and more thorough analysis of ABC commands robustness see [10].

The experiments were conducted as follows: 264 benchmarks from the IWLS and LGSynth benchmarks sets [11], [12] were processed. Given a benchmark, its inputs and/or outputs were randomly permuted in the source file, and the resulting design size (gates, LUTs) was measured. This was repeated 1,000-times for each circuit. The average and maximum (over the 264 circuits) relative size differences between minimum and maximum values are reported in TABLE I.

We can see that almost all the average size differences are higher than 10%, with impressive maximum values.

TABLE I. ABC RESULTS

	Gate mapping	LUT mapping
Permuted inputs	8.67% (max. 74.38%)	11.50% (max. 92.14%)
Permuted outs.	10.52% (max. 70.47%)	12.60% (max. 85.42%)
Permuted both	13.40% (max. 86.27%)	14.81% (max. 95.07%)

B. Modeling and Analysis

To understand how the bias influences the process, we analyzed the frequencies of result sizes in the obtained data. Quite often, we saw distributions as in Figure 2, but we met also distributions as in Figure 3. A Gaussian distribution or a superposition of more Gaussian distributions can be expected, so we modeled the data by Gaussian Mixtures (GM), which are convex combinations of Gaussian distributions, called components of the model. We used EM algorithm [13] implemented in the em4gmm software [14] for modeling. The algorithm, of course, models the probability of all samples and therefore the histograms in the figures are for illustrative purpose only.

The modeling process confirmed that the Gaussian Mixture is a feasible model. In some cases, the components were so close – as in Figure 4 – so that two interpretations are possible: either we see really two superimposed components, or the model tries to estimate a non-Gaussian distribution. Our experience with probability density distribution allows us to conjecture that the first interpretation is correct, and that the difference between Figure 2 and Figure 3 is quantitative in principle. Nevertheless, such a long distance between solution clusters does indicate lack of robustness.

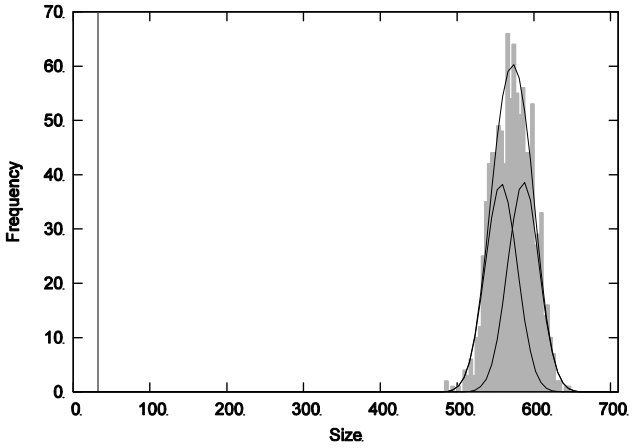

Figure 2. Result sizes of gate mapping on the *cordic* circuit, a model of the distribution, and the size of the best solution known (gray vertiacl line)

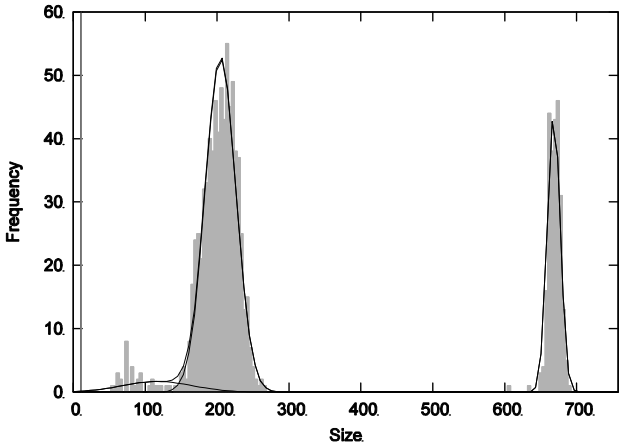

Figure 3. Result sizes of LUT mapping on the *cordic* circuit, a model of the distribution, and the size of the best solution known (gray vertiacl line)

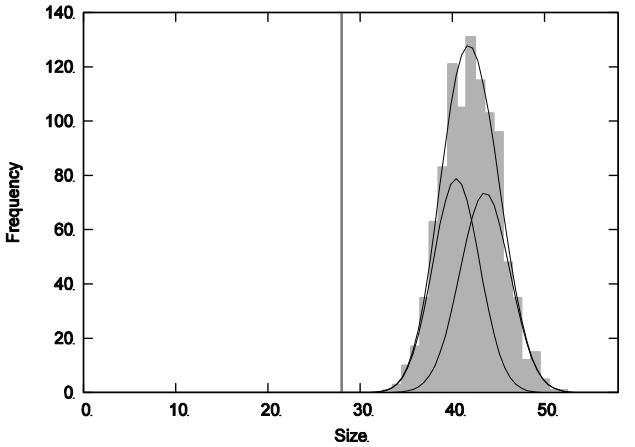

Figure 4. Result sizes of LUT mapping on the *br2* circuit, a model of the distribution, and the size of the best solution known (gray vertiacl line)

Using the GM model, we were able to classify the performance by the number of local maxima and components number. The numbers of benchmarks in these categories are summarized in TABLE II.

TABLE II. MODEL CLASSES BY BENCHMARK, ABC

Local max.	Components	Gate mapping		LUT mapping	
Invariant	Invariant		30		40
1	1	217	53	203	35
	2		148		150
	3		16		18
2	2	11	10	21	10
	>2		1		7
>2	>2		6		4

There are cases where unique solutions were produced, independently of variables ordering (Invariant). For these circuits, the algorithm is perfectly robust. However, these cases mostly represent rather small circuits, for which the globally optimum solution could be found easily.

The case with a single local maximum but two Gaussian components seems to be the most frequent. If our conjecture is true, then in many cases there are two different clusters of solutions alternately attracting the optimization process.

Cases with two or more local maxima and Gaussian components (see Figure 3) are relatively rare, however they happen. The optimization heuristic was probably mislead to a local optimum here, which could not be escaped.

From the user point of view, however, it is the probability of decent performance that matters, and therefore quantitative measures of robustness *and* quality are needed.

C. Quantification of Quality and Robustness

The quality of heuristic algorithms is commonly measured by known optima. As they are not known for the benchmarks, we used best solutions known for the evaluation. For the 224, resp. 234 circuits where ABC gate mapping resp. LUT mapping has been influenced by permutations, we obtained 203, resp. 190 high quality results by Cartesian Genetic Programming [15], [16] and massive iterations of ABC scripts [10].

The mean size given by the GM model was in all cases very close to the average size of all solutions. The former has been actually used for evaluation. In accordance with common terminology, we defined *relative quality* as the ratio of actual size to the best size known, even though it is equal to one for a perfect algorithm and *rises* in cases of suboptimum results.

Let us stress that the probability density distributions which approximate the histograms shown are intrinsically not symmetric. There must be a global minimum (even if we do not know it) for every circuit, cutting the distribution from the left. Moreover, the values at the left are no outliers, they are the cases where the algorithm worked best.

Robustness of a process is usually measured by comparing the disturbance of output with the disturbance of input, or as a value of input disturbance where the output starts to be disturbed too. In both cases, a measure of *input* disturbance is required. With permutations of the input description, unfortunately, no sensible measure can be found; if it is found, it will contribute much to our understanding of the process.

Having resigned on measuring input disturbance, we must resort to measuring the "width" of the histogram between two points and relating it to the mean value as *relative difference*. At the same time, the central point can be related to the best size known, and represent average *relative quality*. We considered measures based on three choices of boundary points:

1. The maximum and minimum.
2. The leftmost and rightmost inflexion point of the GM model.
3. The leftmost and rightmost inflexion point of the GM model, not taking into account small components. Smallness was approximated – rather ad hoc – as having the mixing coefficient less than 0.1.

As will be shown later in Section III.B, the first method gave the most reliable measure and will be as *relative difference* used in the rest of the paper. Notice that relative quality and relative difference together give the best and worse size.

The results for LUT mapping for the 264 benchmark circuits [11], [12] are presented below. The data for gate mapping, as TABLE II. suggests, have similar nature.

Figure 5 shows the distribution of quality. The maximum frequency is around 1.3 (30% above known minimum). However, 69 cases are above 2.0 and 14 are above 10.

Figure 6 is a similar plot for relative difference. The most frequent difference is 0.13, i.e. 13%, however cases as high as 35% are common. For reliable evaluation of a 5% improvement, this is not acceptable. Notice that a difference of 2 means 100%.

It may seem that there are simply difficult circuits and easy circuits. Figure 7 shows that this is not the case. Indeed, correlation between these measures is only 0.08.

LUT mapping and gate mapping do share procedures using the `strash` and `dch` commands. Their relative quality is only weakly correlated, however (factor 0.18, see Figure 8). The reason is that the iterative process managed to reach optimum in some cases, but preserved the original quality in others. The processes have similar robustness (factor 0.67, see Figure 9).

Now we can return to the idea of [8]. As the data prove, the influence of unpredictable bias is not only *comparable* to improvements usually reported by the research community, it is *much worse*. To compare two algorithms, the complete distributions should be reported. In the ideal case, the vulnerability to *all* sources of bias should be analyzed, which is, unfortunately, unattainable.

Examples of such comparisons will be shown in course of the following section.

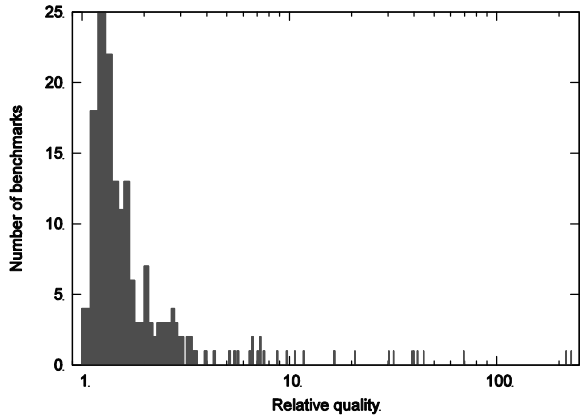

Figure 5. Numbers of benchmarks by relative quality for LUT mapping

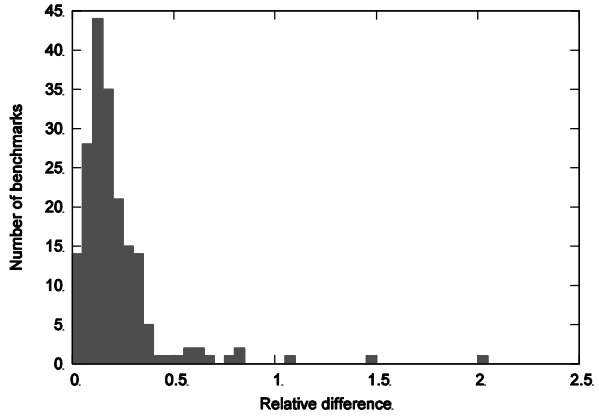

Figure 6. Numbers of benchmarks by relative difference for LUT mapping

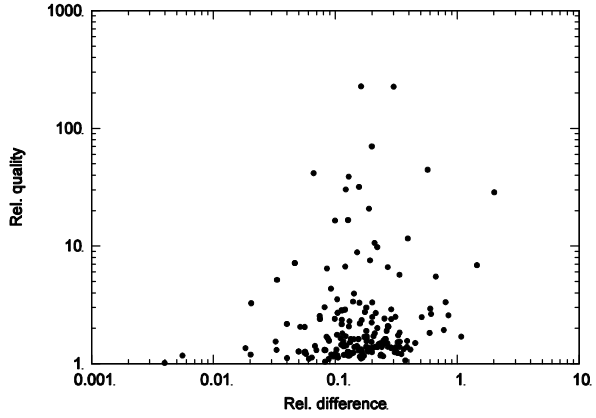

Figure 7 Relative quality versus relative difference for LUT mapping

III. ATTEMPTS AT REMEDY

In this section we present three case studies, aimed at overcoming the bias influence. Their central idea is rather obvious: a better-performing algorithm could suppress perturbations in the input better, and therefore will have better robustness. It is true, however, that this way we correct consequences, not causes.

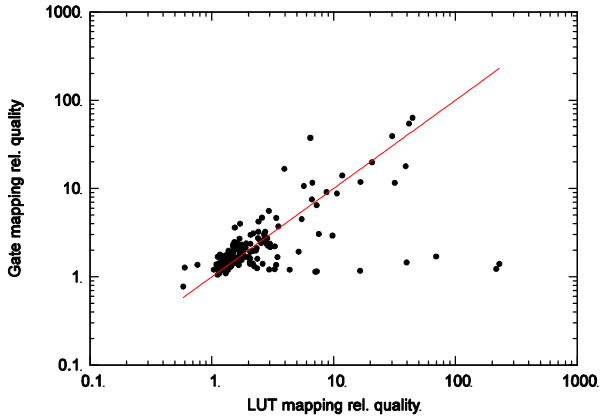

Figure 8. Relative quality of gate mapping versus LUT mapping

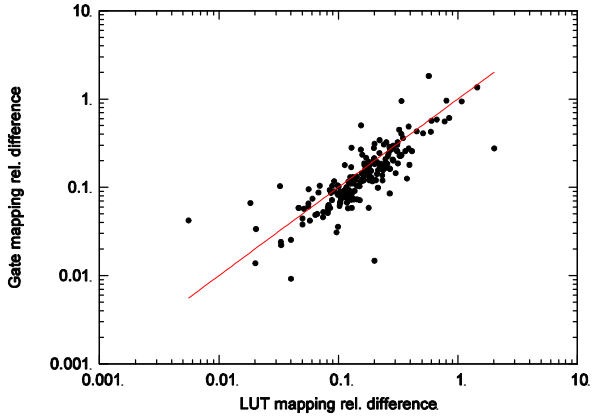

Figure 9. Relative difference of gate mapping versus LUT mapping

A. *Improving the SAT-Compress Test Compression Tool*

The SAT-Compress test compression tool [17] will be used as the first case study. The input to it is the circuit description (netlist) and it directly generates the compressed test sequence. It relies on a pre-generated fault list, which is produced by the Atalanta ATPG tool [18].

We identified three possible sources of bias:

1. The ordering of the fault-list generated by Atalanta,
2. the initial test pattern,
3. the structure of the input netlist.

The latter aspect comes from the fact that SAT instances are generated from the netlist, and then submitted to a SAT solver (MiniSAT [19] in our case). Different netlist structures directly induce different (but semantically equivalent) SAT instances, which may influence the SAT-solver execution. Different SAT solutions may be obtained and later used as test vectors. Therefore, properties of the employed SAT-solver may influence the robustness of the overall algorithm.

The lack of SAT-Compress robustness is reported in [7] in detail. The distributions of solutions of different quality follow the same Gaussian distributions as reported in Section II for ABC, for all the three above-mentioned aspects.

One way of increasing the robustness is to *systematically improve* the search algorithm. Such an improvement was the

recently published CPDCI (Coverage Preserving Don't Care Injection) technique, for details see [20].

Comparison of the standard SAT-Compress algorithm ("std") and the CPDCI enhancement is shown in TABLE III. for some selected benchmark circuits. We have processed 52 ISCAS [21], [22] and ITC'99 [23] benchmarks altogether, 1,000 executions each, with the above described random bias introduced. We have measured the average result size (bitstream length) and relative difference, as there are no high quality solutions available.

The average values from all the tested circuits are shown in the last table row. We see that the quality was improved by almost 10% on average, while the relative difference by 34%, which indicates a significant robustness improvement.

TABLE III. COMPARISON OF BASIC SAT-COMPRESS AND CPDCI

	Bitstream length (avg.)			Rel. difference		
	std.	CPDCI	impr.	std.	CPDCI	impr.
b01	32.9	32.1	2.3%	0.73	0.68	6.8%
b02	20.7	20.6	0.6%	0.72	0.78	-8.3%
b03	163.9	129.9	20.8%	1.0	0.62	38.0%
b04	956.2	796.9	16.7%	1.97	0.48	75.6%
c1355	320.0	302.5	5.5%	0.55	0.35	50.3%
c17	16.5	15.0	9.0%	0.91	0.53	36.3%
c1908	786.9	673.3	14.4%	0.71	0.42	40.8%
average			9.9%			34.1%

Detailed comparisons of the results obtained for two ISCAS'85 benchmark circuits [21] are shown in Figure 10 and Figure 11. Here the algorithm was run 2,000-times, with different random initial vectors.

As we see from the figures, the CPDCI distribution is narrower and displaced to the left from the basic SAT-Compress distribution, indicating both the increase of quality and robustness – CPDCI is less sensitive to the initial vector selection.

From Figure 10 the benefit of CPDCI is clear; there are two disjoint distributions. Here results of CPDCI will be better under any initial vector. However, the distributions in Figure 11 heavily overlap. Therefore, wrong conclusions can be derived from a simple comparison of single algorithms runs, where only one initial vector is used – such an initial vector can be accidently hit, so that it is advantageous for basic SAT-Compress (left part of the dashed histogram) and simultaneously disadvantageous for CPDCI (right part of the solid histogram). Hence, CPDCI may appear to be worse.

B. *Iterative Runs of ABC*

As the authors of ABC suggest, the synthesis in ABC can be run iteratively, as *resynthesis* [5], [10]. The entire synthesis script can be repeated (iterated) several times in order to improve the solution.

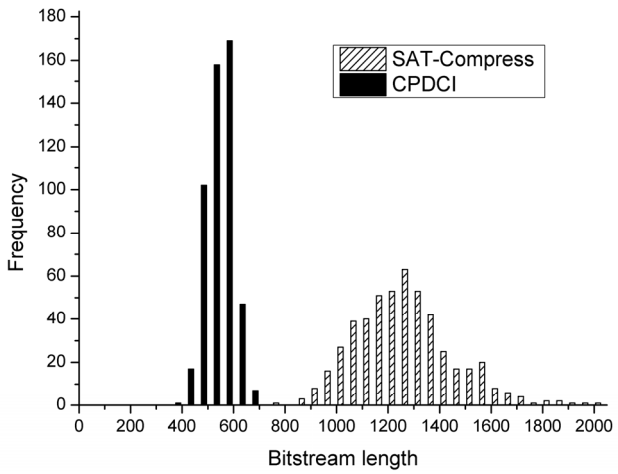

Figure 10. Comparison of SAT-Compress and CPDCI, the *c880* circuit

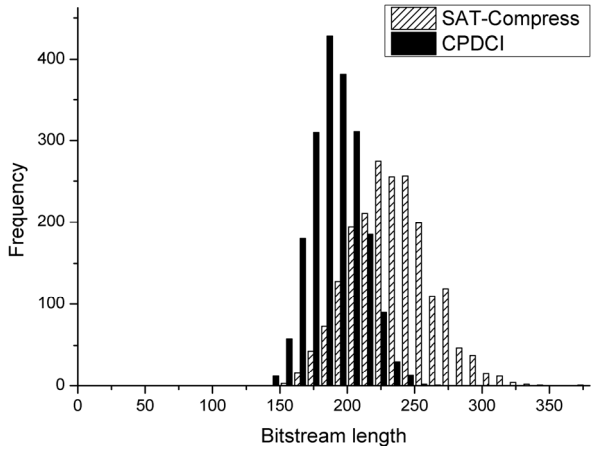

Figure 11. Comparison of SAT-Compress and CPDCI, the *c432* circuit

The following experiment uses the same process, benchmarks, and measurement as in Section II, except that we have iterated the synthesis 100-times, while observing the result quality at the end of each iteration. Figure 12 illustrates various aspects of the iterative process by its performance of iterated gate mapping on the *x1* circuit [11]. Similar behavior can be observed on the *cordic* circuit [11].

We can see that the components with high mean fade quickly, within first 10 iterations. The quality and robustness become acceptable, with difference below 50%. With further iterations, the performance still continues to improve and settles after further 20 iterations. The experiment was continued up to total 100 iterations. Even with such impractically great effort, the histograms show that the iterative algorithm is not entirely robust.

The fast initial convergence is not always the rule. The *alu4* benchmark has initially a neat and relatively narrow distribution with a single Gaussian component. We can clearly see what happens to this distribution in Figure 13 and Figure 14 during iteration. The mean value improves steadily along a smooth convergence curve. Somewhat surprisingly, the relative difference *increases*. At the beginning, the algorithm

is giving the wrong answer with great surety, and in the end it is giving a much better answer, but with less robustness.

Figure 13 also shows how treacherous can the histograms be without knowledge of existing solutions. The initial histogram suggests a robust process. Nevertheless, knowing that the mean value could be reduced to 76% and that there are solutions with 68% gates, the algorithm looks not so attractive.

Only the final design *size* (number of LUTs, gates) was measured here. The same analysis can be done for delay (number of levels) and similar results would be obtained. We will not present the results in this paper, due to a lack of space.

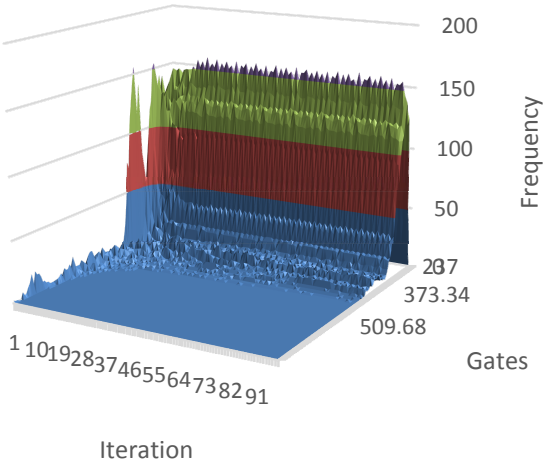

Figure 12. Gate mapping, *x1* circuit, 100 iterations

In general, iteration improves quality, as Figure 15 shows. Relations of relative qualities of a non-iterated process and the process iterated 40-times are given for all the tested 234 circuits. There are several cases where the improvement is substantial. From Figure 16 we can see, that iterating the ABC script actually *trades robustness for quality,* and that the behavior observed for the *alu4* circuit (Figure 13 and Figure 14) is rather common.

We have constructed comparisons similar to Figure 16 using other robustness measures mentioned in Section II.C. The correlation, which is 0.28 here, dropped to 0.19 for Method 3 and to 0.11 for Method 2. This leads us to the conclusion that measures using inflexion points lose information, and therefore Method 1 is preferable.

C. *Introducing Randomness Inside*

Until now, we have examined the influence of the *external* random bias. Now we may wonder what effect would have intentional introduction of randomness *inside*, i.e., introducing the *internal bias.* As a result, the deterministic process will become stochastic, different results will be obtained from different runs on the same input data. Here we will see that also both the average result quality and robustness of the process will be improved.

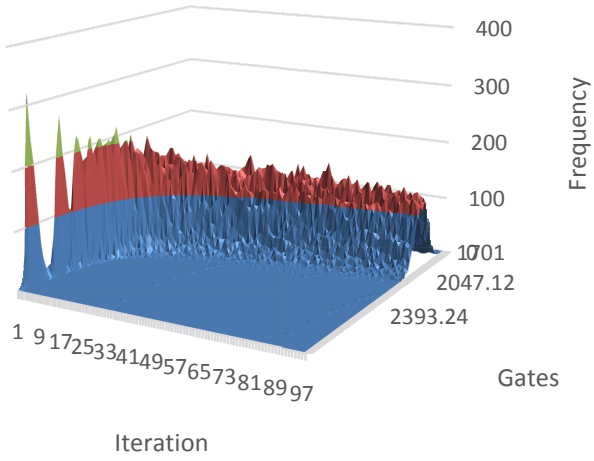

Figure 13. Gate mapping, *alu4* circuit, 100 iterations

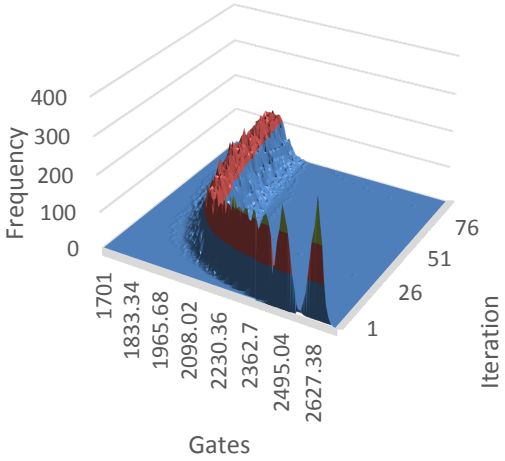

Figure 14. Gate mapping, *alu4* circuit, 100 iterations, from a different angle

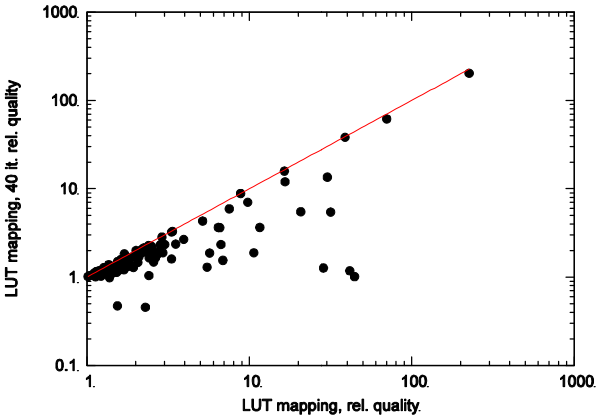

Figure 15. Relative quality of LUT mapping after 40 iterations

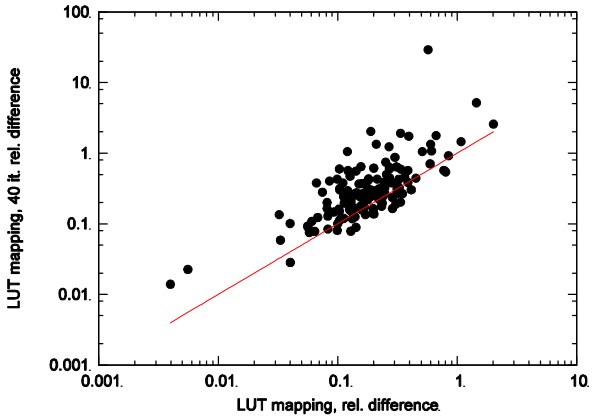

Figure 16. Relative difference of LUT mapping after 40 iterations

This experiment uses the same process, benchmarks, and measurement as the previous one (Section III.B). Randomness was introduced *inside* the iterative process; the ABC "permute" command randomly permuting input and output variables was executed at the beginning of every iteration. Again, the process was executed repeatedly, 10,000-times this time, to obtain statistically more significant results.

To illustrate the effect of randomness introduced into the process, the distributions of the result quality (measured from the 10,000 runs) is shown in Figure 17, for an example circuit *frg2*, where the robustness was improved most. Here it is apparent that both quality and robustness have been improved.

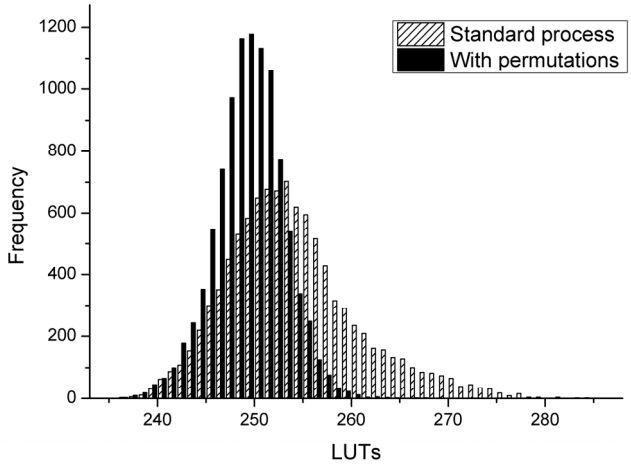

Figure 17. Distribution of solutions – *frg2*

Figure 18 and Figure 19 show how the introduction of randomness changed the performance of the iterative process. We see that relative quality was improved, while relative difference remained the same on average, and did not change much in most cases.

The combination of iterations and randomness injection provides algorithm which still has a better quality (Figure 20) than the original ABC script and better randomness (Figure 21) than the iterative process.

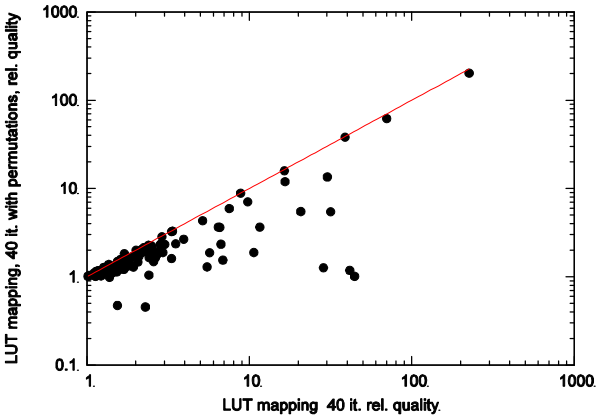

Figure 18. The influence of introduced randomness on relative quality

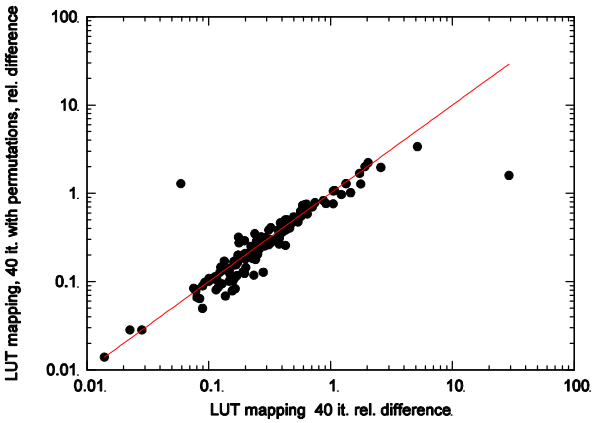

Figure 19. The influence of introduced randomness on relative difference

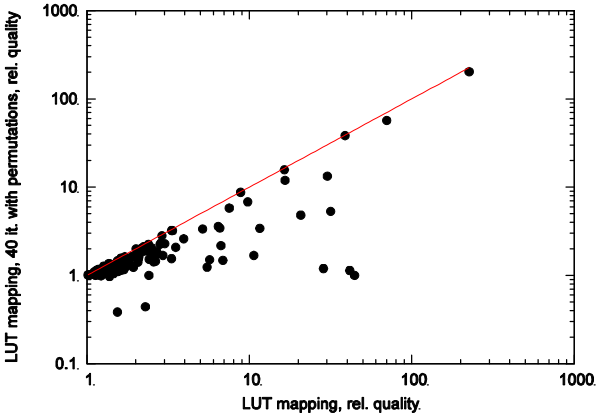

Figure 20. Combined influence of iteration and randomness on rel. quality

IV. Open Problems

No systematic approach to the detection of bias sources is known. The problem can be attacked bottom-up, by inspection of the code as in [9], or top-down, by injecting all possible perturbations into input descriptions [7], [8].

To our knowledge, the probability density distribution of performance (quality, time) in randomized algorithms has not been studied. Hence, our conjecture that individual GM components represent clusters of solutions cannot be verified.

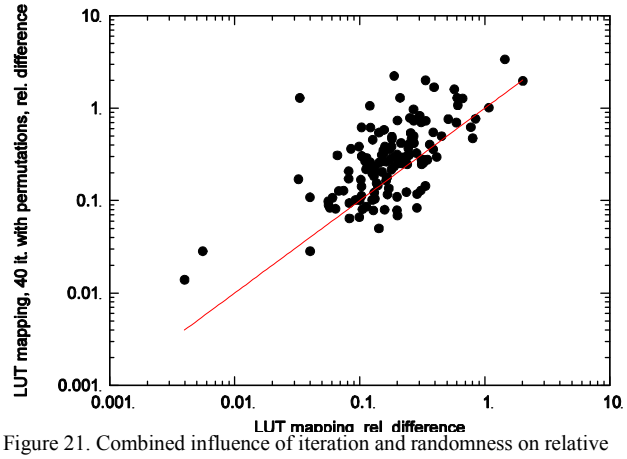

Figure 21. Combined influence of iteration and randomness on relative difference

The evaluation presented here shows the importance of optimum results to the usefulness of the benchmark set. Because the optima cannot be found by brute force, an infrastructure where the optima will be contributed and kept will be of benefit.

V. Conclusions

In this paper we have shown that EDA tools are vulnerable to external bias to a much greater degree than reported. By randomizing the bias, we obtain distributions that can characterize the robustness of the algorithm. These distributions can be modelled by Gaussian Mixtures and further classified. From various measures of robustness, relative difference seems to provide most relevant information. The combined observations of quality and robustness provide better insight into the properties of optimization processes.

Relative quality and relative difference are different properties. A new algorithm can improve both quality and robustness, but can also trade robustness for quality.

Existing, even deterministic, algorithms can be iterated and the convergence of the iterations can be controlled by injection of internal random bias, effectively turning the fault into an advantage.

Acknowledgement

This research has been supported by the grant of the Czech Technical University in Prague, SGS14/105/OHK3/1T/18.

The authors thank dr. Rudolf B. Blažek, CTU for a kind introduction into Gaussian Mixtures and for discussions that improved our insight.

References

[1] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*, Boston, MA, Kluwer Academic Publishers, 1996, 564 p.

[2] S. Hassoun and T. Sasao, *Logic Synthesis and Verification*, Boston, MA, Kluwer Academic Publishers, 2002, 454 p.

[3] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification", http://www.eecs.berkeley.edu/~alanmi/abc/

[4] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial Strength Verification Tool," in Proc. of the 22nd International Conference on Computer Aided Verification, Edinburgh, UK, July 15-19, LNCS 6174 6174, Springer, 2010, pp. 24-40.

[5] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang. "Scalable don't-care-based logic optimization and resynthesis", ACM Trans. Reconfigurable Technology and Systems (TRETS), Vol. 4(4), April 2011, Article 34.

[6] P. Fišer and J. Schmidt, "How Much Randomness Makes a Tool Randomized?," in Proc. of the 20th International Workshop on Logic and Synthesis (IWLS), San Diego, CA, USA, 2011, pp. 136-143.

[7] P. Fišer, J. Schmidt, and J. Balcárek, "Sources of Bias in EDA Tools and Its Influence," in Proc. of 17th IEEE Symposium on Design and Diagnostics of Electronic Systems (DDECS), Warsaw (Poland), April 23-25, 2014, pp. 258-261.

[8] A. Puggelli, T. Welp, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Are Logic Synthesis Tools Robust?," in Proc. of the 48th ACM/EDAC/IEEE Design Automation Conference (DAC), June 5-9, 2011, pp. 633-638.

[9] W. Shum and J. H. Anderson, "Analyzing and predicting the impact of CAD algorithm noise on FPGA speed performance and power," in Proc. of the ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA), 2012, pp. 107-110.

[10] P. Fišer and J. Schmidt, "On Using Permutation of Variables to Improve the Iterative Power of Resynthesis," in Proc. of 10th Int. Workshop on Boolean Problems (IWSBP), Freiberg (Germany), September 19-21, 2012, pp. 107-114.

[11] K. McElvain, "LGSynth93 Benchmark Set: Version 4.0", Mentor Graphics, May 1993.

[12] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide", Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, NC, January 1991.

[13] D. Titterington, A. Smith, U. Makov, "Statistical Analysis of Finite Mixture Distributions", Wiley, 1985, p. 243.

[14] Juan Daniel Valor Miró: "Fast clustering Expectation Maximization algorithm for Gaussian Mixture Models." Accessed at https://github.com/juandavm/em4gmm

[15] P. Fišer, J. Schmidt, Z. Vašíček, and L. Sekanina, "On Logic Synthesis of Conventionally Hard to Synthesize Circuits Using Genetic Programming," in Proc. of 13th IEEE Symposium on Design and Diagnostics of Electronic Systems (DDECS), Vienna (Austria), April 14-16, 2010, pp. 346-351.

[16] Z. Vašíček and L. Sekanina, "On Area Minimization of Complex Combinational Circuits Using Cartesian Genetic Programming," in IEEE World Congress on Computational Intelligence, 2012, pp. 2379-2386.

[17] J. Balcárek, P. Fišer, and J. Schmidt, „Techniques for SAT-based Constrained Test Pattern Generation," in Microprocessors and Microsystems, Vol. 37, Issue 2, March 2013, Elsevier, pp. 185-195.

[18] H.K. Lee and D.S. Ha, "Atalanta: an Efficient ATPG for Combinational Circuits," Technical Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1999.

[19] N. Éen, N. Sorensson, "An extensible SAT-solver," in Lecture Notes in Computer, Science 2919 - Theory and Applications of Satisability Testing. Springer Verlag, Berlin Heidelberg New York, 2004 pp. 333-336.

[20] J. Balcárek, P. Fišer, and J. Schmidt, "Simulation and SAT Based ATPG for Compressed Test Generation," in Proc. of 16th Euromicro Conference on Digital Systems Design (DSD), Santander (Spain), September 4-6, 2013, pp. 445-452.

[21] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," in Proc. of the International Symposium on Circuits and Systems, 1985, pp. 663-698.

[22] F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," in Proc. of the International Symposium of Circuits and Systems, 1989, pp. 1929-1934.

[23] F. Corno, M.S. Reorda, G. Squillero, "RT-level ITC`99 benchmarks and first ATPG results," in Proc. of the IEEE Design and Test of Computers, 2000, pp. 44-53.