

Design of experiments and evaluation of BDD ordering heuristics

Justin E. Harlow III¹, Franc Brglez²

¹Department of Electrical and Computer Engineering, Box 90291, Duke University, Durham, NC 27708, USA;
E-mail: justin.harlow@duke.edu

²Collaborative Benchmarking Laboratory, Department of Computer Science, Box 8206, North Carolina State University, Raleigh, NC 27695, USA

Published online: 15 May 2001 – © Springer-Verlag 2001

Abstract. Traditional approaches to the measurement of performance for CAD algorithms involve the use of sets of so-called “benchmark circuits.” In this paper, we demonstrate that current procedures do not produce results which accurately characterize the behavior of the algorithms under study. Indeed, we show that the apparent advances in algorithms which are documented by traditional benchmarking may well be due to chance, and not due to any new properties of the algorithms. As an alternative, we introduce a new methodology for the characterization of CAD heuristics which employs well-studied *design of experiments* methods. We show through numerous examples how such methods can be applied to evaluate the behavior of heuristics used in BDD variable ordering.

Key words: Benchmarking – BDD – Design of experiments

1 Introduction

The choice of BDD variable order has a profound impact on the size of the BDD data structure. Determining an optimal variable ordering is an NP-hard problem upon which much research has focused, e.g., [1–3]. Despite the many extensions to the basic decision diagram concept, outlined in [4], the ROBDD (*Reduced-Order Binary Decision Diagram*) form remains the most widely used variant. Evaluation of the quality of static and dynamic orderings of ROBDDs has typically been reported in terms of the peak or final size of the data structure for sets of “benchmark” circuits [5–7]. However, as we will demonstrate, these results are subject to serious ex-

perimental error, and often do not represent the general behavior of the ordering algorithms.

We introduce methods rooted in the *design of experiments*, first formalized in [8], to evaluate the properties of variable ordering heuristics. Our approach is radically different from the ones reported in the past, including [2, 9]. Rather than evaluating the algorithms on the basis of relatively few unrelated instances of known benchmark circuits, such as available in [5–7], we evaluate the algorithms on the basis of *a relatively large number of related instances* of circuits, each belonging to an *equivalence class*. Any circuit from the standard benchmark sets, or indeed any circuit at all, can serve as a reference circuit to form an equivalence class in the proposed experiments. In this paper, we utilize a simple *isomorphism class* generated from a reference circuit in order to illustrate the experimental principles. The background and a broader context for this approach is detailed in [10–13].

The paper is organized as follows: in Sect. 2 we introduce the fundamental ideas of design of experiments, both in general terms and as specifically applied to the evaluation of BDD ordering heuristics.

In Sect. 3 we review the traditional methodology used in CAD algorithm evaluation, and highlight its shortcomings through illustrative examples.

In Sect. 4, we give an overview of recent work on function classification and discuss the problem of selecting appropriate equivalence classes.

The main conclusions of the paper and directions for future research are summarized in Sect. 5.

2 Design of experiments

NP-hard problems, such as optimal variable ordering for BDDs, are solved by devising a polynomial-time heuris-

tic, generally with no guarantee whatsoever of the quality of the solution. Changing the starting point for the problem instance can induce unpredictable variability of results when experiments are repeated. To systematically study the behavior of such heuristics, we employ the fundamental principles of experimental design, *randomization*, *replication*, and *organization to reduce error*, first formalized by R.A. Fisher in the 1920s when analyzing problems in agriculture [8]. In the context of experimental evaluation of algorithms, these principles are embodied in the creation of a *circuit equivalence class*, and repetition of the experiments for each member in the class.

2.1 DoE methodology for CAD experiments

Figure 1 illustrates a generic flow for the type of experiments presented in this paper. The principal abstractions of such an experiment are as follows:

1. **Circuit equivalence classes.** To “organize to reduce error,” it is necessary to evaluate the algorithm for a set of closely related circuits. In biomedical experiments, a class consists of, for example, a set of subjects with closely controlled characteristics, such as sex, age, weight, etc. In CAD experiments, we create classes of circuits which are invariant in certain important properties, such as gate count, input and output counts, fanin and fanout distributions, entropy, etc. Definitions of sufficiently discriminating graph-based invariants that support generation of circuit equivalence classes for experimental design are an active area of research [14]. In this paper, we focus on the *logically equivalent* isomorphism equivalence classes, discussed in the next section. For more context on the entropy of Boolean functions, see Appendix A.
2. **Treatments.** In medicine, a “treatment” may consist of the application of a drug to the subjects in the experimental group, with the objective of inducing some desired response (e.g., reduction of blood pressure). In CAD experiments, a treatment is the application of some algorithm to each member of a circuit equivalence class, with the objective of minimizing some cost function.

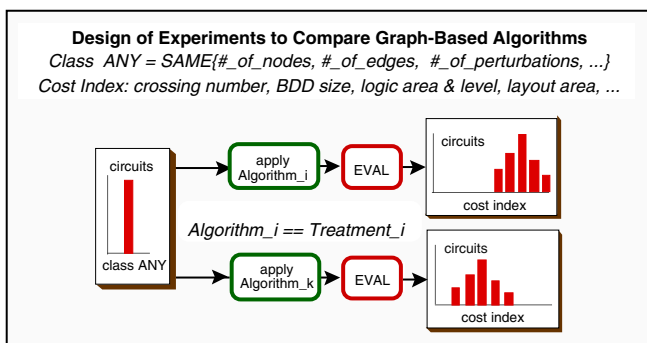


Fig. 1. Generic design of experiments flow for comparing two algorithms

3. **Evaluation.** In medicine, “evaluation” might mean measuring the blood pressure of the experimental subjects. In CAD experiments, evaluation of the results of an experiment consists of calculating the values of the cost function(s) of interest, and examining their frequency distributions(s). Cost functions vary for different CAD applications; in the experiments presented in this paper, we tabulate the final BDD sizes for each instance in a circuit equivalence class.

Figure 1 schematically illustrates a typical experiment based on the ideas above: a circuit equivalence class, having known invariant properties, is processed by two treatments, *algorithm_i* and *algorithm_k*. The results are evaluated and displayed as histograms of the distribution of the cost indices for each treatment. The experiments in this paper follow this general flow. Other types of experimental flows used in CAD algorithm studies are discussed in [12].

2.2 Definition of circuit equivalence classes

In this section, we formalize the notion of a circuit equivalence class. A circuit netlist is represented as a directed graph $G = (V, E)$ comprised of a set V of vertices representing the circuit elements (typically, gates) and a set E of edges representing the connections among the vertices. A circuit equivalence class consists of a set of replicas of a *reference circuit*, G_r , which have been randomly modified according to specific rules for the class. We next define two classes, the *isomorphism class* and the *signature invariant class*.

1. **Isomorphism class.** The most fundamental circuit equivalence class is the isomorphism class, consisting of circuits which are logically identical and graph-isomorphic instances of a reference circuit. The class is constructed as follows:

- Starting with a reference circuit G_r , create a family of identical circuits with randomly reordered netlists.
- For each instance in the class, replace the original variable labels with randomly generated character strings.

The isomorphism class is formally defined as follows:

$$\text{Reference Circuit: } G_r = (V, E) \quad (1)$$

$$\text{Reordered, Relabeled: } V' = rr(V)$$

$$\text{Circuit Instance: } G_i = (V', E)$$

$$\text{Isomorphism Class: } \mathbf{G}_{\text{iso}} = \{G_i | i = 1 \dots n\}$$

The cost function associated with an algorithm over an isomorphism class has a characteristic distribution with a specific mean and a variance greater than or equal to zero. In the case where *variance* = 0, we say that the algorithm is *specification-order independent*.

2. **Signature invariant classes.** Signature invariant classes are formally defined as follows:

$$\begin{aligned} \text{Reference Circuit: } G_r &= (V, E) & (2) \\ \text{Reordered, Relabeled: } V' &= rr(V) \\ \text{Circuit Instance: } G_i &= (V', E) \\ \text{Signature of } G_i : H_i &= H(G_i) \\ \text{Signature Invariant Class: } \mathbf{G}_{\text{si}} &= \{G_i \mid i = 1 \dots n, \\ & H_j = H_k, \forall j, k\} \end{aligned}$$

This formal definition does not describe the specific graph perturbations used to generate instances of the reference circuit. Many different signatures have been proposed, for instance see [15]. In Sect. 4, we discuss the issue of selecting appropriate equivalence classes in more depth.

2.3 Definition of experimental treatments

In this section, we describe the set of experimental treatments used in this paper, tabulated in Table 1. We use the VIS 1.1 and VIS 1.3 software[16] as a convenient framework for conducting experiments, but the procedures are general and may be readily adapted to any particular platform. Construction and optimization of a BDD consists of three distinct phases:

1. **Initial ordering.** An initial variable ordering must be determined prior to construction of the BDD data structure. Several different heuristics are available in VIS. In some treatments, we use the *natural order*¹ of the primary input variables as the initial order. In these cases, the static ordering heuristics of VIS are *disabled* as noted in Table 1. For other treatments, we enable the default static ordering heuristic and VIS establishes the initial variable order.
2. **BDD construction.** The actual BDD data structure is constructed by one of the BDD packages in the VIS software (for most experiments, we use the CU [17] package). For certain treatments, we enable *dynamic variable ordering* using the *sift* heuristic [1]. In these treatments, when the BDD data structure exceeds a predetermined size, the reordering heuristic attempts to reduce the data structure size. In other treatments, this feature is disabled.
3. **Dynamic variable reordering.** In some treatments, we *force* dynamic reordering after BDD construction is complete². In some treatments, we do this repeatedly. The *sift-converge* procedure is used in some treatments; this process performs sifting repeatedly until no further improvement is achieved.

¹ “Natural order” refers to the exact order in which the primary input variables occur in the netlist file.

² This process is commonly used in CAD applications to further reduce BDD size after construction.

Table 1. Standard BDD treatments

Treatment	Static Ordering	Dynamic Reordering (sift)
0	Disabled	None
1	Enabled	None
2	Enabled	Enabled during construction
3	Enabled	Enabled during construction Forced once after construction
4	Enabled	Enabled during construction Forced twice after construction
5	Enabled	Enabled during construction Sift-converge after construction
6	Disabled	Enabled during construction
7	Disabled	Enabled during construction Forced once after construction
8	Disabled	Enabled during construction Forced twice after construction
9	Disabled	Enabled during construction Sift-converge after construction

Table 1 itemizes a total of ten distinctive treatments (BDD variable ordering algorithms) that we apply to circuit equivalence classes in this paper, showing the specific static- and dynamic-variable ordering options selected in each case. These represent both typical sequences used in practical applications (e.g., treatments 3 and 4) and some unusual cases designed to expose possible instabilities in the algorithms (e.g., treatments 0 and 6). The details of each treatment are as follows:

- Treatment 0 designates a treatment in which the variable order is solely determined by the order of the primary input variables, given by each instance of the netlist in the designated class. We refer to this order as a “natural order”, i.e., the order given by the netlist alone. For treatment 0, it is this order, and this order alone, which determines the size of the BDD. Since node and input variable orders in any of the classes are randomized³, treatment 0 is equivalent to sampling random variable orders and reporting the resulting BDD size.
- Treatment 1 designates a treatment in which the variable order, given the natural order of the circuit instance, is transformed into a new order by the static variable ordering heuristic.
- Treatment 2 designates a treatment in which the variable order, given the natural order of the circuit instance, is optimized into a new order by the static variable ordering algorithm, and then further optimized concurrently with BDD construction using the “sift” algorithm.
- Treatments 3–5 designate treatments in which the variable order, given the natural order of the circuit instance, is optimized into a new order by the succession

³ Exception: class ALU4r-in-WD, described in Sect. 2.4.

of treatment 1, treatment 2, and a single-, a double-, or a multiple-sifting operation, respectively.

- Treatment 6 designates a treatment in which the variable order, given the natural order of the circuit instance, is optimized into a new order by the “sift” algorithm during BDD construction, bypassing the static variable ordering algorithm. This treatment measures the effectiveness of the dynamic reordering algorithm in reducing the final BDD size when the initial ordering is random.
- Treatments 7–9 are equivalent to treatments 3–5, except for bypassing the static variable ordering algorithm. These treatments also measure the effect of the dynamic variable reordering algorithm on the final BDD size.

2.4 Application of treatments to equivalence classes

In this section, we illustrate the use of experimental treatments and circuit equivalence classes with two classes derived from the reference circuit `ALU4r`⁴. We use two equivalence classes, consisting of 128 instances which are graph-isomorphic and logically equivalent to the reference circuit, as follows:

- `ALU4r-in-WD`. In this class, each instance is a replica of the reference circuit, in which the netlist has simply been randomly reordered.
- `ALU4r-rn-WD`. Each instance is a replica of the reference circuit, in which the netlist has been randomly reordered, and the variables have been randomly relabeled. This is a true isomorphism class, as defined in equation (1).

Figure 2 summarizes the distributions of final sizes for 1,536 individual BDDs⁵, constructed by VIS 1.1 from the two equivalence classes above, under treatments defined in Table 1. This experiment establishes the importance of using isomorphism classes as defined in equation (1) in any experiments with BDD algorithms.

2.5 Statistical analysis of results

In the context of this paper, the basic statistical decision we want to make involves comparison of two treatments. The objective of each treatment is to reduce the size of each BDD, given an equivalence class of Boolean functions. We use hypothesis testing and evaluate the t -statistic to compare the merits of two treatments,

a methodology described in introductory texts on statistics [20, 21]. A brief tutorial and an illustrative example about this statistic is included in the Appendix. Examples of more advanced methods about multiple comparisons of treatments are described in [22].

For classes of 128 instances reported in this paper, the mean values of two distributions are significantly different, at the 95% confidence level, if $|t| \geq 1.969$. In other words, when $|t| \geq 1.969$, the differences between the observed values of two sample sets are due to the heuristics, and not to due to chance.

The means and standard deviation of the distributions of BDD sizes for all ten treatments, for both classes `ALU4r-in-WD` and `ALU4r-rn-WD`, are tabulated in the table at the left of Fig. 2. Using these values, the t -statistic may be computed for any pair of distributions to test for significant differences in the means. For instance, for class `alu4r-rn-WD`, we compute $t = 0.266$ for the distributions of treatment 0 and treatment 1; thus, we accept the hypothesis \mathcal{H}_0 at the significance level of 0.05. In other words, we are 95% confident that for the equivalence classes in question, treatment 1 is not significantly different from treatment 0; the apparent differences observed in the figure are due to chance rather than to any properties of the algorithms.

Similarly, for class `ALU4r-rn-WD`, we calculate $t = 7.20$ for treatments 3 and 4, indicating that the observed difference between the means of the two distributions is significant at the 95% confidence level; the difference is due not to chance, but to the performance of the heuristic (here, the second post-construction sift) being examined.

We have described here only one of the possible statistical tests that may be applied in conducting experiments on CAD algorithms. More sophisticated techniques are described in standard texts on statistics, e.g., [20].

2.6 Lessons learned

In the experiment described in the previous section, we established that the static variable ordering algorithm in VIS 1.1 was sensitive to the *labeling* of the variables. Results for the class `ALU4r-in-WD` differed significantly from those for class `ALU4r-rn-WD` under identical treatments (treatment 1) in which static ordering was enabled. In addition, results for `ALU4r-in-WD` differed between treatments 0 and 1, where the only difference in treatment is that the static ordering heuristic of VIS is enabled for treatment 1. From these observations, we were able to conclude that there was a defect in the static ordering procedure in VIS, which was later confirmed by the developers of the software [23]. This example clearly demonstrates the power of DoE techniques in the characterization of CAD algorithms.

At a more general level, the observed sensitivity to variable labeling underscores the importance of equivalence class selection and randomization in any statistical

⁴ Based on the standard 74181 circuit. A schematic and description of this circuit may be found in [18]. The version used here is based on a textbook schematic of a 4-bit ALU circuit, independently encoded by J. Calhoun [19], and subsequently by D. Ghosh in 1998 and verified against each other. The standard benchmark circuit `alu4` in [7] differs from this version in three minterms, verified by J. Harlow in 1998.

⁵ 2 classes of 128 graph instances, processed under 6 different treatments, resulting in 1,536 individual BDDs.

Treatment	Circuit Class ALU4r-in-WD		Circuit Class ALU4r-rn-WD	
	Sample Mean	Sample StD	Sample Mean	Sample StD
0	1,236.0	169.9	1,236.0	169.9
1	1,206.0	0.0	1,268.1	159.4
2	1,206.0	0.0	1,268.1	159.4
3	598.0	0.0	707.9	88.6
4	577.0	0.0	637.4	65.7
5	576.0	0.0	631.3	58.7
6	1,236.0	168.9	1,236.0	168.9
7	710.3	74.3	710.3	74.3
8	643.6	63.7	643.6	63.7
9	631.3	62.9	631.3	62.9

Summary of statistics for Treatments 0–9.

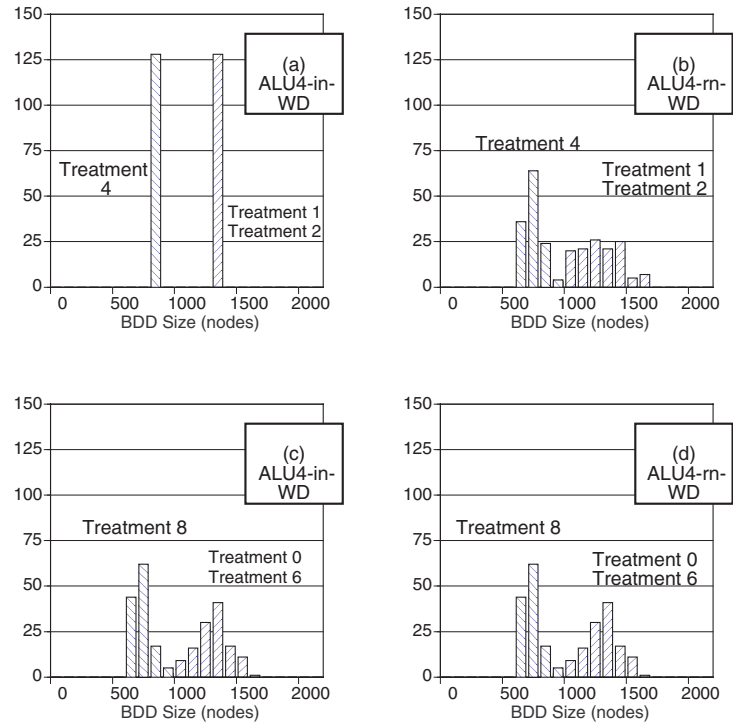


Fig. 2. BDD size (node) statistics under different variable ordering strategies, denoted as ‘treatments’. Some treatments of the ALU4r-in-WD class produce statistics that can be grossly misleading. Distributions of final sizes for 1,536 individual BDDs, constructed by VIS 1.1 from two classes of isomorphic and logically equivalent circuits, under treatments defined in Table 1. Panel **a** shows that, for the class ALU4r-in-WD (randomized order), all instances under treatments 1 and 2 are identical in size. Treatment 4 reduces the size of all BDDs to a smaller size, again identical for all instances in the class. A superficial analysis of these results would suggest that the static and dynamic ordering algorithms in VIS 1.1 are performing exactly as intended: the static algorithm is finding the same starting order for all the circuits, and the dynamic algorithm is uniformly reducing the initial size of the BDDs. Panel **b** illustrates an identical procedure, but using the isomorphism class ALU4r-rn-WD (randomized order *and* randomized labels). Here, the histograms show a near-Gaussian distribution of BDD sizes under treatments 1 and 2. Treatment 4 reduces the mean size of the BDDs, as well as the variance, but clearly the algorithms are not performing as expected. Since the only difference between the experiments in **a** and **b** is the labeling of the variables, we have established that the static ordering algorithm is sensitive to labeling. Panels **c** and **d** illustrate a similar experiment, differing only in the static ordering procedure. Here, the static ordering in VIS 1.1 is disabled, forcing it to use the random initial orderings in the class members. Note that the results are now identical for either class. We conclude that the VIS 1.1 static ordering algorithm, when confounded by random variable labels, produces results equivalent to random initial orderings

experiment. While a number of experiments with algorithms in [24] have been shown to have a total lack of sensitivity to relabeling, we have concluded that for BDD experiments, we must:

- **Lesson 1.** Make no assumptions about whether the algorithm under test is sensitive to relabeling.
- **Lesson 2.** Always create an equivalence class where node order is not only randomized, but the node labels are randomly reassigned for each instance in the class.
- **Lesson 3.** Recognize that many possible experiments *may be invalid*, unless the observations of lessons 1 and 2 have been applied at the outset of the experiment.

3 Limitations of traditional benchmarking

In the CAD community, the performance of algorithms is traditionally evaluated with respect to sets of “benchmark” circuits. The most widely-accepted of these benchmark sets were introduced at the International Symposium on Circuits and Systems (ISCAS) in 1985 and 1989,

and subsequently supplemented by other sets gathered and distributed at a series of workshops sponsored by the Association for Computing Machinery (ACM) and the Microelectronics Center of North Carolina (MCNC). Individual sets of circuits are often referenced in the literature by the forum at which they were introduced (e.g., “the ISCAS 85 benchmarks” [5]), or collectively referred to as “the ISCAS benchmarks”, “the MCNC benchmarks”, etc. A summary of well-known benchmarks in the CAD community was recently published in [26].

Most of the benchmark sets in common use consist of collections of unrelated circuits, represented as *netlists* (graph representations of the interconnection of logic gates implementing the circuit). It is common practice in the CAD literature to include a tabulation of the results of a particular algorithm on each member of one or more of these benchmark sets, and to compare those results to the work of others. In this section, we illustrate the contrast between this approach and our proposed DoE method.

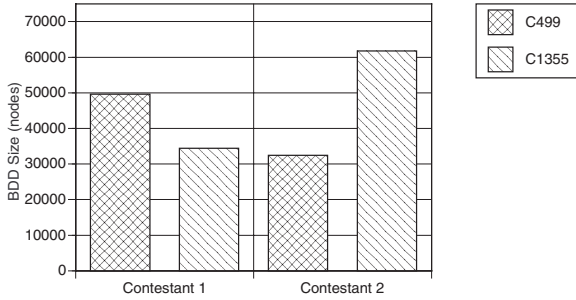


Fig. 3. Results of a hypothetical “contest” using single instances of typical benchmark circuits. Results of a hypothetical “contest” between two researchers. Each contestant is given two circuits; the object of the contest is to evaluate the final BDD size for each circuit, and report which circuit is “larger”. The contestants are *not* told that one circuit is a version of the C499 benchmark and the other is a version of C1355 [5], which are logically identical circuits. The contestants construct BDDs using the CMU BDD package [25] in VIS, and treatment 2. Contestant 1 reports that C1355 is smaller, while contestant 2 reports the C499 is the smaller circuit. Since both contestants used the same algorithm and the same treatment, on logically identical circuits, why are the results different?

3.1 A traditional benchmarking experiment

Figure 3 illustrates the results of a hypothetical “contest” between two researchers. This example was constructed using actual data from isomorphism classes of the benchmark circuits cited. For the “contest,” we chose specific instances from the isomorphism classes to illustrate a major deficiency in the traditional approach to benchmarking: *specific, isolated instances randomly selected from an isomorphism class cannot characterize the performance of an algorithm.* This example represents precisely the perils in the way in which benchmark circuits are typically defined and used, and how results are reported. We assert that many of the results and “improvements” reported in the literature may well be due to chance, and not due to the actual differences in the algorithms.

3.2 Analysis of correlations using DoE

We now discuss a more extensive characterization of an isomorphism class, to contrast DoE methods with traditional benchmarking methods. The C499 and C1355 circuits from the ISCAS 85 set [5], which are two very different logic implementations of the same function, are used to illustrate the shortcomings of traditional methods⁶.

1. Correlation between logically identical circuits.

The correlation plots in Fig. 4 present actual data for the isomorphism classes from which the samples in Fig. 3 were selected. We observe that there is little correlation between the BDD sizes in isomorphism classes of two logically identical circuits. For the contest, we

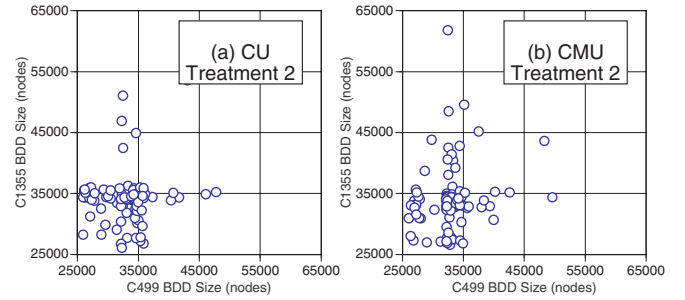


Fig. 4. Correlation between isomorphism classes of two logically identical circuits, under two BDD packages. Actual data for the isomorphism classes from which the samples in Fig. 3 were selected. Each point represents a randomly chosen pair of circuit instances, one from each class, representing the way in which benchmark circuits are actually selected in practice. The obvious lack of correlation between pairs of logically identical circuits illustrates a major deficiency in traditional benchmarking practice. Inspection of these plots also suggests that there may be differences in the performance of the two BDD packages. Such an observation may be formally tested for significance by comparing the distributions using the *t*-test, as described in Sect. 2.5

deliberately chose outliers from these distributions to illustrate the point. However, the actual circuits in all traditional benchmark sets are, in effect, chosen randomly from such a hypothetical population of representations. Without examining statistically meaningful samples of such populations, any reported results on the benchmark sets are subject to large, unknown random variability.

2. **Distributions for several treatments.** Figure 5 presents frequency distributions for the C499 and C1355 isomorphism classes under treatments 1, 2, and 4 for both the CU and CMU BDD packages. Tables 2, 3, and 4 illustrate the use of the *t*-test to determine the significance of observed differences in the distribution means. Further results of such tests have been published in [13, 27], and are explored further in Sect. 4 of this paper.

3. **Correlation analysis of several treatments.** Figure 6 illustrates the relationships between the distributions of Fig. 5 in more detail than a simple test of significance. The statistical tests tell us that, for instance, treatments 1 and 2 give significantly different results, but the correlation plot of Fig. 6a, for instance, gives the additional insight that, in many instances, the BDD size of a circuit actually *increases* under dynamic variable reordering.

4. **Correlation between BDD packages.** Figure 7 contrasts the behavior of one BDD package vs another. Statistical tests of the observations made in the figure are shown in Table 4. This example clearly shows the utility of statistical tests of significance: for the entire C499 isomorphism class, the observed “differences” between the CU and CMU BDD packages are not statistically significant. Clearly though, there are specific instances in the class which, if chosen by chance as benchmarks, would imply a very dif-

⁶ Complete descriptions and circuit diagrams for these circuits and others in the benchmark suites can be found in [18].

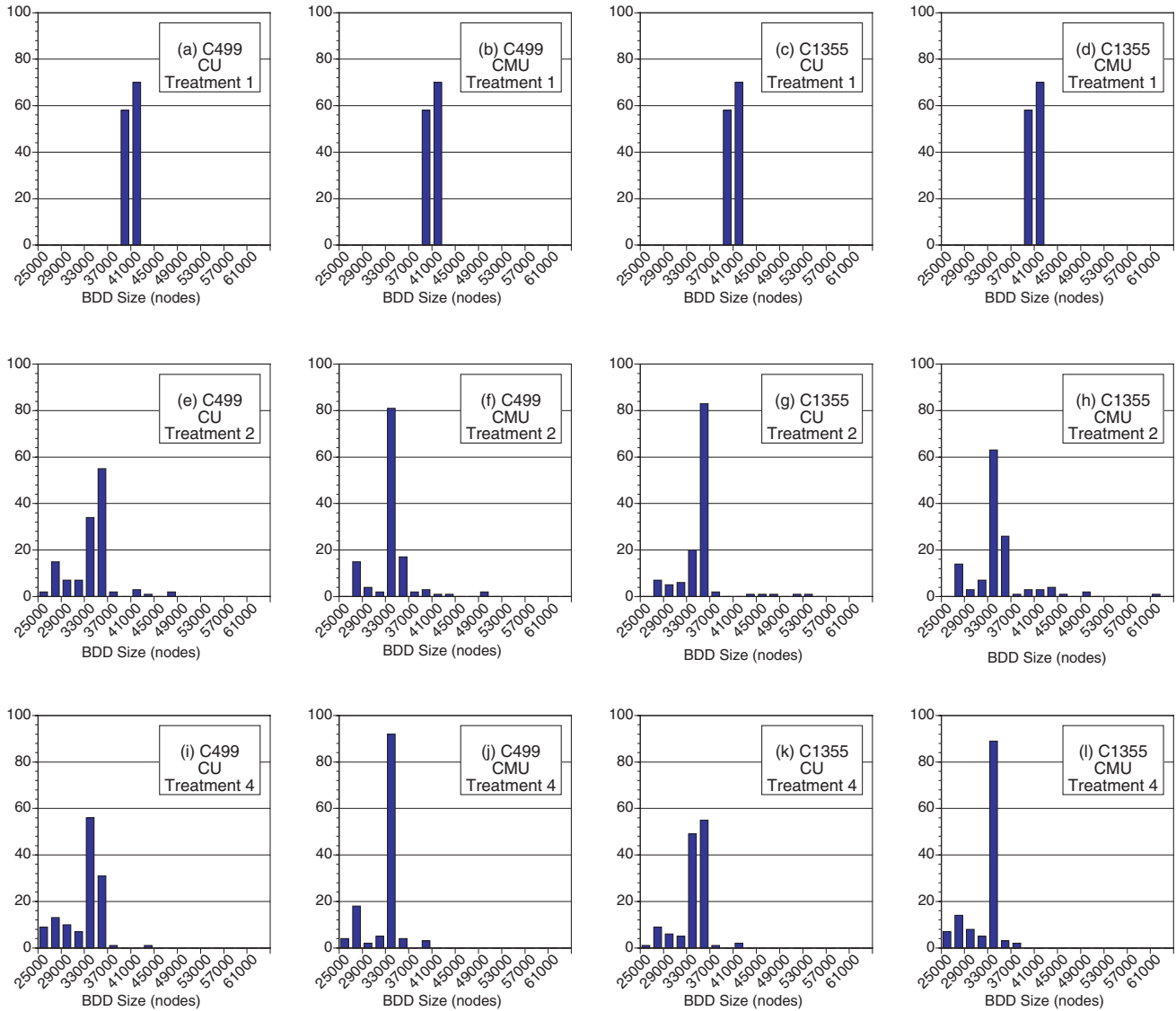


Fig. 5. Statistically significant report of a CAD experiment: BDD sizes for isomorphism classes of logically equivalent circuits under treatments 1, 2, and 4. Frequency distributions for the C499 and C1355 isomorphism classes under treatments 1, 2, and 4 for both the CU and CMU BDD packages. Tables 2, 3, and 4 illustrate the use of the t -test to test pairs of such distributions for significant differences in their means

ferent result. Finally, note that for the C1355 class, the difference between BDD packages is not significant for treatment 2, but significant for treatment 4; this suggests that the dynamic variable reordering algorithm in the CMU package is less effective than that for the CU package, in that it requires more invocations to achieve small BDDs than the CU package.

- Distinguishing software releases.** In [13] (and summarized in the set of experiments above), we demonstrated that the static ordering algorithm in VIS 1.1 was inoperative. It was found that the actual static order calculated by VIS for any circuit was simply the lexicographic order of the variable labels, due to a software error. This experiment demonstrated the importance of a properly randomized isomorphism class, as defined in Sect. 2.2.

Given these results, we evaluated the newer VIS 1.3 release using identical procedures, and compared the results. The difference in results for treatment 1 and 4 are summarized in Table 5; these results are explored in more depth in [27].

Data in the top row of Table 5 have been copied from Fig. 2 and represent results with VIS1.1. Data on the next row represent new results with VIS1.3. The t -test confirms that the distributions are significantly different for treatment 1, indicating that the static ordering algorithm in VIS 1.3 exhibits different behavior than in VIS 1.1. Surprisingly, not only are the differences in mean BDD size significant for treatment 4, the VIS1.1 result is better. We conclude that, for this example, the ‘better’ VIS 1.3 initial ordering heuristic has not contributed to better variable ordering with treatment 4 in VIS1.3.

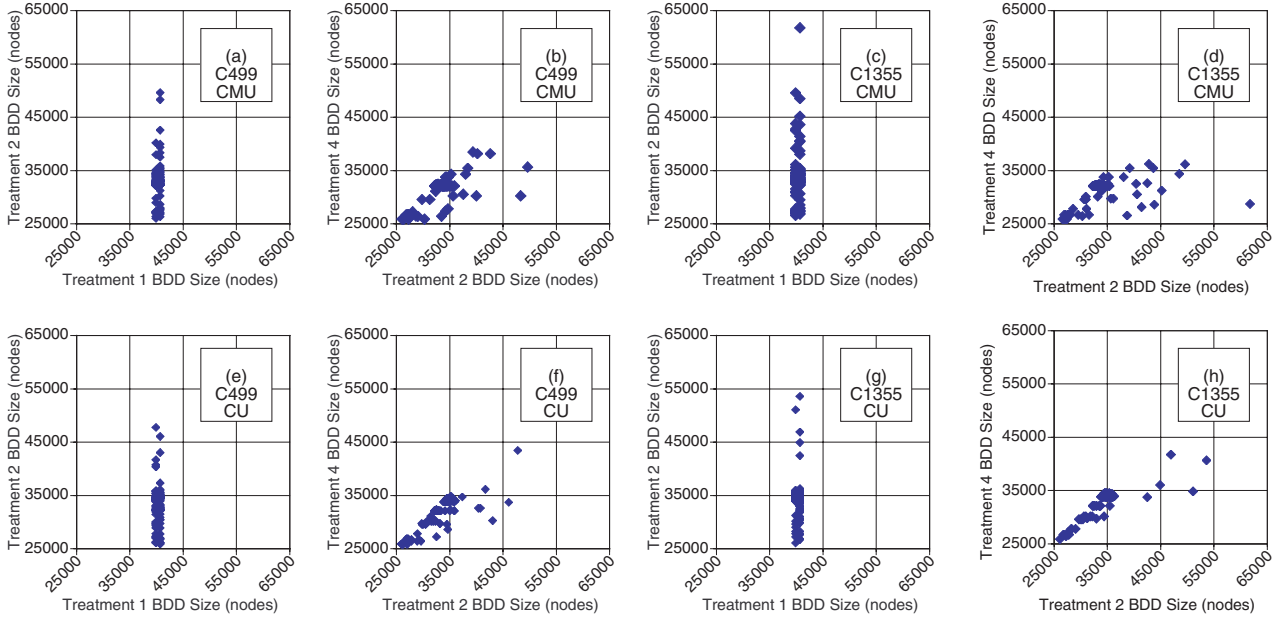


Fig. 6. Correlation among treatment results for isomorphism classes of logically identical circuits. Here we explore the relationships between the distributions of Fig. 5 in more detail than a simple test of significance. Panels **a** and **e** show that treatments 1 and 2 are uncorrelated for the C499 isomorphism class. Panels **c** and **g** illustrate that C1355 behaves similarly. Recalling the definitions of Table 1, it can be hypothesized that (in these examples) enabling dynamic reordering during BDD construction often reduces the final BDD size. However, in some cases this treatment actually appears to increase BDD size. While such an observation can be made visually from correlation plots, it is important to test it statistically for significance before reaching any conclusions about the relative merits of various treatments. Table 2 summarizes the t -test results between treatment 1 and 2. In this case, it is clear that the observed differences in sample means between Treatments 1 and 2 are significant

Table 2. Statistical comparison of treatments 1 and 2 under VIS 1.3 for C499 and C1355 isomorphism classes, using the CMU BDD package: panels **b** vs **f** and **d** vs **h** of Fig. 5. The large values of t indicate that the observed differences in means between treatments 1 and 2 are statistically significant

	C499		C1355	
	Mean	Std Dev	Mean	Std Dev
Treatment 1	40315	399.8	40315	399.8
Treatment 2	33066	3666	34321	3602
t -statistic	22.23		18.71	

Table 3. Statistical comparison of treatments 2 and 4 under VIS 1.3 for C499 and C1355 isomorphism classes, using the CMU BDD package: panels **f** vs **j** and **h** vs **l** of Fig. 5. The observed differences in means are still significant for both classes

	C499		C1355	
	Mean	Std Dev	Mean	Std Dev
Treatment 2	33066	3666	34321	3602
Treatment 4	31811	3038	33161	2422
t -statistic	2.98		3.02	

3.3 Traditional vs DoE: conclusions

The examples in the previous section are illustrations of the types of insights that can be gained through a design of experiments approach, and the profound differ-

Table 4. Statistical comparison of treatments 2 and 4 under VIS 1.3, for C499 and C1355 isomorphism classes under CU and CMU BDD packages. The values of t indicate that the observed differences between the means for C1355, between the CU and CMU packages, is statistically significant. However, the other differences in the table are not significant, and thus indicate no differences between the performance of the two BDD packages

	C499		C1355	
	Mean	Std Dev	Mean	Std Dev
Treatment 2 CU	33066	3666	34321	3602
Treatment 2 CMU	32843	3440	33805	4662
t -statistic	0.501		0.991	
Treatment 4 CU	31811	3038	33161	2422
Treatment 4 CMU	31362	2489	31179	2367
t -statistic	1.29		6.62	

ences between traditional benchmarking and the DoE approach. Here, the traditional approach would have given results for two instances (one each for C499 and C1355) under each treatment, or at most 20 experiments for the set of treatments defined in Table 1. As our experiments have illustrated, the results of the traditional approach are likely to be invalid because of the random choice of test cases. By contrast, the DoE approach represents 2,560 experiments (ten treatments of two classes of 128 instances each), giving the ability to statistically quantify results, including the variability of the results.

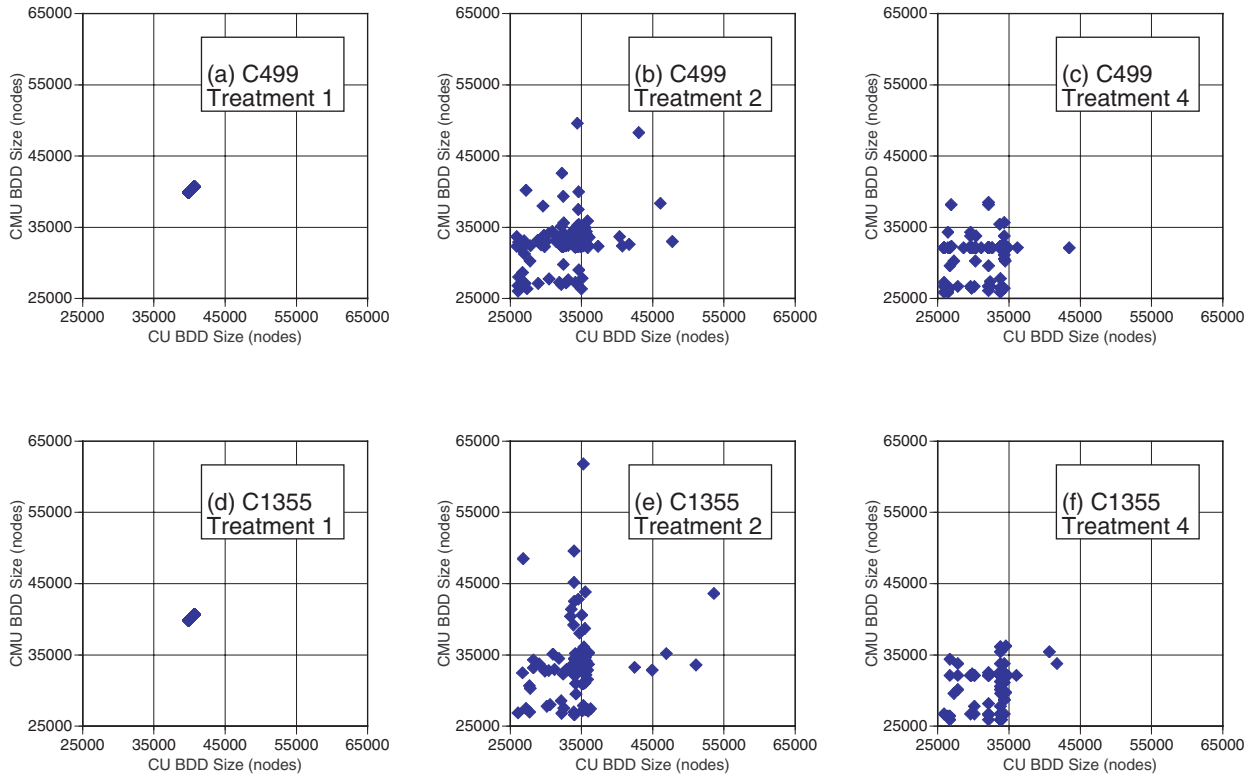


Fig. 7. Correlation between results for isomorphism classes of logically identical circuits, under two BDD packages. Here, the behavior of one BDD package vs another is illustrated. For either circuit isomorphism class, treatment 1 delivers equivalent results (panels a and d). Panels b and e illustrate that treatment 2 increases the variance of BDD size for the classes; furthermore, there appear to be marked differences in results for the two BDD packages. Panels c and f show the result of treatment 4, a common procedure used in BDD variable ordering applications. The difference between the two BDD packages is still apparent, but clearly the “stronger” treatment is reducing the final BDD size and variance in most cases. However, note that in some cases, there is still a large difference in final size between some class members, underscoring the importance of performing experiments with entire classes rather than with a single randomly selected instances from the class

Table 5. Statistical comparison of treatments 1 and 4 for `ALU4r_rn_WD` class under two software releases. The t -test confirms that the distributions are significantly different for treatment 1, indicating that the static ordering algorithm in VIS 1.3 exhibits different behavior than in VIS 1.1. Surprisingly, not only are the differences in mean BDD size significant for treatment 4, the VIS1.1 result is better. For this example, the ‘better’ VIS 1.3 initial ordering heuristic has not contributed to better variable ordering with treatment 4 in VIS1.3

	Treatment 1		Treatment 4	
	Mean	Std Dev	Mean	Std Dev
VIS 1.1	1268.1	159.4	637.4	65.7
VIS 1.3	870.4	59.4	659.0	65.2
t -statistic	26.3		-2.63	

4 Open problems and new directions

The experimental design strategy introduced in this paper demonstrated that equivalence classes of logically equivalent circuits are an essential component for consistent evaluation of BDD variable ordering algorithms. The same strategy can be applied to the evaluation of the equivalence classes themselves, leading to refinements of classes as well as algorithms.

The isomorphism equivalence class as defined in this paper demonstrated the sensitivity of all current heuristics to the initial variable order. Finding a minimized BDD size with variance of 0 for any initial order is a rare event, even for functions with number of variables as small as five. However, even if we do find solutions with small variance, questions to ask should include:

- (1) For the isomorphism equivalence class: *is the small BDD node variance due to the algorithm or due to some intrinsic properties of the function?*
- (2) For the class derived from but not equivalent to the reference circuit: *is the BDD node variance after variable optimization sufficiently small to declare the circuits as members of the same equivalence class?*

Answers to these questions are subject of ongoing research. The notion of the entropy-invariant perturbation is valid but the equivalence classes that are generated may be too broad [15]. Further refinements of the entropy-invariant class are necessary: additional parameters must be identified in order to constitute a well-defined signature for narrower classes of circuits [28]. Consider two single-output reference circuits with the same entropy and the same support variables, one representing the behavior of an ALU, the other a controller. The distribu-

tions of *minimized* BDD nodes sizes for the isomorphism classes of the ALU and the controller reference circuits are not expected to overlap. Circuits forming the equivalence classes may all be different, however, they should unambiguously exhibit the characteristic behavior of either ALU-like circuits or controller-like circuits. Just as for the two isomorphism classes, the distributions of the respective equivalence classes should not overlap. Moreover, the distribution of the ALU (controller) isomorphism class should be at the center of the distribution for its perturbation-based equivalence class. A similar problem has been addressed and effective solutions proposed in the context of graph-based cost functions that arise in physical design [14].

We briefly address some of these questions and before leaving the section, we will report on some surprising answers: small variance, including the variance of 0, is indeed an intrinsic property of some functions, including some that are well-known and important.

4.1 Asymptotic performance of multi-output circuits

Functions associated with multi-output circuits share the supporting variables. A variable order that optimizes BDD size for one function may conflict with an optimized order for another function. We need to study the performance of ordering algorithms over a range of related circuit classes, to gain insight into the scalability of the algorithms. Specifically, we constructed variable-output circuits from the C432 benchmark (a 7-output circuit) with 1, 2, . . . 6 primary outputs (all logic unique to the removed outputs was also removed). For each such derivative circuit, we built an isomorphism class of 128 instances, and again constructed BDDs using VIS 1.3 and a number of treatments. The results are shown in Table 6. The results for treatments 2 and 5 were essentially identical, and only treatment 2 is shown. These results indicate that the ordering heuristics of treatment 2, relative to treatment 1, is uniformly effective across all outputs – up to a point. Relative to the size of the mean, standard deviation for single output class is actually much worse than the standard deviation for multiple outputs where the mean value is much larger. The fact that in all

Table 6. Statistics of BDD sizes of C432 derivatives. Note that treatment 5 gave no improvement over treatment 2

Primary Outputs	Treatment 1		Treatment 2, 5	
	Mean	Std Dev	Mean	Std Dev
1	5735	71.7	488.2	40.8
2	11593	146.3	780.9	39.0
3	17635	225.3	1014.5	50.2
4	21337	233.7	1101.4	34.4
5	27983	323.4	1260.2	46.2
6	30564	387.4	1303.1	35.2
7	30620	396.6	1313.1	31.4

cases, an ideal algorithm would return solutions with variance of 0, leaves the experiment as a challenge for future work. New insights will emerge by studying each output independently of the other, and comparing the optimized variable orders for each function.

4.2 Asymptotic performance of single-output circuits

Characterizing the performance of BDD variable ordering algorithms in terms of single output circuits, with an increasing number of input variables, is essential to understanding both the properties of the underlying function and the limitations of the variable ordering algorithms. By analyzing classes of functions that continue to be related as we increase the number of variables, we expect to find asymptotic trends such as those shown in Fig. 8.

The trends shown in Fig. 8 summarize some of the data based on a series of ongoing experiments reported in [28]. Specifically, the data points represent summaries of BDD variable ordering experiments with isomorphism classes of five well-defined functions, with input variables ranging from 7 to 63. While each of the points in Fig. 8 corresponds to the minimum reported value of BDD node size, it is most instructive to observe also the respective min-max BDD range for each of the experiments with the isomorphism class of 32 function instances under different initial variable orderings.

The functions referenced in Fig. 8 are described below. The functions chosen for this experiment all have

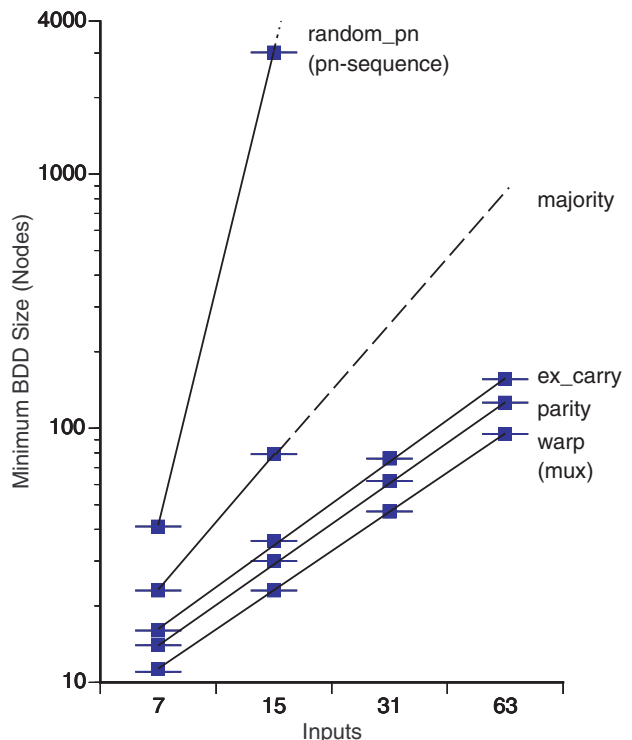


Fig. 8. Asymptotic views of some Boolean functions

identical entropy values of 1.0⁷. Here ‘randomOrders’ represents a randomized arrangement of variable orders, and ‘bestOrders’ represents best orders achievable by a heuristic when starting from a randomized arrangement. Ideally, min-max BDD range for ‘bestOrders’ should be 0, when we evaluate an isomorphism class. The fact that it is not, reflects the limitations of current heuristics, even for functions with only seven variables.

Function warp7: This function, implemented as a chain of 3-input multiplexors, represents a 7-input multiplexor. The min-max BDD ranges for the isomorphism class of this function are:

	min	max
randomOrders	11	31
bestOrders	11	15

Function parity7: This function, implemented as a cascade of 2-input XOR gates, represents a 7-input parity function. The min-max BDD ranges for the isomorphism class of this function are:

	min	max
randomOrders	14	14
bestOrders	14	14

Function ex_carry7: This function, implemented as a cascade of a 6-input carry circuit and a 2-input XOR function, represents a 7-input user-defined function. The min-max BDD ranges for the isomorphism class of this function are:

	min	max
randomOrders	17	30
bestOrders	16	18

Function majority7: This 7-input function is most expediently implemented directly as a truth table (all entries where number of ‘1’s in the truth table is the majority are defined to be ‘1’). The gate-level synthesis of this function is very time consuming as the number of variables increases. The min-max BDD ranges for the isomorphism class of this function are:

	min	max
randomOrders	23	23
bestOrders	23	23

Function random_pn7: This 7-input function implements a random function as a truth table with very well-defined randomness properties [28] and is based on the maximal-length pn-sequence (pseudo-noise sequence) [29]. The min-max BDD ranges for the isomorphism class of this function are:

	min	max
randomOrders	41	44
bestOrders	41	42

Furthermore, as shown in [28], a random function based on any pn-sequence will have a significant overlap with this range.

We conclude with a summary of most important observations we can make at this point in time about the properties of the functions shown in Fig. 8.

- Functions such as parity and majority are *symmetric functions*: no amount of variable order optimization will reduce the size of the underlying BDD; any random order is the best order, regardless of the number of variables. There is considerable literature on the properties of symmetric functions (e.g., [30]) and their detection as used in BDD software [31].
- In the entropy-invariant class, BDDs of functions that are most sensitive to variable order tend to optimize to smallest BDDs (see, for example, the warp function). On the other hand, BDDs of functions that are least sensitive to variable order tend to optimize to largest BDDs (see, for example, the random function).

Future work with single output functions should aim to fill in the gaps shown in Fig. 8, consider function classes with entropy < 1.0, and further expand on analysis.

5 Summary and conclusions

The experiments we have presented here are representative of the problems encountered in the domain of BDD variable ordering, but are by no means complete. For instance:

- We have exclusively measured and presented results for a single cost function, to illustrate the principles of DoE for BDD studies. A comprehensive characterization of the heuristics would include other cost functions, such as run time and peak BDD size.
- We used the VIS package as a convenient framework in which to study a variety of heuristics, but the methods are equally applicable to any software. VIS has many choices of static and dynamic ordering heuristics; here, we have evaluated only the default static ordering and the *sift* reordering heuristic.
- We have used only small example circuits here, although the principles apply equally well to larger functions. However, much can be learned by studying small circuits with known properties, as was demonstrated by our discovery of the problem with static ordering in VIS 1.1.
- The isomorphism classes used here consisted of 128 instances. (128 was chosen in anticipation of future studies of the effects of class size.) However, depending on the required confidence level for statistical tests, smaller classes are usually sufficient. Sampling theory provides precise methods for determining the proper class size for any experiment, but as a practical matter, the *t*-test is nearly equivalent to the *z*-test for more than 30 samples, so we recommend a class size of 32 for most future experiments.
- We have restricted the discussion here to isomorphism and entropy-signature invariant circuit equiva-

⁷ See Appendix A for additional information.

lence classes. However, many other classes are possible, and may have application for evaluating other heuristics. For instance, in [14] it was shown that wiring-signature invariant classes are most useful for physical design heuristics.

- We have dealt here exclusively with combinatorial functions. The considerations for BDD ordering in sequential circuit applications are different, and will require appropriate treatments, equivalence classes, cost functions, and experimental designs which will differ from those presented here.

In most fields of experimental science, careful experimental design is the norm. In software engineering and the closely related field of CAD, however, ad hoc methods of performance evaluation persist and are rarely challenged [32]. In this article, we have presented an overview of how the principles of experimental design, as practiced in agriculture, medicine, and many other fields, can be successfully applied to the evaluation of CAD algorithms, and in particular, to the evaluation of BDD variable ordering heuristics. These principles are generally applicable over a wide range of CAD tasks, but the particulars of the experiments will vary, depending on the problem domain.

We have briefly demonstrated experimental designs for BDD variable ordering heuristics, as follows:

- Performance of static ordering heuristics.
- Performance of dynamic reordering heuristics.
- Comparison of performance of two different BDD packages.
- Comparison of performance of two releases of the same BDD package.
- Asymptotic performance of ordering heuristics.

Acknowledgements. The assistance of the VIS team at the University of California at Berkeley and the University of Colorado at Boulder are gratefully acknowledged. Debabrata Ghosh provided software and advice in formulating and generating isomorphism classes. Nevin Kapur developed statistical summary software that was most useful.

References

1. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: Proc. Int. Conf. on Computer Aided Design, pp. 42–47, November 1993
2. Mercer, M.R., Kapur, R., Ross, D.E.: Functional approaches to generating orderings for efficient symbolic representations. In: Proc. 29th ACM/IEEE Design Automation Conference, pp. 624–627, June 1992
3. Panda, S., Somenzi, F., Plessier, B.F.: Symmetry detection and dynamic variable ordering of decision diagrams. In: Proc. Int. Conf. on Computer Aided Design, pp. 628–631, November 1994
4. Drechsler, R., Sieling, D.: Binary decision diagrams in theory and practice. In: Int J STTT 3(2): 112–136, 2001
5. Brglez, F., Fujiwara, H.: Special session on ATPG (Also introducing 'A Neutral Netlist of 10 Combinational Benchmark Circuits'). In: Int. Symp. on Circuits and Systems, 1985. Now a benchmark directory ISCAS85 at <http://www.cbl.ncsu.edu/benchmarks/>
6. Brglez, F., Bryan, D., Kozminski, K.: Combinational profiles of sequential benchmark circuits. In: IEEE 1989 Int. Symp. on Circuits and Systems – ISCAS89, pp. 1924–1934, May 1989. A basis for a benchmark directory ISCAS89, now archived at <http://www.cbl.ncsu.edu/benchmarks/>
7. Yang, S.: Logic synthesis and optimization benchmarks user guide. Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, N.C., USA, January 1991. This report is now available from <http://www.cbl.ncsu.edu/publications/#1991-IWLS-UG-Saeyang>, and benchmark directories archived at <http://www.cbl.ncsu.edu/benchmarks/>
8. Fisher, R.A.: Statistical methods, experimental design, and scientific inference. Oxford University, 1993. Reprinted, with corrections, from earlier versions, 1925–1973
9. Sentovich, E.M.: A brief study of BDD package performance. In: Proc. FMCAD'96. LNCS 1166. Berlin, Heidelberg, New York: Springer-Verlag, 1996, pp. 389–403
10. Ghosh, D., Kapur, N., Harlow III, J.E., Brglez, F.: Synthesis of wiring signature-invariant equivalence class circuit mutants and applications to benchmarking. In: Proc. Design Automation and Test in Europe, pp. 656–663, February 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-DATE-Ghosh>
11. Kapur, N., Ghosh, D., Brglez, F.: Towards a new benchmarking paradigm in EDA: analysis of equivalence class mutant circuit distributions. In: ACM Int. Symp. on Physical Design, April 1997
12. Brglez, F.: Design of experiments to evaluate CAD algorithms: which improvements are due to improved heuristic and which are merely due to chance? Technical Report 1998-TR@CBL-04-Brglez, CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695, USA, April 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-TR@CBL-04-Brglez>
13. Harlow, J.E., Brglez, F.: Design of experiments in BDD variable ordering: lessons learned. In: Proc. Int. Conf. on Computer Aided Design. ACM, November 1998. Also available from <http://www.cbl.ncsu.edu/publications/#1998-ICCAD-Harlow>
14. Ghosh, D.: Generation of tightly controlled equivalence classes for experimental design of heuristics for graph-based NP-hard problems. PhD thesis, Electrical and Computer Engineering, North Carolina State University, Raleigh, N.C., USA, May 2000. Also available at <http://www.cbl.ncsu.edu/publications/#2000-Thesis-PhD-Ghosh>
15. Harlow, J.E., Brglez, F.: Design of experiments for evaluation of bdd packages using controlled circuit mutations. In: Proc. Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD'98). LNCS 1522. Berlin, Heidelberg, New York: Springer-Verlag, 1998. Also available from <http://www.cbl.ncsu.edu/publications/#1998-FMCAD-Harlow>
16. The VIS Group.: VIS: A system for verification and synthesis. In: Alur, R., Henzinger, T. (eds): Proc. 8th Int. Conf. on Computer Aided Verification. LNCS 1102. Berlin, Heidelberg, New York: Springer-Verlag, 1996, pp. 428–432. Version 1.3 is available from UC Berkeley Design Technology Warehouse at <http://www-cad.eecs.berkeley.edu>
17. Somenzi, F., et al.: Colorado University Decision Diagram package (CUDD), release 2.3.0, 1998. Software available from <ftp://vlsi.colorado.edu/pub/cudd-2.3.0.tar.gz>
18. Hansen, M., Yalcin, H., Hayes, J.P.: Unveiling the iscas-85 benchmarks: a case study in reverse engineering. IEEE Design Test Comput 16(3): 72–80, 1999. Also see <http://www.eecs.umich.edu/~jhayes/iscas/benchmark.html>
19. Calhoun, J., Brglez, F.: A framework and method for hierarchical test generation. IEEE Trans Computer-Aided Design 11(1): 45–67, 1992
20. Box, G.E.P., Hunter, W.G., Hunter, J.S.: Statistics for experimenters: an introduction to design, data analysis, and model building. Wiley, New York, 1978
21. Brownlee, K.A.: Statistical theory and methodology in: science and engineering. Krieger, 1984. Reprinted, with revisions, from second edition, 1965
22. Hsu, J.C.: Multiple comparisons: theory and methods. Chapman and Hall, London, 1996
23. Shipley, T.R.: Private communication, 1998
24. Ghosh, D., Brglez, F.: Equivalence classes of circuit mutants for experimental design. In: Proc. Int. Symp. Circuits

- and Systems (ISCAS), May–June 1999. Also available at <http://www.cbl.ncsu.edu/publications/#1999-ISCAS-Ghosh>
25. Long, D.E.: CMU BDD package, 1993. Available from <http://emc.cmu.edu/pub/bdd/bddlib.tar.Z>
 26. Harlow, J.E.: Overview of popular benchmark sets. *IEEE Design Test Comput* July–September 17(3): 15–17, 2000
 27. Harlow, J.E., Brglez, F.: Mirror, mirror, on the wall ... is the new release any different at all? In: *Proc. Int. Symp. Circuits and Systems (ISCAS)*, May–June 1999. Also available at <http://www.cbl.ncsu.edu/publications/#1999-ISCAS-Harlow>
 28. Brglez, F., Harlow, J.: On classification of Boolean functions and experimental design. Technical Report 2000-TR@CBL-06-Brglez, CBL, CS Dept., NCSU, Box 8206, Raleigh, NC 27695, USA, November 2000. Available at <http://www.cbl.ncsu.edu/publications/#2000-TR-06-Brglez>
 29. Golomb, S.W.: Shift register sequences. Ageon Park, 1982
 30. Wegener, I.: Optimal decision trees and one-time-only branching programs for symmetric Boolean functions. *Inf Control* 62: 129–143, 1984
 31. Möller, D., Mohr, J., Weber, M.: Detection of symmetry of Boolean functions represented by ROBDDs. In: *Proc. Int. Conference on Computer-Aided Design*, pp. 680–684. ACM, November 1993
 32. Snelting, G.: Paul Feyerabend and software technology. *Software Tools Technol Transfer* 2: 1–5, 1998
 33. Shannon, C.E.: A mathematical theory of communication. *Bell Syst Tech J* 27: 379–423 and 623–656, 1948
 34. Helleman, L.: A measure of computational work. *IEEE Trans Comput* C-21: 439–446, 1972
 35. Cook, R.W., Flynn, M.J.: Logical network cost and entropy. *IEEE Trans Comput* C-22: 823–826, 1973
 36. Cheng, K.T., Agrawal, V.D.: An entropy measure for the complexity of multi-output Boolean functions. In: *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 302–305, 1990
 37. Shannon, C.E.: The synthesis of two-terminal switching circuits. *Bell Syst Tech J* 28: 59–98, 1949
 38. Slepian, D.: On the number of symmetry types of Boolean functions of n variables. *Can J Math* 5(2): 565–715, 1952
 39. Golomb, S.W.: On the classification of Boolean functions. *IRE Trans Inf Theor* IT-5: 176–186, 1959
 40. Harrison, M.A.: Counting theorems and their application to classification of switching functions. In: Mukhopadhyay, A. (ed): *Recent developments in switching theory*, pp. 86–121. Academic, New York, 1971
 41. Lechner, R.J.: Harmonic analysis of switching functions. In: Mukhopadhyay, A. (ed): *Recent developments in switching theory*, pp. 122–229. Academic, New York, 1971

Appendix A: Entropy and function classification

Entropy has been defined by Shannon [33] as a measure of “information, choice, and uncertainty” in the context of communication systems. This quantity plays a central role in his subsequent work on the information content of binary-encoded messages.

A definition of computational work based on entropy has been proposed by Helleman [34] and independently by Cook, and Flynn [35]. Cheng and Agrawal generalized the entropy formulation to multi-output functions [36]. They investigated the relationship between entropy and the average amount of logic to implement a combinational network through a series of experiments. These experiments confirm the results anticipated by Shannon [37], namely, that on the average, the amount of logic effectively doubles as one increases the number of input variables by 1.

For single output functions, the maximum value of entropy (1.0) expresses the perfect balance of function values that evaluate to 0 and function values that evaluate to 1. The class of functions with entropy of 1.0 plays a significant role in cryptography, and a number of important functions used in hardware design are in this class. The experiments in [36] confirm not only that the average amount of logic is maximum for the maximum entropy value of 1.0, but also that the variance of logic function implementations appears maximal. A comparison of three well-known functions, each with entropy of 1.0, illustrates this quickly: a degenerate n -input function such as \bar{x}_i requires no more than a single logic inverter; an n -input parity function requires exactly n 2-input logic nodes; an n -input majority function requires at least $O(n^2)$ 2-input logic nodes. In the average case, $O(2^n)$ 2-input logic nodes may be required.

When we classify the set of all 2^{2^n} Boolean functions by entropy, functions having the entropy of 1.0 form the largest class. Sampling the class of all functions having the entropy of 1.0 will most likely produce a class of functions that are too diverse to form a *single class* for use in the experimental design for testing the performance of algorithms. Hence, a number of additional invariants must characterize the function before we can assemble a class of sufficiently related functions for use in experimental design. New experimental research in this direction, complementing the theoretical work in [38–41], is in progress [28].

Appendix B: Statistical tests of significance

In the context of this paper, the basic statistical decision we want to make involves comparison of two treatments. The objective of each treatment is to reduce the size of the binary decision diagram (BDD), given an equivalence class of Boolean functions. Treatment 1 represents a variable ordering heuristic 1 inducing a population mean μ_1 , and treatment 2 represents a variable ordering heuristic 2 inducing a population mean μ_2 . *On the basis of sample data*, we have to decide between two hypotheses:

\mathcal{H}_0 : $\mu_1 = \mu_2$, and any difference is due to chance.

\mathcal{H}_1 : $\mu_1 < \mu_2$, and the difference is due to treatment.

We replicate some of the data from Table 5 to illustrate the decision process:

Calculation of t-statistic		
(sample size $N_1 = N_2 = N = 128$)		
	Sample	
	Mean	Std Dev
	(\bar{x}_1, \bar{x}_2)	(s_1, s_2)
Treatment 1	637.4	65.7
Treatment 2	659.0	65.2
t -statistic		-2.63

In our experiments, the number of samples for a given class is held constant, i.e., $N_1 = N_2 = N$, hence the formula to calculate the t -statistic is particularly simple [20, 21]:

$$t = \frac{\overline{x_1} - \overline{x_2}}{\sqrt{((s_1^2 + s_2^2)/(N - 1))}}$$

Since we are testing the hypothesis that one treatment is better than another, we define the *critical t-value* in the context of *one-sided test* or *one-tailed test*. In such cases, the critical region is a region to one side of t -distribution,

⁸ Testing the hypothesis that one treatment is better than another is different from testing whether one treatment is better or worse than the other. Testing the latter hypothesis requires a *two-sided test* or *two-tailed test*.

with area under the distribution equal to the level of significance α ⁸

A typical level of significance in hypothesis testing is $\alpha = 0.05$, implying a critical value of $t_{\alpha, 2(N-1)}$. For example, we find $t_{0.05, 254} = 1.969$ for two sample sets, each of size 128, or $t_{0.05, 62} = 1.999$ for two sample sets, each of size of 32. At this level of significance, we accept the hypothesis \mathcal{H}_0 only if t -statistic based on 254 degrees of freedom evaluates to $|t| < t_{0.05, 254}$. Our level of confidence, or the probability that the acceptance of \mathcal{H}_0 is actually correct, is 95%.

In the table above, while the sample means evaluate to $\overline{x_1} < \overline{x_2}$, we cannot accept this result as a certainty. However, since we evaluated $|t| = 2.63 > t_{0.05, 254} = 1.969$, we now reject the hypothesis \mathcal{H}_0 that the respective means are the same, and accept that the hypothesis \mathcal{H}_1 ($\overline{x_1} < \overline{x_2}$) is true – with the probability of 95%.