Simulation-based functional test justification using a Boolean data miner

Charles H.-P. Wen, Onur Guzey and Li-C. Wang
Dept. of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
Email: {opwen,oguzey,licwang}@ece.ucsb.edu

Jin Yang
Strategic CAD Lab,
Intel Corporation
Email: jin.yang@intel.com

***Abstract*— In simulation-based functional verification, composing and debugging testbenches can be tedious and time-consuming. A simulation data-mining approach, called *TTPG*[3], was proposed as an alternative for functional test pattern generation. However, the core of simulation data-mining approach is *Boolean learning*, which tries to extract the simplified view of the design functionality according to the given bit-level simulation data. In this work[1], an efficient data-mining engine is presented based on decision-diagram(DD)-based learning approaches. We compare the DD-based learning approaches to other known methods, such as the Nearest Neighbor method and Support Vector Machine. We demonstrate that the proposed Boolean data miner is efficient for practical use. Finally, that the *TTPG* methodology incorporated with the Boolean data miner can achieve a high fault coverage (95.36%) on the OpenRISC 1200 microprocessor concludes the effectiveness of the proposed approach.**

I. Introduction

In practice, functional verification of large and complex designs relies on extensive testbench simulation. Testbench development can be tedious and time-consuming. To alleviate this burden, *constrained random verification* paradigm extended from the Random Test Program Generation (RTPG) [1] methodology has become a popular approach. In constrained random verification, designers develop *test templates* to replace specific tests. A template consists of manually imposed input *constraints* and *biases*, which can be instantiated into many tests.

To guide test templates towards given verification targets, the work in [3] suggests a Target Test Pattern Generation (TTPG) methodology where (sequential) ATPG is first applied locally on the design block containing the target, to produce test patterns at the boundary of the design block. Then, further justification of these local patterns is guided by the so-called *learned models*.

Simulation data can be collected based on instantiated tests from *test templates* and is minded to develop learned models. The learned models help to *guide* the modification of the test templates by assigning more specific values to certain inputs so that future instantiated tests could have a higher possibility to cover the verification target. This methodology is useful when the pattern justification must conform to local logic constraints hard to satisfy manually in designer-composed test templates. However, the effectiveness of the methodology depends on the effectiveness of the learned models.

[1]This project is sponsored by SRC under Task ID 1360.001

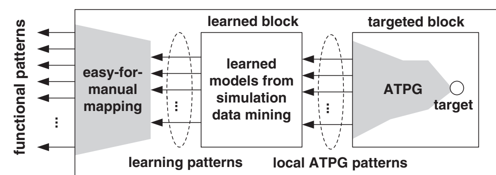
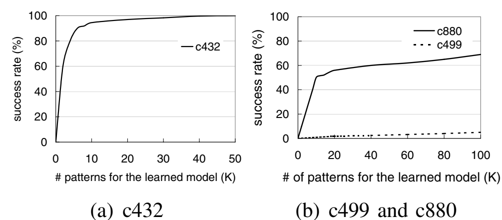


Fig. 1. Combining local ATPG and simulation data learning to guide functional test pattern justification – The motivating application

Figure 1 illustrates the TTPG methodology on a sequential design. Note that on one hand the targeted design block can be complex and far away from the chip boundary as long as ATPG process can finish efficiently. On the other hand, the learned block is prior logic to the targeted block, and starts from the boundary where designers can easily map the learning patterns back to the functional patterns.

When learning bit-level data, it becomes a *Boolean learning* problem. The objective of Boolean learning is to develop a learned model which provides a *simplified view* of the functionality of the learned block and allow us to find a potential *short-cut* to translate the local ATPG patterns to the inputs of the learned block. In addition, it is required that the inverse of the learned model can be computed efficiently so that the learning patterns can be generated easily.



(a) c432 (b) c499 and c880

Fig. 2. Examples of Boolean learnability

For better understanding how difficult the Boolean learning problem is, a learning algorithm based on logic optimization was first proposed to be applied on Boolean circuits in [3]. Authors use three ISCAS85 example circuits, c432, c499 and c880, to illustrate different levels of difficulty on Boolean learnability in Figure 2. X-axis represents the number of used

1-4244-9707-X/06/$20.00 ©2006 IEEE

patterns in the learning models while Y-axis represents the learning accuracy based on another set of randomly-generated patterns. Figure 2(a) shows that c432 is easy to learn while Figure 2(b) shows that learning c499 is not effective and the effectiveness of learning c880 lies in between. Note that how effective the Boolean learning is will decide the success of the TTPG methodology.

From the literature in Computational Learning Theory [4], Boolean learning algorithms can be classified into two categories: *non-query(NQ)-based mining* and *query(Q)-based learning*. In query-based learning, a learning algorithm is allowed to generate any input pattern $\vec{x}$ and obtain the value $f(\vec{x})$ by querying a simulator on the target Boolean function. In NQ-based mining, a set of n *training samples* $D = ((\vec{x_1}, f(\vec{x_1})), \ldots, (\vec{x_n}, f(\vec{x_n})))$ are given in advance. A learning algorithm relies solely on the information provided in D to derive a learned model. No queries are allowed during the learning.

Practically speaking, query-based learning requires a learning algorithm to have full control of the simulator, which may limit its applicability in certain situations. However, non-query data mining has no such constraint [3]. Thus, we are more interested in non-query based mining than query-based learning. In this paper, we describe an efficient non-query based data mining engine for the Boolean domain. This engine is built on a novel decision-diagram (DD) based learning approach. We abandoned the use of existing learning approaches such as Fourier based analysis method [12] or SVM [15] for building the learning engine because those popular learning methods usually do not support the efficient computation of the inverse learned models, which is a crucial requirement for our engine to be used in practice.

Our DD-based learning approach stores the learned model of an output function in terms of a Binary Decision Diagram that can be easily converted into an Ordered Binary Decision Diagrams (OBDDs) [5]. This feature greatly simplifies the computation of inverse learned model. To evaluate the effectiveness of our learning engine, we compare the performance of our NQ-based mining algorithm to those of two existing methods: the simplest method implementing the *Nearest Neighbor* (NN) principle and the most advanced method employing the *Support Vector Machine* (SVM) [16][15]. We show that our engine is much more efficient than NN and SVM, and is comparable in accuracy to SVM.

The rest of the paper is organized as: Section II discusses the background and formulates the learning problem in this work. Section III proposes and analyzes the NQ based mining algorithms. Section IV summarizes the experimental results to show the effectiveness of our Boolean data miner. We also apply the TTPG methodology incorporated with our Boolean data miner to a microprocessor design. Section V concludes the paper.

II. Background

The problem of learning Boolean functions has been studied extensively since the pioneer work of L. Valiant [6]. Valiant defined a learning model called *Probably Approximately Correct* (PAC) model. Given a target function $f(x)$ on n inputs, i.e. $||x|| = n$, and two small numbers, ϵ and δ, f is PAC learnable if there exists an algorithm that runs in polynomial time in terms of $\frac{1}{\epsilon}, \frac{1}{\delta}$ and n, and with a probability at least $1 - \delta$ produces a learned model $LM(x)$ such that LM can approximate f with an error probability at most ϵ, where ϵ is arbitrarily small, i.e. $\epsilon \rightarrow 0$. The number, δ, is used for quantifying the success rate of learning algorithms because learning algorithms are *randomized algorithms* that rely on random samples to get their answers.

Most algorithms in previous studies [6]-[13] were for facilitating the proofs of some theoretical results on the complexity of the problem. Little effort has been devoted to investigating the possibility of developing an efficient Boolean learning engine for practical use in EDA applications. Therefore, in this work we pursue such an investigation.

Our problem formulation:

The Boolean learning problem in our work is different from the theoretical studies mentioned above. From one perspective, our problem is harder than the classical Boolean learning problem formulation because (1) for a learned model LM, we require that LM^{-1} can be computed efficiently , and (2) given an output pattern z, we demand accurate learning on $||z|| = m$ outputs so that pattern justification can be effectively guided by $LM^{-1}(z)$. Note that a learning algorithm usually handle one output at a time. Hence, pattern justification requires learned results on individual outputs to be combined together.

From another perspective, our Boolean learning problem could be easier due to two reasons: (1) in practice, the error ϵ of learning can be relaxed to a small percentage number, like 2%, instead of an arbitrarily small number $\epsilon \rightarrow 0$. Notice that this means that our learning algorithm is still allowed to make mistakes on *an exponential number of inputs* because $0.02 * 2^n$ is still on the order of $O(2^n)$. (2) for pattern justification, we are allowed to generate k input vectors $x_1, \ldots, x_k$ from $LM^{-1}(z)$ (typically $LM^{-1}(z)$ is not unique). This is because we can simulate $f(x_1), \ldots, f(x_k)$ and then verify if any x_i actually satisfies $f(x_i) = z$. If one of them does, then we have successfully found a justified pattern. Hence, by increasing k, we can reduce the need for high learning accuracy.

III. Non-query based data mining

NQ Boolean learning can be transformed into a *binary classification* problem [14]. For binary classification, SVM [16][15] is one of the most popular approaches in recent years. It is a supervised learning technique which non-linearly maps the input space onto a high dimensional feature space and then finds the separation function for the classification purpose. Due to the space constraints, we omit the detailed discussion of the SVM algorithm. Interesting readers can refer to [16][15]. If we treat SVM as one of the most advanced learning approaches, then the most naive and straightforward approach may be the implementation of the *Nearest Neighbor (NN)* principle.

Suppose that there are four input variables $x_1 x_2 x_3 x_4$. Suppose that the simulation gives a data set based on five

	$\vec{v}_1$	$\vec{v}_2$	$\vec{v}_3$	$\vec{v}_4$	$\vec{v}_5$
x_1	1	0	1	0	1
x_2	0	1	1	1	0
x_3	1	1	0	0	1
x_4	1	0	1	1	0
$f()$	0	0	1	1	0

(a) original data

	$\vec{v}_5$	$\vec{v}_2$	$\vec{v}_4$	$\vec{v}_3$	$\vec{v}_1$
$\hat{x}_1(x_4)$	0	0	1	1	1
$\hat{x}_2(x_3)$	1	1	0	0	1
$\hat{x}_3(x_2)$	0	1	1	1	0
$\hat{x}_4(x_1)$	1	0	0	1	1
$f()$	0	0	1	1	0

(b) sorted by ordering

<table>
<tr><th></th><th>$\vec{v}_4$</th><th>$\vec{v}_3$</th><th>$\vec{v}_5$ $\vec{v}_1$</th><th>$\vec{v}_2$</th></tr>
<tr><td>$\hat{x}_1$</td><td colspan="4">X</td></tr>
<tr><td>$\hat{x}_2$</td><td colspan="2">0</td><td colspan="2">1</td></tr>
<tr><td>$\hat{x}_3$</td><td colspan="2">1</td><td>0</td><td>1</td></tr>
<tr><td>$\hat{x}_4$</td><td>0</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>$f()$</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
</table>

(c) regularized model

TABLE I

AN EXAMPLE OF ORDERED NEAREST NEIGHBOR

input vectors $f(\vec{v}_1 = 1011) = 0$, $f(\vec{v}_2 = 0110) = 0$, $f(\vec{v}_3 = 1101) = 1$, $f(\vec{v}_4 = 0101) = 1$, and $f(\vec{v}_5 = 1010) = 0$. Suppose we are given two input vectors, $\vec{v}_6 = 1100$ and $\vec{v}_7 = 1111$, which do not appear in the simulation data.

Nearest neighbor (NN): Given an input x, the NN algorithm finds a v_i in the simulation data set, whose (Hamming) distance to x is the smallest and uses the value $f(v_i)$ as the answer to $f(x)$. Taking 1100 as example, this vector differs from v_3 only by the last bit at x_4 position and differs from others by at least two positions. Since $f(v_3) = 1$, the NN algorithm reports $f(1100) = 1$. For input 1111, this vector differs from v_1 and v_3 by one bit. Because $f(v_1) = 0$ and $f(v_3) = 1$, the algorithm may choose either one of them, or may take the average of values from all the nearest neighbors.

A. Ordered nearest neighbor (ONN)

Given an ordering, Π, of n inputs and the simulation data, D, we store the data into a matrix form, M, according to Π. Then, we re-arrange the columns by sorting the input vectors where each column vector is viewed as a binary word and the top/bottom input variable is the most/least significant bit.

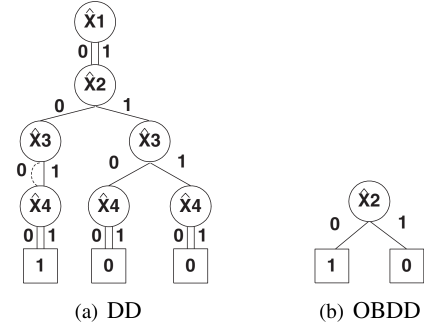
Suppose Π gives the ordering of inputs as $\hat{x}_1\hat{x}_2\cdots\hat{x}_n$. Given two bit vectors, $(p_1,\ldots,p_n),(q_1,\ldots,q_n)$, conceptually at each level k, we utilizes a *weighted distance function*, $\omega(k) = \Sigma_{i=1}^{n}(p_i \oplus q_i)2^{n-i}$ to determine the nearest neighbor. The weighted scheme ensures that $(p_k \oplus q_k)$ weights more than $(p_i \oplus q_i)|_{i=(k+1)..n}$.

We note that applying weighted nearest neighbor with $\omega(k)$ may expand our learning model into the complete binary decision tree in the worst case. From the statistical learning [16] point of view, this may be seen as the *over-fitting* problem, which briefly states that the complexity of the learned model is too high to allow induction to be made on the data. The result is that, the learned model, although fits the training data well, has a poor generalization accuracy, i.e. perform poorly on another data set during validation. To avoid the over-fitting problem, the answer from statistical learning theory is *regularization* that controls the capacity of a learning machine. Therefore, one may see ONN as a result of regularized NN.

Let's illustrate ONN onto the simulation data D shown as Table I(a) in the beginning of this section. Suppose Π gives the order of $\langle\hat{x}_1,\hat{x}_2,\hat{x}_3,\hat{x}_4\rangle = \langle x_4,x_3,x_2,x_1\rangle$. We sort the simulation data D into the matrix form M shown in Table I(b). At the 1st level, $\vec{v}_5$ under $\hat{x}_1 = 0$ and $\vec{v}_1$ under $\hat{x}_1 = 1$ share a common sub-vector ($\hat{x}_2\hat{x}_3\hat{x}_4 = 101$) and result in the same answer. We say that such pair of sub-vectors *witnesses* the redundancy of $\hat{x}_1$, and thus $\hat{x}_1$ can be removed and marked as X in Table I(c). From the 2nd level on, we keep sorting the matrix when fixing one input variable $\hat{x}_i$, and then splitting nodes into 0-branch and 1-branch. Since no other pair of *witnessing* sub-vectors can be found, the final regularized model is constructed as shown in Table I(c).

Now given $\vec{v}_6 = 1100$ ($\hat{x}_1\hat{x}_2\hat{x}_3\hat{x}_4 = 0011$). Since $\hat{x}_1$ is marked as X, we skip the comparison at this level. The next step is to match $\hat{x}_2 = 0$, and this gives the subset $\{\vec{v}_4,\vec{v}_3\}$ as its nearest neighbors. Similarly, $\{\vec{v}_4\}$ stays when matching $\hat{x}_3 = 1$ and $\hat{x}_4 = 1$. Therefore, $\vec{v}_6$ takes $\vec{v}_4$ as its nearest neighbor and output the answer $f(\vec{v}_6)$ as 1. By the same manner, for $\vec{v}_7 = 1111$, we can identify $\vec{v}_2$ as the nearest neighbor and give the answer $f(\vec{v}_7) = 0$. Note that, in NN, we may find multiple nearest neighbors which have conflicting output values for a given vector. However, the ONN algorithm always finds an unique answer from nearest neighbor set.

B. Implementing ONN with decision diagrams



(a) DD (b) OBDD

Fig. 3. Example of DD and OBDD views in ONN

It is natural to use a (binary) decision diagram to implement the ONN algorithm just described. For example, Figure 3(a) shows the decision diagram representation of the learning model in Table I(c). The dotted lines denote the missing branches not seen in the matrix. If we remove each node whose 0-branch and 1-branch point to the same child, the result is a *reduced* decision diagram shown in Figure 3(b).

Let T_λ be the sub-tree rooted at the node given by following the path λ in the decision diagram. For example, T_{10} in the example decision diagram leads to the sub-tree rooted at the left x_3 node. Similarly, we define M_λ to be the sub-matrix that consists of all columns whose first $||\lambda||$ inputs match the prefix λ. For example $M_{10} = \{\vec{v_4},\vec{v_3}\}$ in the example. Then, T_λ is the decision diagram representation of M_λ.

In ONN, merging $T_{\lambda 0}$ and $T_{\lambda 1}$ is based on two rules: (1) $(M_{\lambda 0} = \phi) \cup (M_{\lambda 1} = \phi)$ is true, or (2) $\nexists$ any sub-vector $s_1 \in M_{\lambda 0} \cap M_{\lambda 1}$ where $f(\lambda 0 s_1) \neq f(\lambda 1 s_1)$ and $\exists$ one sub-vector $s_2 \in M_{\lambda 0} \cap M_{\lambda 1}$ where $f(\lambda 0 s_2) = f(\lambda 1 s_2)$. We call this *ONN compatibility check*. In other words, when implementing ONN with a decision diagram, we can check, for each prefix λ if $M_{\lambda 0}$ and $M_{\lambda 1}$ are *ONN-compatible*, and decide to merge $T_{\lambda 0}$ and $T_{\lambda 1}$ or not. The algorithm is illustrated below.

Algorithm 1 describes the conversion of a data matrix into a decision diagram. The resulting decision diagram has not been

Algorithm 1 ONN learning algorithm: ONN(M_λ)

1: **if** $M_\lambda \neq constant$ **then**
2: $\quad x \leftarrow$ the 1st variable in M_λ;
3: $\quad$ create node T_x;
4: $\quad$ **if** $compatible(M_{\lambda 0}, M_{\lambda 1})$ **then**
5: $\quad\quad M_{\lambda x} = merge(M_{\lambda 0}, M_{\lambda 1})$;
6: $\quad\quad$ return ONN($M_{\lambda x}$);
7: $\quad$ **else**
8: $\quad\quad T_x.lefttree \leftarrow$ ONN($M_{\lambda 0}$);
9: $\quad\quad T_x.righttree \leftarrow$ ONN($M_{\lambda 1}$);
10: $\quad$ **end if**
11: $\quad$ **return** T_x;
12: **else**
13: $\quad$ **return** $constant(M_\lambda)$;
14: **end if**

	$\vec{u}_1$	$\vec{u}_2$
$\hat{x}_3$	0	1
$\hat{x}_4$	0	1
$\hat{x}_5$	1	1
$f()$	1	1

(a) $M_{\Lambda_1=\{00\}}$

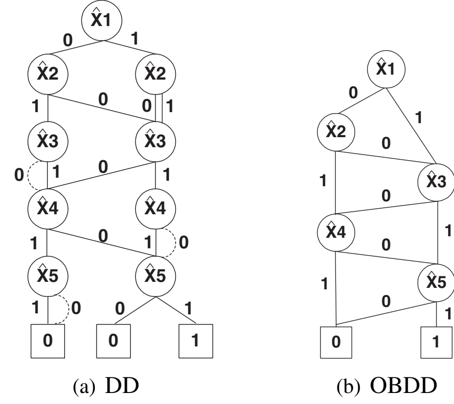
	$\vec{u}_5$	$\vec{u}_4$	$\vec{u}_6$
$\hat{x}_3$	0	0	1
$\hat{x}_4$	0	1	1
$\hat{x}_5$	0	1	1
$f()$	0	0	1

(b) $M_{\Lambda_3=\{10,11\}}$

	$\vec{u}_5$	$\vec{u}_1$	$\vec{u}_4$	$\vec{u}_2$	$\vec{u}_6$
$\hat{x}_3$	0	0	0	1	1
$\hat{x}_4$	0	0	1	1	1
$\hat{x}_5$	0	1	1	1	1
$f()$	0	1	0	1	1

(c) $M_{\Lambda=\{00,10,11\}}$

TABLE II
AN EXAMPLE OF MERGING NODES IN OIR



(a) DD (b) OBDD

Fig. 4. Example of DD and OBDD views in OIR

an OBDD yet. However, converting such a decision diagram into an OBDD can be easily done with an existing OBDD package [17]. Note that among the compatibility checks in ONN, nodes merged due to rule (1) occur more frequently at upper levels while nodes merged due to rule (2) only happen at lower levels since it is more likely to see the common sub-vector at lower levels.

C. Ordered input removal (OIR)

In Algorithm 1, compatibility check only applies between two sibling nodes, $M_{\lambda 0}$ and $M_{\lambda 1}$, and requires at least one common sub-vector to merge $M_{\lambda 0}$ and $M_{\lambda 1}$. We can further *relax* the rule of the compatibility check across sibling nodes. For example, we can say that two sub-matrices $M_{\Lambda'}$ and $M_{\Lambda''}$ are *compatible* if $\nexists$ any sub-vector $s \in M_{\Lambda'} \cap M_{\Lambda''}$, where $f(\Lambda' s) \neq f(\Lambda'' s)$. Note that Λ' and Λ'' in the OIR compatibility check represent the set of all prefix paths $\lambda' s$ from the root. This rule is also valid on the case that even the two sub-matrices do not share any input sub-vector. We call this *OIR compatibility check*.

For example, assume that the inputs consist of five variables ordered by $< \hat{x_1}, \hat{x_2}, \hat{x_3}, \hat{x_4}, \hat{x_5} >=< x_1, x_2, x_3, x_4, x_5 >$. Simulation data contains six input vectors and their results, $f(\vec{u}_1 = 00001) = 1$, $f(\vec{u}_2 = 00111) = 1$, $f(\vec{u}_3 = 01111) = 0$, $f(\vec{u}_4 = 10011) = 0$, $f(\vec{u}_5 = 11000) = 0$, and $f(\vec{u}_6 = 11111) = 1$. At the 2nd level $\hat{x_2}$, the compatibility check merges the sibling nodes, and we first obtain three nodes, $M_{\Lambda_1=\{00\}}$, $M_{\Lambda_2=\{01\}}$ and $M_{\Lambda_3=\{10,11\}}$.

When checking across sibling nodes, we can further observe that M_{Λ_1} is OIR-compatible to M_{Λ_3} because there does not exist one sub-vector conflicting under M_{Λ_1} and M_{Λ_3}. Therefore, we can merge M_{Λ_1} and M_{Λ_3} into M_Λ shown as the Table II.

Figure 4(a) illustrates the decision diagram after applying OIR onto the simulation matrix. Figure 4(b) is the final OBDD after removing the redundancy. We can observe that the across-sibling compatibility check in OIR is very aggressive and tries to minimize the width of decision diagram while, in contrast, ONN is relatively conservative to check compatibility on sibling nodes only.

D. Analysis of ONN and OIR

The *compatibility check* determines the behavior of NQ-based mining algorithms, and further plays the important role to *regularize* the complexity of the learning model. We summarize below some interesting properties of the ONN and OIR compatibility checks.

Property 3.1: Let n be the number of input variables. Let d be the number of simulation vectors in the data matrix M (Typically, $d >> n$). Then, in the resulting decision diagram given by ONN and OIR, the number of nodes at every level should be $\leq d$. As a result, the upper bound of total number of nodes is $2^{\lfloor \log_2 d \rfloor + 1} + d \times (n - \lfloor \log_2 d \rfloor) - 1$.

Proof: For both ONN and OIR, in the worst case, we can have the complete binary tree from the 1st level to the $\lfloor \log_2 d \rfloor^{th}$ level. Therefore, the number of nodes from the 1st level to the $\lfloor \log_2 d \rfloor^{th}$ level is $N_u = 1 + 2 + \cdots + 2^{\lfloor \log_2 d \rfloor} = 2^{\lfloor \log_2 d \rfloor + 1} - 1$. From the $(\lfloor \log_2 d \rfloor + 1)^{th}$ level on, at most d nodes corresponding to the d output values can be generated at one level. Therefore, the number of nodes from the $(\lfloor \log_2 d \rfloor + 1)^{th}$ level to the nth level is $N_l = d \times (n - \lfloor \log_2 d \rfloor)$. To sum up, the maximum number of nodes for ONN and OIR is $N = N_u + N_l = 2^{\lfloor \log_2 d \rfloor + 1} + d \times (n - \lfloor \log_2 d \rfloor) - 1$ □

Property 3.2: Let DD be the decision diagram resulting from applying ONN or OIR on data matrix M. For any input vector $\vec{v} \in M$, the evaluation of $f(\vec{v})$ on DD always gives the correct answer.

Proof: This is straightforward. □

Property 3.3: If $M_{\alpha 0}$ and $M_{\alpha 1}$ are ONN-compatible, then $M_{\alpha 0}$ and $M_{\alpha 1}$ are also OIR-compatible.

Proof: (Case 1) if $M_{\alpha 0}$ and $M_{\alpha 1}$ are ONN-compatible because of rule (1) of the ONN compatibility check in Section 3.2, then either $M_{\alpha 0}$ or $M_{\alpha 1}$ is empty, and $M_{\alpha 0}$ and $M_{\alpha 1}$ cannot

share any sub-vector. Hence, $M_{\alpha 0}$ and $M_{\alpha 1}$ are also OIR-compatible. (Case 2) if it is due to rule (2) in ONN, then the condition that $\nexists s_1 \in M_{\lambda 0} \cap M_{\lambda 1}$ where $f(\lambda 0 s_1) \neq f(\lambda 1 s_1)$ must hold from ONN. Let $\Lambda' = \{\lambda 0\}$ and $\Lambda'' = \{\lambda 1\}$. Assume that $M_{\alpha 0}$ and $M_{\alpha 1}$ are not OIR-compatible. That is, $\exists$ one sub-vector $s \in M_{\Lambda'} \cap M_{\Lambda''}$ such that $f(\Lambda' s) \neq f(\Lambda'' s)$ in OIR. Obviously, it contradicts the above condition in ONN. Therefore, such sub-vector does not exist, and $M_{\alpha 0}$ and $M_{\alpha 1}$ are OIR-compatible. □

In statistical learning, if a learning model satisfies Property 3.2, we say that this model is *over-fitting*. Therefore, to prevent the *over-fitting* problem, a modern statistical learning algorithms SV like SVM typically allows existing some $\vec{v} \in$ training data set M where $SV(\vec{v}) \neq f(\vec{v})$.

However, the notion of over-fitting and the related treatments in statistical learning are usually defined on the infinite continuous space (such as $\Re^n$). In the Boolean domain where the space (Boolean functions) is finite and discrete (enumerable), such notion cannot be applied directly. Therefore, what condition(s) to Property 3.2 would make the result over-fitting is not clear at this point. Whether the treatments in statistical learning still work in Boolean learning is under investigation.

It is also interesting to observe that Property 3.1 implies that the DD-based algorithm performs some *compression* on the data set. Superficially, ONN and OIR algorithms behave similarly to logic optimization algorithms. It is because the compatibility check also reduces the decision diagram size as other logic optimization algorithms do but fundamentally their goals are different.

The Boolean data mining is concerned more about the information complexity of the training data set, i.e. how much useful information is contained in the set of data. More specifically, given the simulation data and a selection of learning algorithms, Boolean data mining problem can be viewed as the problem of finding the best model that encodes all information presented in the data set while utilizing limited number of samples (not all) in the data set. Because the number of *care* samples used to construct the learned model is smaller than the total number of samples, a certain degree of induction can be carried out. Essentially, both ONN and OIR try to follow this principle.

IV. Comparison of NQ learning methods

In this section, we conduct experiments to assess the performance of NN, ONN, OIR, and SVM. We use the SVM package Torch3 [18] and assign *Gaussian kernels* [15] in the learning. Note that we did try using *Polynomial kernels* [15] in SVM and discovered that their performances were in general worse than using Gaussian kernels.

Performance was evaluated from two aspects: (1) learning accuracy, and (2) run-time efficiency in *training* a learned model and in evaluating the learned model. For all methods, learning accuracy depends on the input vectors used to generate the data matrix. In all experiments, we fix the number of input vectors in the data matrix to 100K for simplicity. For SVM, this number is reduced to 10K because otherwise, most runs would not finish after days. In addition, we use training vectors produced by *uniformly random* selection between 0 and 1 on each bit. Fixing the number of vectors and the way to generate these vectors ensures a fair comparison of all methods. We did try with other variations but found that they did not change the conclusions to be drawn later. Hence, we omit the discussion of these issues in detail.

	NN			ONN			OIR			SVM		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
c499	61.4	62.2	63.7	49.9	80.8	99.3	57.1	84.6	98.7	99.5	99.6	99.7
c880	49.9	56.8	60.4	49.8	71.2	100	49.7	56.6	100	50.4	91.9	100
c1908	52.2	59.5	63.1	49.7	85.2	99.7	59.7	89.8	100	58.7	86.9	99.9
c3540	50.1	54.5	54.5	50.0	67.0	100	50.0	55.0	100	49.5	70.9	100
c6288	49.8	52.8	52.9	49.8	57.1	100	49.6	55.3	95.9	48.5	59.7	100
c7552	49.8	52.5	56.6	49.6	56.0	100	49.6	56.0	100	49.3	75.6	100
total	train	evaluate		train	evaluate		train	evaluate		train	evaluate	
runtime	0.18 hr	> 10 days		3.86 hr	0.04 hr		3.37 hr	0.04 hr		> 1 day	32.8 hr	

TABLE III

Learning accuracy and run-time based on 100K random input vectors for four NQ learning methods

Experimental results: In the initial experiments, we use some combinational benchmark examples from [17]. These examples are good at evaluating the performance because they represent general Boolean functions. However, we emphasize, as shown in Figure 1, that our algorithm is not only intended for learning combinational circuits. Moreover, a learning algorithm treats the circuit as a black box. The circuit size is relatively un-important as long as the simulation is efficient. What impacts the performance of a learning algorithm the most is the complexity of the input-output behavior. Hence, a learning algorithm may perform very well on a large design but very badly on a small circuit block.

Table III summarizes the accuracy comparison (in %) between the four methods. In each case, we show the minimum, the average, and the maximum learning accuracy over all outputs. Total runtime is also provided to compare the efficiency.

Accuracy is evaluated based on a set of newly-generated 100K random vectors, by comparing their output values reported from each learned model, to the true answers reported by the simulator. Let R_1 be the percentage of the vectors such that the true answer is 1 and for those vectors, the learned model's answer is also 1, and R_0 be defined similarly. Then, we calculate the learning accuracy as $\frac{R_1+R_0}{2}$. For example, suppose the output values of the 10K patterns have 9999 1's and only one 0. If a model answers 1 with 100% accuracy but answers 0 with 0%, then the learning accuracy would be counted as 50%.

The average learning accuracy among four methods is particularly shown in Fig 5. In general, we can observe that SVM performs consistently better with respect to the learning accuracy but is considerably slower than ONN and OIR. NN is extremely slow in model evaluation and the accuracy is the worst. Note that the "training" time for NN is only the simulation time because an NN learned model is nothing but the original data matrix.

It is interesting to observe that although ONN also employs

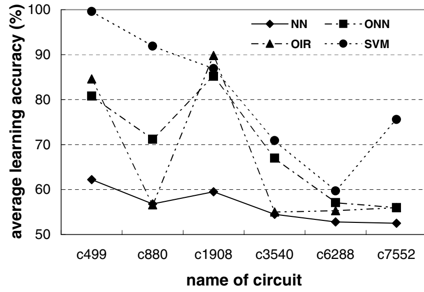


Fig. 5. Average learning accuracy of NQ learning methods

	c499	c880	c1908	c3540	c6288	c7552
upper bound (K)	84194	117808	45777	77684	55394	2076956
ONN DD size (K)	63716	113104	19904	73701	34956	1982320
comp. ratio	1.3	1.0	2.3	1.1	1.6	1.0
OIR DD size (K)	124	207	74	228	364	2603
comp. ratio	679.1	570.0	622.8	340.5	152.4	798.0
ONN OBDD size(K)	258	327	209	352	709	6975
OIR OBDD size(K)	117	199	70	221	343	2491
ONN BDD/OIR BDD	2.2	1.6	3.0	1.6	2.1	2.8

TABLE IV

TOTAL NUMBER OF DD NODES, COMPRESSION RATIOS AND OBDD NODES OVER ALL OUTPUTS

the Nearest Neighbor principle, by fixing an input variable ordering, ONN substantially outperforms NN. This is somewhat surprising because it indicates that the *ordering* of learning can be a significant factor in Boolean learning. We emphasize that both NN and SVM are not suitable for use in pattern justification, because it is not clear how to obtain inverse NN and SVM learned models (to efficiently compute LM^{-1}).

From the discussion in Section III-D, we know that both ONN and OIR perform the information compression on their learned models. Property 3.1 gives the upper bound of number of nodes that a DD-based model can have. Therefore, the compression ratio can be computed as the upper bound divided by the number of DD nodes by ONN and OIR, respectively. More interestingly, how about their final OBDD sizes? Table IV shows the DD sizes, compression ratios, and OBDD size for ONN and OIR. Our result shows that although OIR have much higher compression ratio then ONN on DD sizes, their OBDD sizes have no more than 2 times difference.

A. Boosting accuracy by changing ordering

The data above were obtained by following the netlist input ordering on each case. For ONN and OIR, we suspected that the learning accuracy could be sensitive to the input variable ordering in use. Hence, it might be possible to alter the input orderings in the experiments and improve the accuracy results. Suppose we were given with a data matrix and an output y to be learned. To find a good input ordering, we applied *association rule mining* (ARM) [19] on the matrix. For each input x, we calculated its *correlation* to the output based on the *Support-Confidence Framework* [19] in ARM. This correlation aimed to quantify the likelihood that a change of value on x may cause a change of value on y. If this correlation is high, we would rank x high.

	c499	c880	c1908	c3540	c6288	c7552
From Table 1: O (min)	49.9	49.8	49.7	50.0	49.8	49.6
New orderings: O' (min)	99.2	49.9	59.9	50.0	49.8	49.7
From Table 1: O (avg)	80.8	71.2	85.2	67.0	57.1	56.0
New orderings: O' (avg)	99.2	86.3	87.7	70.1	59.9	78.9

TABLE V

BOOSTING ACCURACY IN ONN BY USING O'

output#	1	2	20*	3	6*	7*	8*	9	12	17	13	18	19	14	16
SVM	100	100	90.1	87.6	82.8	82.6	81.6	78.4	71.1	68.4	65.3	65.0	64.9	64.8	64.6
ONN	100.0	99.6	51.6	76.0	52.8	58.5	60.8	73.4	79.7	67.3	62.8	64.6	56.6	60.0	63.4
OIR	99.9	88.0	51.6	75.5	52.9	58.3	58.0	70.7	76.5	67.2	62.5	64.7	56.6	60.0	63.4

TABLE VI

BASED ON SVM'S TOP-15 MOST ACCURATE OUTPUTS ON C3540

Table V shows the improved learning accuracy results based on ONN using the ARM-based orderings (results on OIR were similar and hence are not shown). We observe that using the new orderings (i.e. the rows denoted by "O'") could substantially boost the accuracy results. The average learning accuracy results becomes more comparable to those given by SVM in Table III.

Results on selected outputs: It is interesting to examine the results on a few selected outputs in more detail. Take c3540 as an example. Table VI summarizes the accuracy results on 15 selected outputs. These outputs were selected because they were the 15 most accurate outputs based on SVM. We observe that for outputs # 20, 6, 7, 8, SVM performs significantly better than ONN and OIR. In addition, we observe that ONN and OIR performs similarly on almost all outputs.

In addition, Table VII shows the comparison results on selected outputs from various circuits. We notice that for the four outputs of c3450, # 20, #6, #7, #8 mentioned above, using the new orderings could boost the accuracy on outputs #20, #6, #7 to similar levels as those achieved by SVM (shown in Table VI) but could not do so on output #8.

To sum up, from Figure 5, ONN performs slightly better than OIR in general and hence, for most cases it suffices to try just ONN. This is surprising because from Table IV OIR always has much higher compression ratio on DD sizes, which seems to be a better solution for the overfitting problem (but actually not necessarily). On the other hand, the *ordering* of learning can significantly impact the performance of ONN (and OIR), i.e. using the ARM-based orderings can substantially boost the accuracy results (Table VI and Table VII).

B. Potential of the engine

We demonstrate the potential of our learning engine by showing:

circuit	c3540				c499			c880	c6288			c7552	
output#	20*	6*	7*	8*	22	23	26	26	28	29	30	27	53
O	51.1	52.8	58.5	60.8	50.4	49.5	50.2	50.7	50.3	50.8	52.4	50.1	50.2
O'	88.9	83.8	81.2	67.3	99.3	99.2	99.2	67.5	76.9	79.1	98.9	80.9	75.1

TABLE VII

COMPARISON OF USING ORDERINGS O AND O' BASED ON ONN

out bit	1	4	8	12	16	20	24	28	32
True OBDD	4773	4773	4773	4773	4773	4773	4773	4773	4773
ONN's OBDD	1213	1131	1184	1256	1232	1272	1249	1277	1199
min-terms difference %	1.49	1.38	1.47	1.49	1.49	1.43	1.58	1.39	1.45

TABLE VIII

RESULTS ON C499 USING ONN WITH ARM-BASED ORDERINGS

output bit	13	14	15	16	25	26	27	28	29	30	31	32
ONN's OBDD	25K	25K	25K	25K	24K	23K	21K	12K	11K	16	7	3
accuracy %	49.4	51.0	50.8	48.4	51.6	55.0	57.8	80.0	81.2	98	99	99
True OBDDs	216K	596K	1631K	4554K	-	-	-	-	-	-	-	-

TABLE IX

1-BIT JUSTIFICATION ACCURACY AND LEARNED OBDD SIZES FOR SELECTED OUTPUTS ON C6288

1) A learned OBDD can be much smaller than the true OBDD while the functions represented by the two are still very similar.
2) A good approximate learned OBDD can still be developed even when computing the true OBDD is not feasible because of the exponential size blow-up.
3) When combining multiple outputs' learned OBDDs into one, a reasonable accuracy can be maintained.
4) Pattern justification can be effectively guided by combining learned OBDDs.

(1) Consider a function $f(x) = x_i \oplus h(x)$ where $h(x)$ is a complex function with a small percentage of inputs making $h(x) = 1$. Then, $x_i = 1 \Rightarrow f(x) = 1$ with a high probability, and $g(x) = x_i$ is actually a good approximation of $f(x)$. Table 8 shows results of this type of approximation on c499. Given the good orderings from association rule mining, the learned OBDDs from ONN are much smaller and have a min-term difference less than 1.6% when comparing the true OBDDs.

(2) It is well known that OBDD representation is not efficient for multipliers [5]. Table IX shows that for the most significant bits of a 16×16 multiplier, c6288, learning could obtain very good approximate OBDDs while the true OBDDs could not be computed due to OBDD blow-up in the middle of the circuit. The accuracy is measured by randomly justifying 0/1 using a learned OBDD 10K times, and simulating the 10K justified vectors on the circuit to verify their accuracy.

(3) Suppose we have m outputs with learned OBDDs $B_1, \ldots, B_m$ that achieve accuracy $\rho_1 \geq \ldots \geq \rho_m$. Suppose we are given a random and *justifiable* pattern $A_m = [a_1, \ldots, a_m]$ to be justified. Note that it is crucial to assume that A_m is justifiable. Otherwise, there is no chance for a learned model to produce an input vector to justify the pattern. Further note that to answer the question if a given pattern is justifiable or not, requires the use of a deterministic search method such as a SAT solver. A Boolean learning engine cannot answer that question. In our experiments, we only used justifiable output patterns because our purpose was to evaluate the effectiveness of learning, not the effectiveness of our learning algorithms to decide justifiability.

	c499	c880	c1908	c3540	c6288	c7552
# of selected outputs	32	19	25	8	7	68
k=10	95.6	93.7	93.4	94.6	97.1	44.9
k=100	96.6	100	95.2	100	100	68.9

TABLE X

LEARNING RATE COMPARISON WITH DIFFERENT K'S

For guiding the pattern justification, we need to combine the m learned OBDDs as $J_m = (B_1 \oplus \overline{a_1}) \wedge \cdots \wedge (B_m \oplus \overline{a_m})$. If we justify k patterns from J_m (as discussed in our problem formulation in Section II) , let $\delta_m(k)$ be the *(justification success) rate* that at least one of the k patterns achieves A_m in the simulation. Then, we expect that $\delta_m(k)$ increases as k increases, and decreases as m increases. Let $P_m = (\rho_1) * \cdots * (\rho_m)$. We see that P_m decreases as m increases. Figure 6 plots P_m and $\delta_m(k)$ for various m and k based on the learned OBDDs obtained using ONN with ARM-based orderings on c1908.

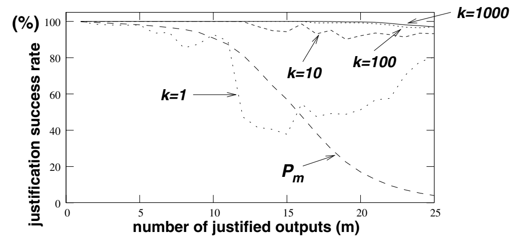


Fig. 6. Justification success rates for $m = 1 \ldots 25$ and $k = 1, 10, 100, 1000$ based on circuit c1908

Figure 6 demonstrates that for a large enough k (ex. $k = 100$), a random pattern consisting of 25 outputs can almost surely be justified based on combined result of individual learned OBDDs. Notice that for $m = 25$, $P_m < 10\%$ already. Hence, we see that by using a large k, the justification success rate can be much higher than simply multiplying the individual learning accuracies together (i.e P_m).

(4) For the remaining circuit examples, we continued to use ONN with ARM-based orderings. We conducted pattern justification experiments as follows: We selected all outputs whose learning accuracies were at least 70%. We produced 1K *justifiable* patterns on these outputs. We report the justification success rates in Table X. It is interesting to note that in Figure 2(b), c499 is classified as low learnability in [3]. With the help of ARM-based orderings, we can justify all outputs of c499 with success rate of 96.6% as $k = 100$.

In Table XI, we applied the learning engine to some ISCAS89 sequential circuits. We applied ONN with ARM-based orderings to obtain the best learned OBDDs. For each circuit, we assigned 0 to all flip-flops as the initial state. Then, the learning was based on 3-timeframe expansion similar to that illustrated in Figure 1. Note that we did not constrain any input and hence, the learning was based on all inputs across the 3 timeframes. 1K randomly generated justifiable output patterns at the end of the 3rd timeframe were used to calculate the success rates. We see that good results could be achieved

Circuit	PIs	POs	FFs	success rates k=10	success rates k=100	Circuit	PIs	POs	FFs	success rates k=10	success rates k=100
s641	35	24	272	95.6	95.6	s712	35	23	254	96.4	96.4
s1423	17	5	74	100	100	s9234	36	39	211	94.6	100

TABLE XI

PATTERN JUSTIFICATION BASED ON 3-TIMEFRAME SIMULATION

TTPG method	original	DD-based data miner
detected faults	18966	19049
fault coverage	94.94%	95.36%
test coverage	97.40%	97.82%

TABLE XII

FAULT COVERAGE COMPARISON ON THE ALU MODULE

on these examples.

C. Application to OpenRISC 1200 processor

OpenRISC 1200 is a public microprocessor core, which many industrial applications use, such as an SOC design from Flextronics Semiconductor and a speech recognizer from Voxi Inc. The current design is a 32-bit scalar RISC with Harvard microarchitecture and a 5-stage integer pipeline supporting 52 core instructions. An implementation of the CPU core, synthesized by Synopsys Design Compiler, contains 285218 collapsed stuck-at faults and 2349 state-holding elements.

We incorporate the proposed Boolean data miner into the TTPG methodology, and mainly focus on control signals in the ALU module. ALU contains 3608 gates performing arithmetic, logic, comparison and shift/rotate operations. 30 core instructions can produce signal activities on the ALU. The total number of stuck-at faults in the ALU is 19976 in which 19472 faults (structural fault coverage 97.47%) are reported testable by Fastscan.

The overall fault coverage of DD-based data miner and previous method[3] is compared and shown in Table XII. The result shows that the TTPG methodology with DD-based data miner can also achieve high test coverage as the original TTPG methodology does. The minute difference can be reasonably expected since the OpenRISC, in general, is still easy to learn. The arithmetic learning engine (polynomial testing and interpolation) in the TTPG methodology works very well on this arithmetic-intensive block. DD-based data miner helps to capture additional 83 detected faults only coming from the signal associated with logic operations and control signals in the ALU. In the future, we expect to find a more complex logic-intensive design to see more difference.

V. CONCLUSION AND FUTURE WORK

Deterministic functional test pattern generation has been a long-standing open problem, which is important for both design verification and manufacturing testing. A simulation-based data-mining methodology, called *TTPG* [2][3], was proposed as an alternative approach for functional test justification. However, Boolean learning problem is the core of simulation data mining.

In this work, several types of Boolean learning algorithms are studied. The DD-based learning approach performs well in terms of both efficiency and accuracy. Our study also shows that *ordering* of learning is crucial and can boost learning accuracy. Experimental results show its effectiveness in guiding pattern justification. From the aspect of practicality, the TTPG methodology incorporated with our DD-based Boolean data miner can achieve a high fault coverage (95.36%) on OpenRISC 1200 processor core and capture more faults when comparing to the previous method in [3].

In the future, several interesting issues are worth exploring, including sampling scheme, sampling size, optimal input ordering, and etc. Moreover, practical research is required to combine the Boolean learning engine with other search engines such as SAT or ATPG to more efficiently solve the justifiability problem mentioned before.

REFERENCES

[1] A. Aharon, D. Goodman, M. Levinger, Y. Lichtenstein, Y. Malka, C. Metzger, M. Molcho, and G. Shurek, "Test program generation for functional verification of PowerPC processors in IBM," Proc. Design Automation Conf., pp. 279-285, 1995.

[2] C. Wen, L.-C. Wang, K.-T. Cheng, W.-T. Liu, and C.-C. Chen, "Simulation-based target test generation techniques for improving the robustness of a software-based self-test methodology," Proc. Int'l Test Conf., pp. 936-945, 2005.

[3] C. Wen, L.-C. Wang and K.-T. Cheng, "Simulation-based functional test generation for embedded processors," to appear in IEEE Tran. Computers, 2006.

[4] M.J. Kearns, and U.V. Vazirani, "An introduction to computational learning theory," The MIT Press, 1994.

[5] R.E. Brayant, "Graph-based algorithms for Boolean function manipulation," IEEE Tran. Computers, vol. 35, no. 8, pp. 677-691, 1986.

[6] L. Valiant, "A theory of the learnable," Communications of the ACM, vol. 27, no.11, pp.1134-1142, 1984.

[7] M. Kearns, M. Li, L. Pitt and L. Valiant, "On the learnability of Boolean formulae," Proc. 19th Symp. on Theory of Computing, pp. 285-295, 1987.

[8] M. Kearns, M. Li, L. Pitt and L. Valiant, "Recent results on Boolean concept learning," Proc. 4th Int. Workshop on Machine Learning, pp. 337-352, 1987.

[9] M. Kearns, and L. Valiant, "Learning Boolean formulae or finite automata is as hard as factoring," Technical Report TR-14-88, Harvard University, 1988.

[10] D. Angluin, "Queries and concept learning," Machine Learning, vol. 2, no. 4, pp.319-342, 1987.

[11] E. Kushilevitz and Y. Mansour, "Learning decision trees using the fourier spectrum," SIAM Jour. Computing, vol. 22, no.6, pp.1331-1348, 1993.

[12] N. Linial, Y. Mansour, and N. Nisan, "Constant depth circuits, Fourier transform, and learnability," Journal of ACM, vol. 40, no. 3, pp. 607-620, 1993.

[13] J. Jackson, "An efficient membership-query algorithm for learning DNF with respect to the uniform distribution," Journal of Computer and System Sciences, vol. 55, no. 3, pp. 414-440, 1997.

[14] T. Hastie, R. Tibshirani, and J. Friedman, "The elements of statistical learning - date mining, inference, and prediction," Springer, 2001.

[15] N. Cristianini, and J. Shawe-Taylor, "An introduction to support vector machine and other kernel-induced-based learning methods," Cambridge University Press, 2002.

[16] V. N. Vapnik, "The nature of statistical learning theory," Springer-Verlag, 1999.

[17] http://www.bdd-portal.org/evaluation/evaluation.html

[18] R. Collobert, S. Bengio and J. Marithoz. "Torch: a modular machine learning software library," Technical Report IDIAP-RR 02-46, IDIAP, 2002.

[19] C. Zhang and S. Zhang, "Association Rule Mining," Springer, 1998.

[20] http://www.opencores.org/projects.cgi/web/or1k/openrisc_1200, OpenRISC 1200 Specification, Operncores.org.