Int J Softw Tools Technol Transfer (2011) 13:247–261
DOI 10.1007/s10009-010-0160-z

HVC 2008

Automatic boosting of cross-product coverage using Bayesian networks

Dorit Baras · Shai Fine · Laurent Fournier · Dan Geiger · Avi Ziv

Published online: 16 May 2010

Abstract Closing the feedback loop from coverage data to the stimuli generator is one of the main challenges in the verification process. Typically, verification engineers with deep domain knowledge manually prepare a set of stimuli generation directives for that purpose. Bayesian networks based CDG (coverage directed generation) systems have been successfully used to assist the process by automatically closing this feedback loop. However, constructing these CDG systems requires manual effort and a certain amount of domain knowledge from a machine learning specialist. We propose a new method that boosts coverage in the early stages of the verification process with minimal effort, namely a fully automatic construction of a CDG system that requires no domain knowledge. Experimental results on a real-life cross-product coverage model demonstrate the efficiency of the proposed method.

Keywords Functional verification · Coverage · Coverage directed generation · Bayesian networks

D. Baras (✉) · S. Fine · L. Fournier · A. Ziv
IBM Research Laboratory in Haifa, Haifa, Israel
e-mail: doritb@il.ibm.com

S. Fine
e-mail: fshai@il.ibm.com

L. Fournier
e-mail: laurent@il.ibm.com

A. Ziv
e-mail: aziv@il.ibm.com

D. Geiger
Department of Computer Science, Technion, IIT, Haifa, Israel
e-mail: dang@cs.technion.ac.il

1 Introduction

Functional verification remains one of the main challenges in the hardware design cycle [1]. In current industry practice, dynamic verification is the leading technique for functional verification. To cope with the ever increasing complexity of modern designs, the verification process is highly automated. It relies on sophisticated tools to replace the human engineer in almost every aspect of operating the verification environment, such as generating stimuli for the *design under verification* (DUV), and checking that the DUV behavior is correct according to its specification [1].

Functional coverage is the main technique for checking that the verification has been thorough [2]. Coverage can help monitor the quality of verification and direct the verification team toward areas that have not been adequately verified. The analysis of coverage information and the use of this information to direct the stimuli generator toward uncovered or lightly covered areas is one of the remaining human bottlenecks in today's verification environment. Therefore, considerable effort is spent finding ways to automate the covering procedure; that is, to close the loop of coverage analysis and stimuli generation. Although in early stages of the verification process, reaching 100% coverage is not the main priority of the verification team, reaching a high level of coverage as fast as possible is important because bugs found in the early stages of the design require far less time and effort to fix. Consequently, it would be helpful to boost coverage in the early stages of verification with minimal effort from the verification team. This requirement motivated the work presented in this paper.

Data-driven coverage directed test generation (CDG) is a technique to automate the feedback from coverage analysis to stimuli generation. In data-driven CDG, the CDG system discovers the relations between the directives that control the

stimuli generation and the coverage events, based on observations of specific settings of the directives and the coverage events to which they lead. Reports on several CDG systems of this kind have been published in recent years, including systems based on Bayesian networks [3,4], Markov chains [5], genetic algorithms [6], and inductive logic [7].

Bayesian networks [8] are well suited to address the challenges of data-driven CDG and provide the kind of modeling required for CDG. First, Bayesian networks offer a natural and compact representation of the rather complex relationship between the CDG ingredients, namely, the coverage events on the one hand and the test directives on the other. In addition, Bayesian networks provide the ability to encode essential domain knowledge in the CDG system. As a result, Bayesian network CDG systems were able to produce high coverage and a high coverage rate in several industrial settings [3,4,9,10]

The CDG approaches mentioned above require a certain amount of domain knowledge from an expert familiar with the design details and an expert in machine learning. In addition, construction of the CDG system may require significant effort that cannot be invested in early stages of the verification process. We propose a fully automated method for constructing CDG systems based on Bayesian networks, without the need for domain expertise. In contrast to existing works using Bayesian networks [3,4], we suggest a process that does not require stages of preprocessing or help from either verification engineers or machine learning specialists. Similar to the manual construction process described in [3], the components of the automated process include three main stages: feature selection in which selection of relevant directives is performed; structure learning, where the structure of the Bayesian network is defined; and parameters learning.

The feature selection stage, which is formally defined in Sect. 6, is needed to narrow down the number of nodes in the network which makes the tasks of learning the structure and parameters more efficient. It also improves the networks quality because effects of noise are filtered. As will be shown, this task is quite difficult for the CDG domain. Identifying the structure of the Bayesian network is the most difficult part in the manual process. In the automated process, we replace the manual construction with several generic structure learning algorithms [11–14]. We then enhance these algorithms with two heuristics that suit the characteristics of the CDG setting, namely, pruning edges between directives and quantizing the probabilities of directives. These automatically constructed networks may not be as good as the manually constructed ones, but they do provide enough power to achieve the goals of coverage boosting.

We tested our method on a cross-product coverage model used in the verification of the instruction fetch unit (IFU) of the IBM z10 processor. Our results indicate that the automatic booster approach is working. We were able to achieve

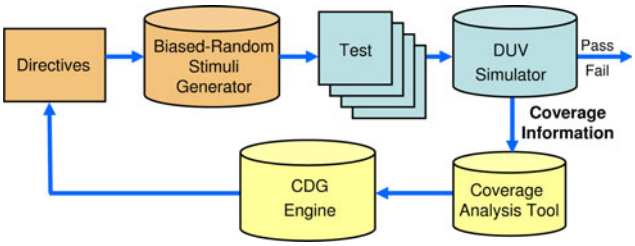


Fig. 1 Structure of a verification environment with CDG

significant improvement in coverage over the regression suite used in the verification process with a fully automated process that requires minimal effort. Our proposed feature selection methods yield beneficial sets of directives, and we were able to produce good networks. Moreover, the two heuristics we propose improve the quality of the learned Bayesian networks and perform better than the generic structure learning algorithms.

The rest of the paper is organized as follows: Sect. 2 provides details on Bayesian networks and their application to coverage directed generation. In Sect. 3, we explain the concept of the coverage booster. Section 4 describes the setting of the experiments to follow. Section 5 describes the fully automatic process of the coverage booster. Section 6 describes the feature selection procedures and their related results. Section 7 describes the structure and parameter learning procedures and thier corresponding experimental results. Section 8 presents the results of the entire booster. We conclude and present directions for future work in Sect. 9.

2 Coverage directed generation using Bayesian networks

In today's highly automated verification environment, analysis of coverage information and use of this information to direct the stimuli generator toward uncovered or lightly covered areas is one of the remaining human bottlenecks. Therefore, considerable effort is spent finding ways to automate the covering procedure. That is, to close the loop of coverage analysis and stimuli generation. This automated feedback from coverage analysis to stimuli generation, known as *Coverage Directed stimuli Generation* (CDG), can reduce the manual work in the verification process and increase its efficiency. In general, the goal of CDG is to automatically provide the stimuli generator with directives that are based on coverage analysis [3]. Figure 1 presents a sketch of a verification environment with CDG. The CDG engine receives information from the coverage analysis tool about the state and progress of the coverage, and generates directives to the random test generator that are designed to achieve one or many of the CDG goals.

There are two main approaches to CDG. In direct CDG, or model-based CDG, an external model of the design under verification is used to generate stimuli to accurately hit the coverage events [15]. In data-driven CDG, which is often called feedback-based CDG, the CDG system relies on inference of the required stimuli directives from observations of past behaviors [3]. This inference is usually done with machine learning techniques [3,6,7]. In this paper, we refer to CDG systems based on Bayesian networks for cross-product coverage [3]. The rest of this section explains how such systems are constructed and used.

2.1 A brief introduction to Bayesian networks

A Bayesian network is a graphical representation of the joint probability distribution for a set of variables [8]. A Bayesian network consists of two components. The first is a directed acyclic graph in which each vertex corresponds to a random variable. This graph represents a set of conditional independence properties of the represented distribution: each variable is independent of its non-descendants in the graph given the state of its parents. The graph captures the qualitative structure of the probability distribution, and is exploited for efficient inference and decision making. The second component is a collection of local interaction models that describe the conditional probability $p(X_i|Pa_i)$ of each variable X_i given its parents Pa_i. Together, these two components represent a unique joint probability distribution over the complete set of variables X [8]. The joint probability distribution is given by the following equation:

$$p(X) = \prod_{i=1}^{n} p(X_i|Pa_i) \tag{1}$$

It can be shown that this equation implies the conditional independence semantics of the graphical structure given earlier. Equation 1 shows that the joint distribution specified by a Bayesian network has a factored representation as the product of individual local interaction models. Thus, while Bayesian networks can represent arbitrary probability distributions, they provide a computational advantage for those distributions that can be represented with a simple structure.

Following Pearl [8], Fig. 2 describes a simple example of a Bayesian network. The network contains five indicator random variables as its nodes: earthquake to indicate whether an earthquake occurred, burglary to indicate if someone broke into the house, radio to indicate whether the earthquake was announced on the radio, alarm if the alarm went off in the house, and call to indicate that the security company notified the owner about the alarm.

The edges in the graph indicate direct influence of the source node on the target node. For example, an earthquake directly increases the probability of setting the alarm because

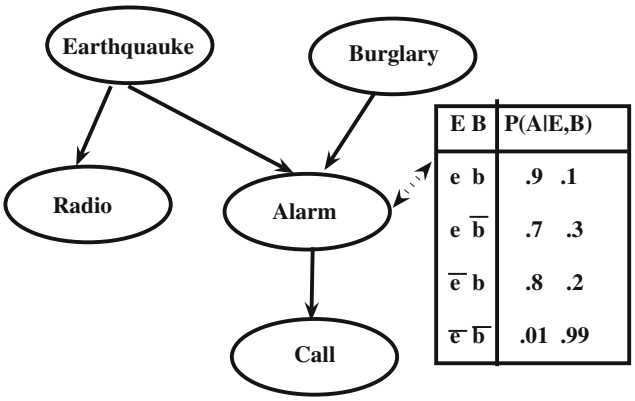


Fig. 2 Simple example of a Bayesian network

of the movements in the house that may trigger the alarm sensors. Similarly, the probability of receiving a call from the security company is directly dependent on the alarm actually being activated. Note that a lack of edge or directed path between nodes in the graph does not mean that the two variables are independent. For example, receiving a call from the security company after hearing about an earthquake on the radio reduces the probability that someone broke into the house, as compared to the case where there was no announcement about an earthquake.

The conditional probabilities that describe the local interactions between a node and its parents are often stored in tables adjacent to the node. The table on the right hand side of Fig. 2 holds the probability for the state of the alarm node conditioned on the occurrence of burglary and earthquake. For example, if no burglary and no earthquake occurred, the probability of the alarm being activated is 1%; while if the house was broken into and no earthquake occurred, the probability of setting the alarm is 80%. Given all the conditional probabilities of the nodes in the graph, we can use Eq. 2 below to express the joint probability in the space induced by the five nodes.

$$\begin{aligned} &p(B, E, R, A, C) \\ &\quad = p(B) \cdot p(E) \cdot p(R|E) \cdot p(A|B, E) \cdot p(C|A) \end{aligned} \tag{2}$$

The characterization given by Eq. 1 is a purely formal characterization in terms of probabilities and conditional independence. An informal connection can be made between this characterization and the intuitive notion of direct causal influence. It has been noted that if the edges in the network structure correspond to causal relationships, where a variable's parents represent the direct causal influences on that variable, then resulting networks are often very concise and accurate descriptions of the domain. Thus, it appears that in many practical situations, a Bayesian network provides a natural way to encode causal information. Nonetheless, it is often difficult and time consuming to construct Bayesian networks from expert knowledge alone, particularly because

of the need to provide numerical parameters. This observation, together with the fact that data is becoming increasingly available and cheaper to acquire, has led to a growing interest in using data to learn both the structure and probabilities of a Bayesian network (cf. [8,16,17]).

Typical types of queries that can be efficiently answered by the Bayesian network model are derived from applying the Bayes rule to yield posterior probabilities for the values of a node (or set of nodes), X, given some evidence, E, i.e., assignment of specific values to other nodes:

$$p(X|E) = \frac{p(E|X) \cdot p(X)}{p(E)} \tag{3}$$

Thus, a statistical inference can be made in the form of either selecting the *Maximal A Posteriori* (MAP) probability, $\max p(X|E)$, or obtaining the *Most Probable Explanation* (MPE), $\arg\max p(X|E)$.

2.2 Bayesian networks for CDG

The idea behind using Bayesian networks for CDG cross-product coverage models starts from the understanding that the space containing the directives to the stimuli generator on one side and the coverage model on the other side is a large distribution space. On one side of this space stand the directives to the stimuli generator, which are defined as probability distributions over a domain of values. On the other side stands the coverage model that is defined as a cross product over a finite set of variables which are named attributes. Each attribute is defined over a finite, possibly categorical, set of values. Activating the verification environment[1] with different settings of the directives, or even with the same settings but different random seed, yields different coverage events. Therefore, the coverage attributes can also be viewed as random variables.

Bayesian networks can represent this large distribution space in a compact form. The structure of the network captures the true dependencies between the various components of the space. Specifically, it captures directives that directly affect the values of specific coverage attributes and dependencies between the values of various coverage attributes. Once the structure and parameters of the Bayesian network are defined, an *abductive* query that provides evidence on the effect nodes (desired coverage events) can be used to determine the possible cause (directives settings).

Constructing a Bayesian network for a CDG system comprises three main steps. The first step is selecting the relevant directives to be used in the system. The number of inputs to the stimuli generator, namely directives, is narrowed down to avoid networks that are too large for training and inference. It also reduces the amount of data required for the learning process that follows. The second step, which is the most difficult one, involves defining the network structure. In many applications of Bayesian networks, the structure of the network is defined manually by a domain expert. Although structure learning algorithms exist (e.g., [11,18]) their performance is usually inferior to manually constructed networks. The last step involves estimating the conditional probabilities of each node in the network. There are efficient algorithms that can learn these parameters from observations on data in the form of values to (some or all) the network nodes. In the CDG case, the set of observations is a sample set of directive settings along with their coverage events as resulting from activating the simulation environment. The constructed Bayesian network can be used in a CDG system to determine directives for a desired coverage event.

It is important to note that the use of Bayesian networks for CDG is different from most "classical" uses. These characteristics are caused by the way stimuli generators and verification environments behave. In many cases, stimuli generators use the settings of the same directive many times during their operation, each time randomly choosing a different value according to the distribution specified in the settings. This allows them to generate rich sequences of values out of one directive. Therefore, it is important to provide the Bayesian network and the algorithms that learn it with settings that specify probability distributions on the possible values of a directive, not just settings that determinately define its value. We call such settings *soft evidence*. The implications of using soft evidence in the feature selection and automatic construction of a Bayesian network are discussed in Sects. 6 and 7 respectively.

3 The coverage booster

Coverage closure is commonly acknowledged as one the most important goals of the verification process. The recurring observation in this domain is that a portion of the events to be covered are not reached through the initial attempts. These events are usually some of the most complex and subtle events, since they resisted the preliminary regular attempts to create them. Therefore, this involved task is usually carried out by experts, both in the application domain and in the stimuli generation technology.

Due to the high cost of the coverage closure task, both in terms of time investment and expertise required, much attention has been geared toward inserting some automation into this classically manual process. Our approach, which relies on machine learning techniques [3], has shown good initial signs indicating the potential for automating this task. Indeed, there were several success stories [4,10] that demonstrated the adequacy of this approach, and showed that this

[1] That is, generating stimuli using the directives settings, simulating the DUV, and obtaining coverage data.

apparently intrinsically manual task could be modeled and performed by a program.

While machine learning techniques have shown their capacity to capture part of the compound relationship between events and stimuli generation directives, and automate the process of closing the loop between stimuli generation directives and coverage events, constructing the CDG system is still a manual process requiring both expertise and effort. This significantly limits the opportunities of CDG to provide real benefits to the verification process. First, CDG is beneficial only in places where it replaces significant effort in coverage closure. This usually means that there are either large and complex coverage models [10], or extremely important coverage events [4]. In addition, CDG is possible only when a significant effort is directed at coverage closure. In many cases, this happens only towards the end of the verification effort.

The "Coverage Booster" is our new approach for exploiting the demonstrated capacity of machine learning in this domain, while showing an improvement, or boost, in efficiency measured in overall human effort. Instead of placing full coverage as a primary goal, we target minimal human effort above any other consideration. The coverage booster may not reach full coverage, but because of the zero human effort spent, covering new events and improving the coverage rate are enough to create real benefits to the verification process. Therefore, the coverage booster expands the envelope of opportunities for CDG in two ways. First, CDG can be useful when the verification team cannot allocate much effort for additional coverage models. Second, CDG can be used in earlier stages of the verification process, before coverage becomes a top priority.

4 Experimental environment

To evaluate our proposal we conducted several experiments using the verification environment for the z10 processor's instruction fetch unit (IFU), which is built into the latest IBM System z (mainframe) computers [19]. The IFU is responsible for efficiently requesting and buffering instruction fetches from memory, pre-decoding these instructions, and transferring them to the decode and execution units for additional processing. One of the most important functions of the unit, which has significant impact on the processor performance, is branch prediction. This is a complex task which includes the prediction of several aspects of a branch such as whether it is taken or not and its target address. Successful branch prediction can reduce the delay penalty of waiting for the actual outcome of the branch.

The IFU is connected to several other units. Its main interfaces are to the cache units that send instructions to the IFU based on fetch requests; the decode unit that receives predecoded instructions for the IFU; and the execution units that send information to the IFU regarding the execution of instructions. The verification environment of the IFU contains stimuli generators that control the interfaces of the IFU with these units. The behavior of the generators is controlled by a large set of directives that are provided by the user. These directives affect various aspects of the behavior of the unit interfaces. For example, there are directives that control the frequency of branch instructions in the instruction steam that is fed via the interface to the cache units, directives that control the actual outcome of branch instructions, and directives that control the frequency and type of flushes arriving from the execution units.

To ensure the quality of the verification process, the IFU verification team uses several cross-product coverage models that capture various aspects of the unit state and behavior. During simulation, the verification environment collects coverage events for these coverage models and saves them in trace files. Later, these trace files are processed by coverage measurement and analysis tools that produce coverage reports for the users and the CDG system. Our experiments were focused on one of these coverage models, namely, the IfuPipe model. This coverage model examines the state of the processor pipeline, or more precisely, the state of the pipeline in the IFU and nearby units. The IfuPipe model contains attributes that describe the state of each pipe stage, as well as other flags relating to branches and pipe recycles. There are 13 attributes in the model, with domain sizes of 2–3. There are 139,968 events in the coverage space, out of which approximately 27,000 events are included in the legal and interesting subspaces by the verification team. The verification environment of the IFU includes hundreds of directives, out of which, our experimental settings included a set of 23 directives provided by the IFU verification team. This set of directives is supposed to be the one that best controls the behavior of the IFU as captured by the IfuPipe model. Most of the directives in the set control the behavior of branch instructions and branch prediction.

The experiments described in this paper were done in two parts. The first part, as reported in Sect. 7.3, was done during the verification of the unit. This part of the experiment contributed directly to the verification of the unit by improving the coverage of the IfuPipe model and helping the verification team reach areas in the model that they could not reach before. The second part, with results reported in Sect. 6.3 was done after the verification of the unit and the entire z10 processor was completed. Major changes in the design and the verification environment between the experiments prevent us from comparing these two sets of results.

The interface between the verification environment and the CDG system is done in two ways. During execution of the CDG system, it extracts coverage information from the coverage analysis tool and uses this information and its under-

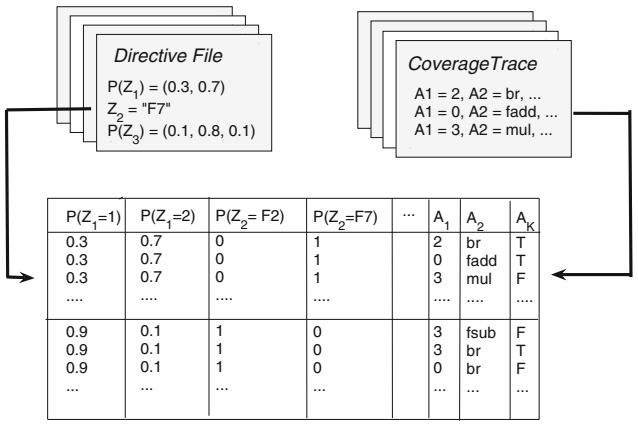


Fig. 3 Input data to the CDG process

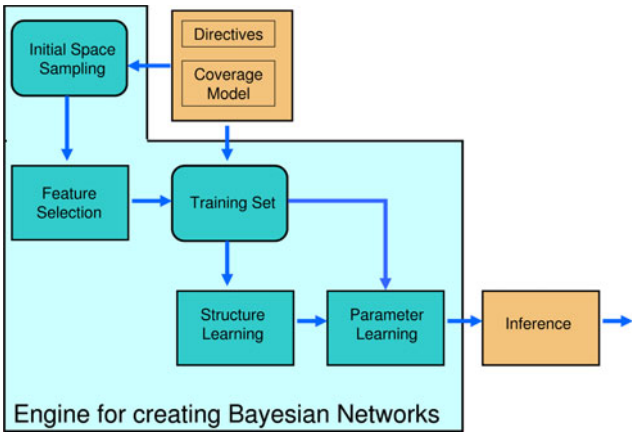


Fig. 4 Construction of the Bayesian networks for CDG

standing of the relation between coverage and the directives to produce directive files that are fed into the verification environment, as depicted in Fig. 1. During the construction of the CDG system, the verification environment provides the constructor of the CDG system with directive settings and the coverage to which these settings lead. Specifically, each simulation run is controlled by a directive file that provides distributions over possible values for some directives and assignments to specific values for others. The simulation, in turn, provides coverage traces, with each trace containing all the coverage events of a specific model that occurred during the run. These directive files and coverage traces pairs are presented to the CDG system in a tabular form as shown in Fig. 3. The table is composed of two parts, one that contains the settings of the directives and another that contains the coverage events. Each row in the table contains a coverage event and the settings of the directives in the simulation run that produced this event. Since each simulation run (with specific settings) produces many coverage events, the table contains many rows whose directive part is identical, as shown in Fig. 3.

5 Automatic construction of CDG engine

At the heart of a Bayesian network-based CDG engine lies a Bayesian network that allows a compact yet accurate description of the stochastic space of the coverage attributes on the one side and the stimuli generator directives on the other side. As noted previously, the goals of our coverage booster are to eliminate the need for manual intervention and boost coverage in early stages of the verification process. To meet these goals, the Bayesian network needs to be created automatically. The automated process described in this section generally follows the manual process described in [3], except that the manual steps requiring domain expertise in either the DUV and verification environment or machine learning techniques are replaced with automated steps. This enhanced automation may lower the quality of the Bayesian network and prevent it from reaching coverage closure. However, it does provide enough power to achieve the goal of coverage boosting.

The automated process is composed of three stages: feature selection, structure learning, and parameter learning. Fig. 4 illustrates the automatic process that creates a Bayesian network, which is later used for coverage boosting. Initially, the verification team provides a description of the coverage model and a list of directives that control the stimuli generator. The number of directives needs to be cut to allow efficient learning. In order to perform this selection we create an initial random sampling of all the directives that were provided and simulate with them. This is followed by a feature selection phase in which we narrow down the number of directives that are used later; we do this by choosing those with maximal influence. Next, we create a training set with random sampling of the directives we chose, while the rest of the directives are given default values, which are provided by the verification team. This training set is the input to the structure learning procedure that is followed by a parameter learning phase in which the conditional probabilities of the variables are estimated. At this point we have a Bayesian network that can be used in a CDG engine. The following sections provides more details on each of the stages of the automatic construction.

6 Feature selection

A typical verification environment contains tens or even hundreds of directives, which control the stimuli generator and affect the stimuli it generates. Learning with such a large set of directives is very difficult or even impossible. Therefore, it is essential to narrow down the number of directives and select a small high quality subset for the CDG process. It turns out that in most cases only a small subset of the directives are required to control the coverage of a specific model,

so this reduction does not reduce the ability of the coverage booster to achieve its goal.

Feature selection is a general term given to a variety of algorithms that extract the subset of features most relevant for a given task [20,21]. For CDG, the features are directives and the feature selection task involves finding directives that highly influence the coverage attributes. Feature selection in the CDG settings has several unique characteristics that make it difficult. The first, and most difficult issue, is the fact that the data is not fully observed. Most algorithms perform feature selection on a feature space that is fully observed. That is, the values of all features are known for every sample. In the verification case, we are not given the deterministic values of the directives, but rather a distribution over them. Second, the data is very noisy because of the highly stochastic nature of the simulation environment. Third, dominant values that are present in some of the coverage attributes can mask a dependencies between directives and a coverage attributes that affect dominated values. For example, if a specific value of an attribute appears in 90% of the samples, it would be hard to detect that a given directive can change the frequency of other value of that attribute from 5 to 6%. An additional problem is that the data is categorical and not ordinal, so feature selection methods based on ordering (such as [22]) cannot be applied. Because of these unique characteristics, commonly used algorithms cannot be applied as is.

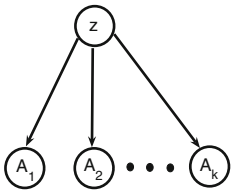
In addition to those problems, in CDG systems the feature selection problem is a many-to-many task (in contrast to many-to-one tasks like classification). Our solution divides the task of selecting the relevant features into two parts: estimating connection strength and choosing directives. First, we estimate the strength of the connection between each attribute-directive pair. This is followed by a step of choosing the directives that have the strongest impact on all the attributes under some predefined criterion. We do not take into account relations of more than single directive at a time. These relations are handled by the structure and parameter learning algorithms described later.

6.1 Estimation of the influence matrix

We use a measure based on *mutual information* [23] to measure the level of influence between pairs of attribute-directive. In information theory, mutual information is a quantity that measures the information between two random variables. Let $P(X)$ and $P(Y)$ be the marginal distributions of X and Y respectively and let $P(X, Y)$ be the joint distribution over the two random variables. The mutual information between the random variables is given by

$$I(X;Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

Fig. 5 Naive Bayes model



This measure does not directly reveal the type of connection between the pair (it does not connect certain values together), but it captures connections and scores the strength of influence between the two random variables.

To overcome a problem of low uncertainty in either a directive or an attribute, caused, for example, by dominant values, we normalize the mutual information. The measure that we actually use is:

$$NMI(A;Z) = \frac{I(A;Z)}{\min\{H(A), H(Z)\}}$$

where $H(X) = -\sum_x P(x) \log P(x)$ is the Shannon entropy [23], which measures of the uncertainty in a random variable.

Algorithm 1 introduces a method for estimating the influence matrix $IM(A;Z)$, where A is an attribute and Z is a directive. Estimating the influence (here, the normalized mutual information) requires estimating the joint probability $P(A, Z)$ and the marginal probabilities $P(A)$ and $P(Z)$. We use a naive Bayes model for each individual directive in which the directive is a parent of all attributes, and there are no additional edges in the network, as shown in Fig. 5. The main part of the algorithm is the Expectation Maximization procedure [8], in which the the conditional probability $P(A|Z)$ and the probability $P(Z)$ are estimated from the training data.

6.2 Subset selection

Given an estimation of the strength of connection between every attribute-directive pair, the next task is to select a "good" subset of directives that have a strong effect on all the attributes. There are several observations regarding this task. First, this is a many-to-many task, in which the number of required directives can be smaller than the number of attributes. Therefore, the impact of this task cannot simply be divided into choosing the "best" directives for each attribute and then taking the union of all directives. Second, the behavior of each attribute alone versus all directives is not necessarily fixed; there may be attributes that are easily controlled and strongly connected (high influence value) to many or even all directives. On the other hand, there are attributes that are hard to control and are weakly connected to all directives. To overcome this problem, the algorithms we suggested made assumptions about the behavior of the influence of all directives on an attribute. These assumptions are presented in the section corresponding to the specific algorithms.

Input : D - Training data, as shown in Fig. 3
Output: IM - Influence Matrix, where $IM_{i,j}$ is the influence of directive Z_j on attribute A_i

for $j \leftarrow 1$ **to** N **do**
  `EM(`D_j`) //` $D_j \leftarrow$ `The relevant columns to` Z_j `and all attributes in` D
  for $i \leftarrow 1$ **to** M **do**
    `// Estimate probabilities`
    $P(A_i = a_i) \leftarrow \sum_{z_j} P(a_i|z_j)P(z_j)$
    $P(a_i, z_j) \leftarrow P(a_i|z_j)P(z_j)$
    `// Calculate influence`
    $I(A_i, Z_j) \leftarrow \sum_{z_j, a_i} P(a_i, z_j) \log \frac{P(a_i, z_j)}{P(a_i)P(z_j)}$
    $H(A_i) \leftarrow -\sum_{a_i} P(a_i) \log P(a_i)$
    $H(Z_j) \leftarrow -\sum_{z_j} P(z_j) \log P(z_j)$
    $IM_{i,j} \leftarrow \frac{I(A_i;Z_j)}{\min\{H(A_i),H(Z_j)\}}$ `// The influence is the normalized mutual information`
  end
end

`EM(`D`)`
  Input : D - The relevant columns directive Z and all attributes in the training data
  Output: $P(A_i|Z), i = 1, 2, \ldots, M$ - The conditional probability of each attribute A_i given Z
    $P(Z)$ - The marginal probability of Z
  `// Initialize`
  for $i \leftarrow 1$ **to** M **do** $P(A_i|Z) \leftarrow$ random probabilities
  $P(Z) \leftarrow$ random probabilities
  $N \leftarrow$ number of rows in D
  repeat
    `// E Step`
    foreach *row* $r \in D$ **do**
      $e_r \leftarrow (a_{1,r}, a_{2,r}, \ldots, a_{N,r})$ `// The values of all the attributes in row` r
      foreach $z_k \in domain(Z)$ **do**
        $D_r(z_k)$ denotes the probability of $Z = z_k$ in row r
        $P(z_k|e_r) \leftarrow \frac{\prod_i P(a_{i,r}|z_k) \cdot D_r(z_k)}{\sum_{z_l} \prod_i P(a_{i,r}|z_l \cdot D_r(z_l))}$
      end
    end
    `// M Step - Estimate the parameters of the current Naive Bayes model using maximum likelihood`
    foreach $z_k \in domain(Z)$ **do** $P(Z = z_k) \leftarrow \frac{1}{N} \sum_r D_r(z_k)$
    for $i \leftarrow 1$ **to** M **do**
      foreach $z_k \in domain(Z), a_l \in domain(A_i)$ **do** $P(a_l, z_k) \leftarrow \frac{1}{N} \sum_{r, a_{i,r} = a_l} P(z_k|a_l)$
      foreach $z_k \in domain(Z), a_l \in domain(A_i)$ **do** $P(a_l|z_k) \leftarrow \frac{P(a_l, z_k)}{\sum_{a_m \in domain(A_i)} P(a_m, z_k)}$
    end
  until *change in estimation between iterations is lower than a given threshold*
end

Algorithm 1: Calculate the influence matrix IM

6.2.1 Greedy algorithm

Our experiments in real verification environments revealed three main types of expected behaviors between attribute-directive pairs. First, there are many directives that have a weak influence on all attributes (for a given model). These are the main candidates to be filtered out in the feature selection process. Second, there exist attributes that are "easy to control" in the sense that almost all directives influence them. A reasonable feature selection algorithm would perhaps ignore such attributes when choosing directives because they are easy to control anyhow. Third, we observed that if an attribute is not easy to control, it has a relatively small number (if any) of directives that are associated with relatively high influence values. These observations motivated the main assumption of this algorithm: the distribution over influence values for (A_i, Z_j), $j = 1, \ldots, n$ should be almost uniform over low values and very low for high values (if any exist). Denote by $IM_{i,j}$ the value of influence between the ith attribute and the jth directive. If we sort these values for a fixed i (namely, a certain attribute), we expect to see one of the following behaviors:

1. All values are low.
2. Almost all values are relatively high.
3. Most values are low, there are few high values (less than 20%), and the difference between the high and low values is very sharp.

Algorithm FS_Greedy (Algorithm 2) identifies high influence values, based on this assumption. It favors directives that have high information values for several attributes. The algorithm takes into account only the quality of the directives alone and does not look at the "coverage" of the attributes; therefore, there may be attributes covered by more than one directive, and others that are not covered by any chosen directive.

```
FS_Greedy(IM, Budget)
  Input  : IM - Influence Matrix
           Budget - Number of directives to choose
  Output: S - Set of directives to be used
  // Initialize
  Ẑ ⇐ ∅
  α = 0.5
  while (|Ẑ| < budget)and(α > 0.25) do
    for i ← 1toM do
      Sort the influence values IM_{i,j}, j ∈ {1,...,N} in
      descending order: v_1, v_2, ..., v_N
      Compute the ratios v_1/v_2, v_2/v_3, ..., v_{N-1}/v_N
      if ∃k, s.t. v_k/v_{k+1} > α · (v_1 − v_N) and k < N/5 then
        // v_1, ..., v_k are significantly high
        Z^i ← the set of directives that have significantly
        high mutual information with the i'th attribute
      else
        Z^i ← ∅
      end
    end
    Ẑ = Z^1 ∪ Z^2 ∪ ... Z^M
    if |Ẑ| < budget then
      α_new = 0.95 · α_old
    end
  end
  For each Z_j ∈ Ẑ calculate its score
  Where score Z_j = Σ_k 1{Z_j ∈ Z^k} (number of occurrences)
  Return S, the Budget directives with highest scores
end
```

Algorithm 2: Feature selection greedy algorithm

6.2.2 Scoring directives

The most simple method to score directives would be to count their occurrences in Z^i, as defined in Algorithm FS_Greedy. However, instead of taking into account only the fact that a directive has an influence over an attribute, we can consider a weighted version that takes into account the level of influence. For this purpose we need to define a scoring function that gets as input the level of influence of a certain directive on all attributes and returns a score for the quality of the directive. Denote by v the vector of values of corresponding to level of influence of a directive on all the attributes. $v_i = 0$ if this directive has no influence on the ith attribute (namely directive $\notin Z^i$). We used one of the following options:

- Count: $\text{score}(v) = \sum_{i:v_i>0} 1$
- Sum: $\text{score}(v) = \sum_i v_i$
- Square sum: $\text{score}(v) = \sum_i v_i^2$
- Square root sum: $\text{score}(v) = \sum_i \sqrt{v_i}$

The difference between the scoring options is the score given when there are few strong values compared to many medium values. When counting, there is bias toward directives that have many values larger than zero that are not necessarily high, while summing over values biases toward directives that can have high values associated with a small number of directives. There is no answer to the question of which is preferable, hence we use several subsets, each given as the output of a different scoring function.

6.2.3 Attribute coverage oriented algorithm

FS_Greedy algorithm does not take into account considerations of the following form:

- It can be better to prefer a directive that has a lower influence value with a certain attribute, if this attribute is not affected by any other directive.
- It can be better to prefer directives that have medium influence on several attributes, over a directive that has a high influence value with a single attribute.

Observations over the results of FS_Greedy led us to the conclusion that such consideration needs to be accounted for. To overcome the weaknesses of the previous algorithm, we suggest a new set selection method called FS_Cover (see Algorithm 3). Algorithm FS_Cover finds a good subset, taking into account coverage of attributes by directives. Algorithm FS_Greedy is first used to choose three directives that are used as the initial set. Next, we iterate until a budget is reached as follows: at every iteration, we consider adding each of the remaining directives. We score the current set with the new directive, and choose the directive that leads to the highest score. The score depends on rows of a matrix that is composed only of current set columns, and takes into account the mean and standard deviation values of the mutual information values. The pseudo code (including the setScore algorithm) is presented in Algorithm 3.

```
FS_Cover(IM, Budget)
  Input  : IM - Influence Matrix
           Budget - Number of directives to choose
  Output: S - Set of directives to be used
  // Initialize
  S ← FS_Greedy(IM, 3)
  if |S| > 3 then Randomly choose 3 members
  S_left ← {Z_i | 1 ≤ i ≤ N, Z_i ∉ S}
  α ← 0.5
  while (|S| < budget) and (α > 0.25) do
    for Z_k ∈ S_left do
      score(Z_k) ← SetScore(IM, S ∪ {Z_k})
    end
    k_1 ← arg max_k score(Z_k)
    k_2 ← arg max_{k≠k_1} score(Z_k)
    k_3 ← arg max_{k≠k_1,k_2} score(Z_k)
    Randomly choose k' from the triplet k_1, k_2, k_3
    S ← {S ∪ {Z_k'}}
    S_left ← S_left − {Z_k'}
  end
  return S
end
SetScore(IM, S)
  Input  : IM - Influence Matrix
           S - subset of directives
  Output: Score of the given subset
  M ← IM(:, S) // sub matrix composed of the
      directives in the input set
  w ← Σ_j M(:, j) // sum over the directives
  μ ← mean(w)
  σ ← std(w)
  score ← μ / max{σ, ε} // ε > 0 is a small
      predefined constant (handle cases of
      σ → 0)
  return score
end
```

Algorithm 3: Feature selection cover algorithm

6.3 Experimental results

We now present the results of the feature selection procedure for the CDG process. The experiment was conducted on the z10 IFU environment described in Sect. 4. Figure 6 presents the log of influence values that were estimated for the problem. The values are presented in logarithmic scale for clarity of presentation. Dark cells in the matrix correspond to low influence and light cells correspond to high influence. Each row corresponds to an attribute and each column corresponds to a directive. The chosen directives are marked with ellipses. These results correspond to using the Algorithm FS_Cover with a budget request of five directives. We can clearly see that the chosen directives have very high values over several attributes, and that except for attribute 8, which has low values with all directives, all attributes are covered with at least one directive. These results overlap with manual feature selection for three out of five directives. As demonstrated in Sect. 8, the automatically chosen directives gave high quality results.

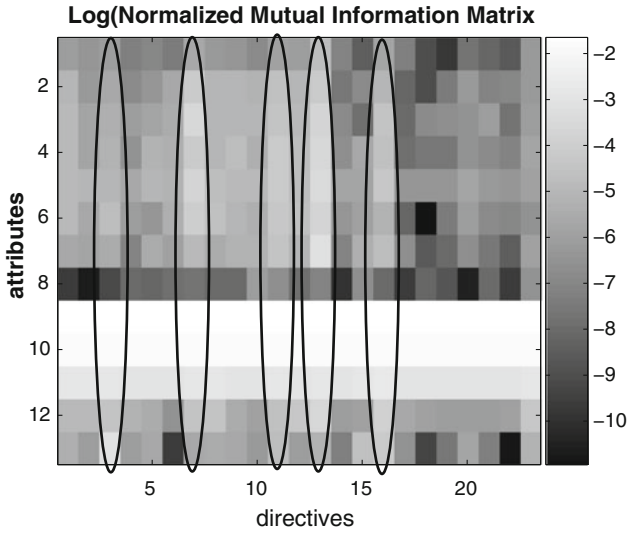


Fig. 6 Normalized Mutual information matrix on real CDG data

7 Structure and parameter learning

In our CDG systems, the relations between the directives and coverage attributes are modeled using a Bayesian network, which is composed of two components: the structure and the parameters. The structure is a DAG that captures causal relations and the parameters represent the conditional probability tables that describe the distribution of a node given its parents.

7.1 Structure learning for CDG

While there are applications where structure learning algorithms provide high quality results (e.g., [10,18]), these algorithms generally have many weaknesses that cause them to be highly inferior to manually constructed structures. Some of these weaknesses are closely related to the unique requirements of Bayesian networks for CDG. One example is the use of soft evidence, which is known to produce a richer set of events on the one hand, but makes structure learning more difficult on the other. Another example is the difficulty constructing networks with hidden nodes, which proved to be essential in past CDG work [3]. Therefore, we do not expect the output of structure learning algorithms to be as good as manually constructed networks. Still, our results show that automatic structure learning can provide structure that is sufficient to boost coverage.

There are two main approaches to structure learning: generic algorithms that can be applied to any problem satisfying predefined conditions and application-based algorithms

that suit the specific setting of given problems (usually with unique domains or characterization). We are not aware of application-specific algorithms that fit the CDG setting; therefore, we use several well known generic algorithms: K2 [11], mcmc [12], gs [13] and structural EM [14]. These algorithms are used in a wide range of applications with various levels of success.

The initial experiments with these algorithms provided reasonable results. However, we wanted to improve the results by taking into account the specific nature of CDG settings. For that purpose, we use two techniques. The first is a post-processing heuristic that prunes arcs between directives in the network graph. Our initial learned networks have arcs between directive nodes, for valid reasons. First, some combinations of assignments to directives cause simulations to fail or produce small sets of coverage events resulting in dependencies between directives in the training set. In addition, the random independent sampling of the directive space for the training set may still contain undesired dependencies. Still, in the CDG settings, these edges are not desirable because we have total control on the directives' settings throughout the process. Moreover, these edges may actually interfere with learning the really important relations between coverage attributes and directives. Therefore, we decided to investigate removing these arcs. The resulting networks, which we refer to as *pruned networks*, yield better results (see Sect. 7.3). The same reasoning cannot be used for edges connecting two attribute nodes because they capture dependencies that are essential for high quality networks. Note that removing the edges between directives can result in directives nodes that are disconnected from the attributes nodes, as indeed happened in our experiments. These orphan directives are no longer part of the CDG system, therefore, the pruning heuristics can be used as a second order feature selection procedure.

The second improvement to the generic learning algorithm is a pre-processing step that modifies the domain of the directives in order to enrich the settings of directives and overcome the soft versus hard evidence problem. Hard evidence is the term we use when we assign a single value to a directive. This is similar to having soft evidence with one probability set to one and all others to zero. In verification, directive nodes are used many times in a single simulation run; in each use, a new value is randomly chosen based on the probability specified. Therefore, hard evidence, which forces a single value to each directive, strongly limits the stimuli sequences generated. Consequently, soft evidence that sets the probabilities for each value in the domain of the directives is essential. However, the structure learning algorithms we use are not capable of handling soft evidence. In order to deal with this, we suggest a heuristic that allows us to incorporate soft evidence in the existing algorithms. We refer to this new method as *quantized directives*.

The main idea of this method is to replace each directive node Y whose domain is $(y_1, \ldots, y_n)$ with node $\tilde{Y}$ whose domain is a discrete set of probabilities over the domain of the original directive Y. For example, if Y has a domain (y_1, y_2, y_3), we can create a node $\tilde{Y}$ with seven values representing the probabilities $\{(1, 0, 0), (0, 1, 0), (0, 0, 1), (\frac{1}{2}, \frac{1}{2}, 0), (\frac{1}{2}, 0, \frac{1}{2}), (0, \frac{1}{2}, \frac{1}{2}), (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})\}$ over (y_1, y_2, y_3). Using this setting allows us to run simulations with soft evidence that enables the full richness of the generated sequences, and use the corresponding value in $\tilde{Y}$ as hard evidence in the structure learning algorithm. Once the structure is learned, we return to the original directives for the parameter learning stage. This quantization procedure is not limited to uniform values over subsets. It can include any soft evidence chosen. However, there is a trade-off between the ability to use many soft evidence values and the domain size of the extended directive. Domain sizes that are too large will cause the learning to be more difficult and lower the ability of the network to perform inference.

7.2 Parameter learning

Once the structure of the Bayesian network is known, the next step is to learn the parameters of its nodes, i.e., the conditional probability of each node given the values of its parents. Unlike structure learning, there exist algorithms for parameter learning that deal with soft evidence [17]. The algorithm we use is the EM (Expectation-Maximization) algorithm [17]. This is an iterative algorithm that is guaranteed to converge to a local maximum of the likelihood. The EM algorithm learns not only the conditional probabilities of internal nodes, but also the prior probabilities of root nodes (nodes with no parents). When such a node corresponds to a directive, calculating the prior probabilities is not needed or even not desirable for the same reasons that led to the pruning heuristic. We are currently investigating how to generalize the learning algorithms to address this issue.

7.3 Experimental results

We tested the ability of four structure learning algorithms to boost coverage over normal activations of the verification environment using its regression suite. The regression suite is a collection of directives settings that are manually designed by the verification team to cover areas of interest. The number of directives used in the regression suite is much larger than the five directives we selected for our experiments or even the initial 23 directives given to us. We started our experiments after about 20,000 simulation runs that covered 5222 events in the model. At that time, we observed a considerable slow down in the coverage rate (see Fig. 7). This was done for two main reasons. First, we wanted to avoid dealing with the very easy-to-cover events that are hit anyway and thus

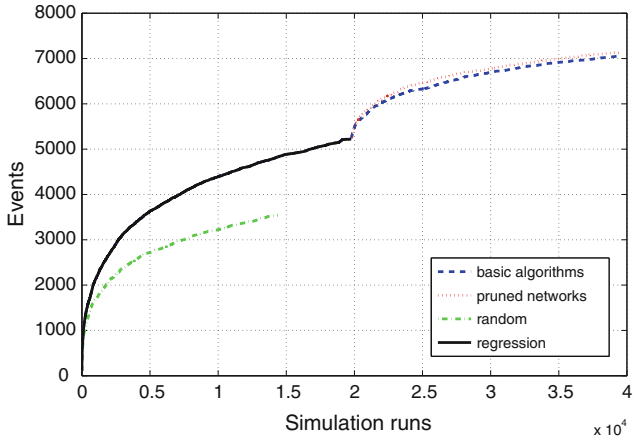


Fig. 7 Coverage progress for random, regression, and aggregated results of basic and prune experiments

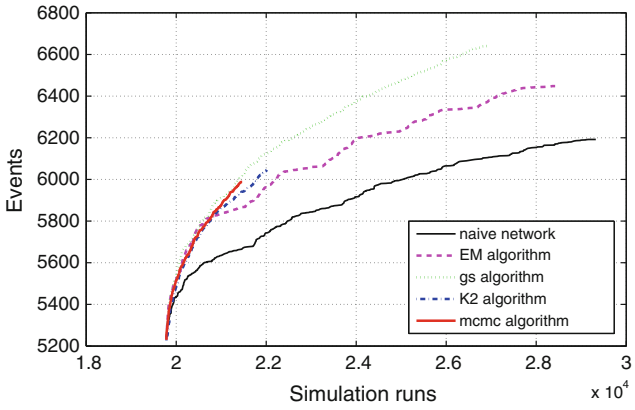


Fig. 8 Coverage progress for the naive network and the four networks in the basic experiment

are not important to the coverage booster. Second, this setting represents a more realistic setting, where the coverage booster is not operated from day one.

We used an additional two references in the experiments. The first was a large set of randomly created directive settings for the five selected directives. This reference was used to verify that the regression suite is not naive and more importantly, to ensure that the feature selection phase alone is not enough for boosting. The last reference was a naive Bayesian network that contained edges between directives and attributes and between pairs of attributes with high mutual information. All the experiments followed the same procedure. We applied a structure learning algorithm and the parameter learning algorithm to obtain a trained Bayesian network. Then we selected 5000 uncovered events and generated (for each network) two sets of directives for each event (using MAP and MPE queries). Finally, we simulated the IFU with the generated directives and measured the corresponding coverage. The networks were not able to produce directives in all the cases, so the total number of simulation runs for each network was smaller than the maximum possible 10,000.

The experiments tested the quality of the four structure learning algorithms mentioned in Sect. 7.1, namely K2 [11], mcmc [12], gs [13] and structural EM [14]. The experiments tested and compared the basic algorithm and the two heuristics proposed in Sect. 7.1. In addition, we tested the ability of the structure learning algorithm to perform feature selection. Specifically, the four experiments we conducted were as follows:

– *Basic algorithm.* The goal of this experiment was to test the ability of the structure learning algorithms in their basic form to produce networks that boost coverage. The results of this experiment were also used as a reference to the heuristics in the following experiments.
– *Pruning.* This experiment was aimed at testing the performance of our proposed pruned networks heuristic. We used the networks of the previous experiment, but removed edges between directives. Removing these edges resulted in directive nodes that are orphans, i.e., not connected to any attribute.
– *Quantization.* The goal of this experiment was to test the ability of the quantization heuristic to overcome the soft evidence problem of the structure learning algorithms. Here, we used a subset of three out of the five original directives. These were chosen because they remained connected to attributes in all the pruned networks of the second experiment (except EM, which already proved to result in sparse networks).
– *Large set of directives.* In this experiment, we added six more directives to the structure learning algorithm, with various levels of correlation. The experiment tested the ability of the structure learning with pruning to perform feature selection by leaving orphans nodes.

Figures 7, 8, 9 and Table 1 summarize the results of the four experiments. Figure 7 shows the coverage progress for the random sampling and the regression suite, and the progress of the aggregated results of the basic and pruning experiments. As expected, the random sampling shows the worst behavior over all our settings. More importantly, the figure shows that both the basic algorithms and their pruning enhancement are able to boost the coverage.

The next figures and table provide more details on the performance of the four basic algorithms and their enhancements. Figure 8 compares the performance of the four basic algorithms. The figure presents the coverage progress of each of these algorithms and the naive network. The figure shows that all four algorithms provide a better coverage rate than the naive network. This confirms the benefits of the structure learning algorithms, even in their basic form. For all four

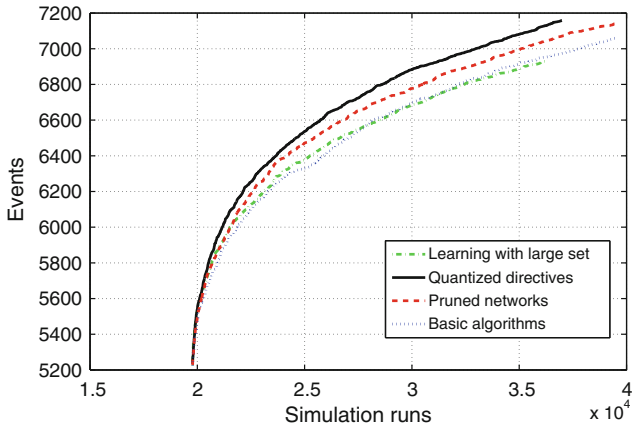


Fig. 9 Coverage progress of aggregated results of all experiments

algorithms, there is some variation in the coverage rate they provide, the number of simulation runs they produced out of the 10,000 possible ones, and the number of new events they cover. These differences are also presented in Table 1. For each algorithm and each experiment, the table shows the number of simulation runs produced and the number of new events covered.

Out of the four algorithms, the gs algorithm provides the best results. It produces a high number of simulation runs and has a high coverage rate. Therefore, although the EM algorithm produces more runs, and K2 and mcmc have a slightly better rate, gs covers the largest number of new events.

The pruning heuristic used in the second experiment not only pruned the edges between directives, it also left some directives orphaned. In fact, one of the directives, which was the weakest among the five selected directives, was removed in three out of the four networks. In general, as Table 1 shows, the pruned networks produced roughly the same number of runs as the networks before the pruning and cover 3–15% more events. The overall improvement of the pruning heuristic over the basic algorithms is also shown in Figs. 7 and 9. The quantization technique allows the structure learning algorithm to use soft evidence. The results of this experiment show that even with the limited amount of softness used in the experiment, the networks were able to produce many more runs (twice as many for K2) and reach many more events (more than 50% for K2). The EM algorithm failed to learn a network in this experiment and therefore did not produce any results.

The attempt to use the structure learning algorithm with pruning for feature selection produced mixed results. On one hand, the pruning removed most of the new directives that are not included in the original set and the directives removed in the second experiment. It also left most of the directives that were not pruned in the pruning experiment. On the other hand, the results of this experiment are worse than the original pruning algorithm. We believe that the "new" directives affect the ability of the structure learning algorithms to capture the relations between the 'good' directives and the coverage attributes. Therefore, to improve the results of this experiment, we will try to learn the structure of the network again after the pruning.

8 Experiments with the full booster

We tested the entire coverage booster flow on the IFU of z10 and the IfuPipe coverage model described in Sect. 4. We started with a set of 23 directives that were considered important for coverage of the model by the unit verification team. During the feature selection phase, the number of directives was reduced to five relevant directives. Specifically, after building the influence matrix using the method described in Algorithm 1, we used the FS_Cover algorithm described in Algorithm 3 to select the best five directives. The results of this phase appear in Sect. 6.3. It is important to note that the directives selected by the automatic feature selection procedure are similar, but not identical to the five directives we selected by hand (using both statistical methods like the automatic procedure and domain knowledge) for the structure learning experiments reported in Sect. 7.3. In fact, out of the five directives, three were selected by both the automatic and manual selection. Moreover, the two directives that were selected by the manual procedure and not the automatic procedure have a score close to the directives that were selected.

Table 1 Summary of experimental results

Network	Basic		Pruning		Quantization		Large set	
	Events	Runs	Events	Runs	Events	Runs	Events	Runs
Naive	970	9,545						
K2	824	2,233	918	2,335	1,374	5,546	833	2,309
gs	1,419	7,150	1,454	6,868	1,553	9,306	1,498	10,723
mcmc	766	1,685	871	1,653	919	2,356	927	3,492
EM	1,226	8,641	1,261	8,916				
Aggregate	1,837	19,709	1,918	19,772	1,936	17,208	1,713	16,524

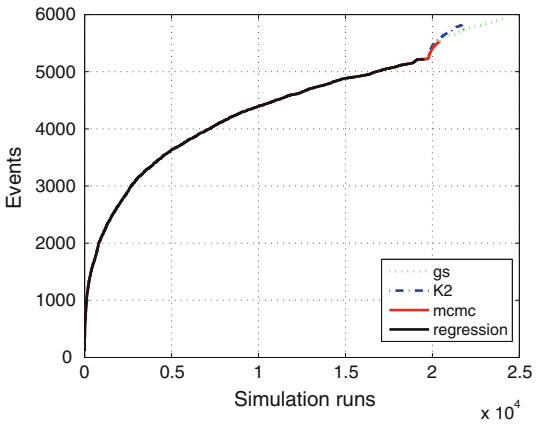


Fig. 10 Coverage progress for the networks in the basic experiment

For the structure learning phase, we used the three useful algorithms we identified in Sect. 7.1, namely, gs, K2, and mcmc. We used each of these algorithms to learn the structure of a Bayesian network and trained each network using the EM algorithm, as described in Sect. 7.2. Then we activated the CDG system containing each trained network and used them to try and cover 3,000 events that are not covered by the unit regression. The results are shown in Fig. 10. The results in the figure are for the basic structure learning algorithms. The results when network pruning is used are similar. The figure shows that each of the networks is able to achieve significant boosting over the unit regression and that there are big differences between the performance of the various networks in terms of the number of directive files they are able to produce and the ability of these directive files to improve coverage. In that sense, the results here are similar to the results of Sect. 7.3.

A close comparison between the results shown here and the results of Sect. 7.3 reveals that while the automatic booster is able to significantly improve the coverage of the unit regression, its performance is not as good the boosting achieved when the manual feature selection was used. There are two main reasons for this difference. First, the automatic feature selection procedure was not able to take advantage of the domain knowledge we used in the manual process. In addition, during the manual selection, we were able to experiment with several sets of directives and fine tune the selected directives. All of this resulted in a manual selected set of directives that is better than the set selected automatically. The second reason for the difference in results are changes in the verification environment and the design itself that occurred between the experiments.

9 Conclusions and future work

Closing the loop from coverage analysis to directives to the stimuli generation, and reaching high coverage, is one of the most difficult and time consuming challenges faced by verification teams. In this paper we presented an automatic technique for constructing a data-driven CDG engine based on Bayesian networks aimed at providing coverage boosting with minimal human effort.

Early results obtained on a coverage model used in the unit verification of an IBM mainframe processor indicate that our technique is able to boost coverage. Namely, it improves coverage in a significant way with minimal human effort, both in the construction and activation of the CDG system. Still, there is a lot of room for improvement and three are issues that we need to address. First, an automatic process needs to determine the quality of subsets of directives in order to choose high quality subsets. Second, we must further investigate the automatic construction of networks with soft evidence. We are currently working on these issues and believe that improving these weaknesses will result in a higher quality booster.

It is clear that the extreme goal, the Holy Grail, is to realize full coverage closure in a totally automatic way without any human effort, even for setting up the automatic scheme. This goal was not yet achieved and we assume that its complexity might cause it to remain elusive in the foreseeable future. Yet, we believe that the coverage booster approach has the potential to bring us closer to this Holy Grail and provide more benefits to the verification process.

Acknowledgments We would like to thank Steve Mittermaier, Matthias D. Heizmann and Hilary Mallar for their advice and work in the IFU pipe model work.

References

1. Wile, B., Goss, J.C., Roesner, W.: Comprehensive Functional Verification: The Complete Industry Cycle. Elsevier, Amsterdam (2005)
2. Piziali, A.: Functional Verification Coverage Measurement and Analysis. Springer, Berlin (2004)
3. Fine, S., Ziv, A.: Coverage directed test generation for functional verification using Bayesian networks. In: Proceedings of the 40th Design Automation Conference, pp. 286–291 (2003)
4. Fournier, L., Ziv, A.: Using virtual coverage to hit hard-to-reach events. In: Proceedings of the 3rd Haifa Verification Conference, pp. 104–119 (2007)
5. Wagner, I., Bertacco, V., Austin, T.: Microprocessor verification via feedback-adjusted Markov models. IEEE Trans. Computer-Aided Des. Integrated Circuits Syst **26**(6), 1126–1138 (2007)
6. Bose, M., Shin, J., Rudnick, E.M., Dukes, T., Abadir, M.: A genetic approach to automatic bias generation for biased random instruction generation. In: Proceedings of the 2001 Congress on Evolutionary Computation CEC2001, pp. 442–448 (2001)
7. Hsiou-Wen, H., Eder, K.: Test directive generation for functional coverage closure using inductive logic programming. In: Proceedings of the High-Level Design Validation and Test Workshop, pp. 11–18 (2006)
8. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference. Morgan Kaufmann, San Francisco (1988)

9. Fine, S., Ziv, A.: Enhancing the control and efficiency of the covering process. In: Proceedings of the High-Level Design Validation and Test Workshop, pp. 96–101 (2003)
10. Fine, S., Freund, A., Jaeger, I., Mansour, Y., Naveh, Y., Ziv, A.: Harnessing machine learning to improve the success rate of stimuli generation. IEEE Trans. Comput. **55**(11), 1344–1355 (2006)
11. Cooper, G.F., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. J. Mach. Learning **9**(4), 309–347 (1992)
12. Laskey, K.B., Myers, J.W.: Population markov chain Monte Carlo. J. Mach. Learning **50**(1–2), 175–196 (2003)
13. Chickering, D.: Optimal structure identification with greedy search. J. Mach. Learning Res. **3**, 507–554 (2002)
14. Friedman., N.: The Bayesian structural EM algorithm. In: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, pp. 129–138 (1998)
15. Ur, S., Yadin, Y.: Micro-architecture coverage directed generation of test programs. In: Proceedings of the 36th Design Automation Conference, pp. 175–180 (1999)
16. Cowell, R.G., Dawid, A.P., Lauritzen, S.L., Spiegelhalter, D.J.: Probabilistic Networks and Expert Systems. Springer, Berlin (1999)
17. Heckerman, D.: A tutorial on learning with Bayesian networks, Technical report, Microsoft Research. Redmond, Washington (1996)
18. Rusakov, D., Geiger, D.: Asymptotic model selection for naive Bayesian networks. J. Mach. Learning Res. **6**, 1–35 (2005)
19. http://en.wikipedia.org/wiki/IBM_z6
20. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learning Res. **3**, 1157–1182 (2003)
21. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artif. Intell. **97**(1-2), 273–324 (1997)
22. Jolliffe, I.T.: Principal Component Analysis. Springer Series in Statistics. Springer, Berlin (2002)
23. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley, New York (1991)