J Electron Test (2016) 32:125–136
DOI 10.1007/s10836-016-5577-1

Exemplar-based Failure Triage for Regression Design Debugging

Zissis Poulos[1] · **Andreas Veneris**[1]

Received: 30 July 2015 / Accepted: 23 February 2016 / Published online: 19 March 2016

Abstract Modern regression verification often exposes myriads of failures at the pre-silicon stage. Typically, these failures need to be properly grouped into bins, which then have to be distributed to engineers for detailed analysis. The above process is coined as failure triage, and is nowadays increasing in complexity, as the size of both design logic and verification environment continues to grow. However, it remains a predominantly manual process that can prolong the debug cycle and jeopardize time-sensitive design milestones. In this paper, we propose an exemplar-based data-mining formulation of failure triage that efficiently automates both failure grouping and bin distribution. The proposed framework maps failures as data points, applies an affinity-propagation (AP) clustering algorithm, and operates in both metric and non-metric spaces, offering complete flexibility and significant user control over the process. Experimental results show that the proposed approach groups related failures together with 87 % accuracy on the average, and improves bin distribution accuracy by 21 % over existing methods.

Keywords Failure triage · Design debugging · Regression verification · Satisfiability · Cluster analysis

Responsible Editor: L. M. Bolzani Poehls

✉ Zissis Poulos
zpoulos@eecg.toronto.edu

Andreas Veneris
veneris@eecg.toronto.edu

[1] Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, ON M5S 3G4, Canada

1 Introduction

Functional verification poses a major bottleneck in modern design cycles [4]. It becomes even more complex when performed in regression mode during early design stages, where thousands of input vectors are applied to heavily exercise both standard and corner-case functionality. The co-existence of potentially multiple design errors at this stage results in hundreds of error traces being exposed. Design debugging, which accounts for more than 60 % of the verification effort [4], is the task that locates design errors using information provided by these traces. It does so by using formal tools, which take as input a single error trace and automatically determine possible error sources (suspects) in the RTL [3, 7, 13, 14]. These are finally examined by the engineer to track down the exact error, a process known as detailed debug [11].

However, this classical debugging approach has two major drawbacks. First, it targets each failure in isolation. As a result, different engineers may spend unnecessary resources performing detailed debug for failures that originate from the same RTL error. Second, it does not identify failures that should be high in priority for detailed debug. As such, it cannot determine the most appropriate engineer to further analyze each error trace. This uncertainty often creates confusion, with error traces constantly circulating until they are placed to the right queue for analysis.

To break this uncertainty there is a need for a preprocessing step that properly categorizes failures and assigns them to the best-suited engineer(s) for analysis. This preprocessing step is referred to as *failure triage*, and it consists of two main tasks. First, failure binning is performed. Its goal is to determine correlations between the exposed failures and bin together these failures that are likely to be caused by the same design error. The second step, failure bin distribution,

identifies these failures that should be prioritized within each bin. Next, it assigns each bin to the engineer(s) most familiar with these high-priority failures.

Traditionally, the above steps are performed manually in the industry. Typically, engineering teams employ primitive forms of debug, such as simple error (*i.e.,* log) messages from end-to-end "golden-model" checkers, exception checkers and various assertions [11]. Often a verification engineer is dedicated to the task of constantly monitoring error logs and empirically deciding how to pass failures to developer teams. In other cases, a script is used to parse these error logs and allocates failures following a rule-based strategy. Such *ad-hoc* manual practices convey limited information and often fail to identify correlations between traces.

Following the successful paradigm of data mining algorithms in the verification domain [15], engineers are now turning their attention to that field as a means to address the verification pain of failure triage. Along these lines, recent works have automated failure triage by formulating it as a clustering problem [9, 10]. To perform failure binning through clustering, the authors in [9] first map failures as data points into a metric space (Euclidean), whereas in [10] failures are implicitly mapped into a non-metric space. A limitation of these frameworks is that clustering can only be performed by algorithms that operate exclusively in metric or non-metric spaces, respectively. Most importantly, though, these formulations focus in failure binning and do not offer efficient solutions for failure bin distribution.

To overcome these drawbacks, in this paper, triage is viewed as an exemplar-based clustering problem. With the proposed formulation, not only is failure binning automated, but failure bin distribution is also properly addressed. This is because exemplar-based clustering not only partitions the failure set accordingly, but also algorithmically identifies these failures that are representative of other failures in each bin. These failures-exemplars are then naturally considered as ones of high priority for detailed debug. Moreover, the algorithm that is applied, Affinity Propagation (AP) [5], operates in both metric and non-metric spaces, and thus leverages the merits of both representations, unlike prior art.

Experiments on four industrial designs show that the proposed work achieves 87 % average accuracy for failure binning and improves bin distribution accuracy by 21 %, on the average, against existing methods.

The remainder of this paper is organized as follows. Section 2 discusses prior work in failure triage for design debugging. Section 3 describes the proposed formulation and presents the exemplar-based clustering process. Finally, Section 4 discusses experiments and Section 5 concludes the paper.

2 Preliminaries and Prior Art

2.1 Failure Binning

Consider an erroneous design with a single or multiple errors in the RTL that undergoes regression testing. We say that a failure occurs, when a mismatch between the expected "golden" value(s) (0,1 or X for unknown) and the observed one(s) is identified at some observation point (primary output, probed internal signal or the output of an assertion). Suppose that at the end of regression testing, N design failures are exposed, denoted $\mathbf{F} = \{F_1, F_2, \ldots, F_N\}$. In this work we target scenarios where verification is done via logic simulation and potentially returns error traces, where each error trace is a sequence of stimuli exposing a failure. Thus, it is hereby assumed that for each failure its corresponding error trace is also given. This paper does not address debugging instances where no error trace is available, such as the debugging of unreachable states [1].

The goal of failure binning is to produce a complete partition of the failure set $\mathbf{F}$ into K disjoint clusters. Ideally, failures that are caused by the same RTL error are placed into the same cluster, and into distinct clusters otherwise. These clusters (or groups) of failures are then assigned to engineers. They, in turn, target each group for root-cause analysis and eventually perform a fix that can potentially remove all failures belonging to the same cluster. A high-level view of the process and the scope of each stage are illustrated in Fig. 1.

For the process to be accurate, two key points have to be properly addressed. First, pairwise failure similarity needs to be quantified based on the above desired relationship, and second, a "good" number of clusters, K, needs to be selected.

The quality of the required metrics above heavily relies on the availability and type of data that a triage tool collects

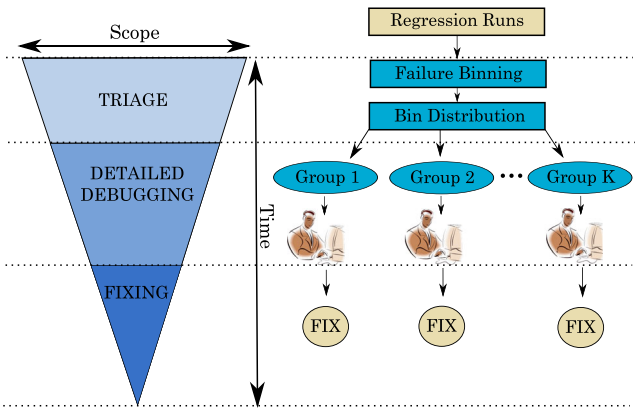


Fig. 1 A modern design debugging flow

from regression runs. Especially concerning failure similarities, recent work in [10] and in [9] has shown that data collected from SAT-based debuggers and logic simulators can generate proper failure similarities.

Precisely, in [9, 10] a baseline SAT-based debugging pass is first executed, and, for each failure F_i, the automated debugger outputs a set of design components (RTL blocks or modules), denoted as $S_i = \{s_1, s_2, \ldots, s_{|S_i|}\}$. Components $s_1, s_2, \ldots, s_{|S_i|}$ are referred to as *suspects*, and include all possible design components that can be responsible for the observed failure. Although automated debuggers generally can perform the task at the gate-level (high resolution debugging), in this work we focus on RTL-level debugging, which is where state-of-the-art debuggers operate. As such, a suspect can be an *always* block, an *if* statement, a module definition or instantiation etc.. Due to its exhaustive nature, SAT-based debugging guarantees that the design location responsible for some failure F_i will be included in suspect set S_i. In this context, suspect set S_i can be viewed as a "signature" that characterizes failure F_i.

Example 1 To demonstrate the above concepts consider an error trace, as depicted in Fig. 2. In that figure we show the sequential behavior of the circuit for that trace using its Iterative Logic Array (ILA) representation [14]. In more detail, an error at component s_2 is excited in cycle $m-2$ and propagates to cause a failure (F_1) at an observation point in cycle m. The generated error trace of length m is then passed to an automated debugger. The result is a suspect set $\mathcal{S}_1 = \{s_1, s_2, s_3\}$ of design components that can explain the wrong output. Suspects s_1, s_2 and s_3, excited in cycles k, $m-2$ and $m-1$ respectively, along with their propagation paths are illustrated in Fig. 2. Note that the erroneous component is included in the set S_1 as suspect s_2. For illustration purposes, suspects that correspond to the responsible design error are shown by a solid circle (suspect s_2 in this case), whereas suspects that can explain the failure but are not actual erroneous components are shown by dotted circles (suspects s_1 and s_3 in this example).

As previously discussed, each suspect component provides some guidance to the general error location related to each failure. However, some of the suspect locations (i.e. reset signals, primary inputs, dangling logic, bit-flips etc.) can explain the failure but may be irrelevant to the erroneous module or signal responsible for it. Thus, all collected suspect components need to be appropriately weighted to quantify the likelihood of being an actual design error.

Ideally, we need to identify and promote suspects that exhibit behavior similar to that of typical design errors. Recent work has experimentally shown that there are two properties often observed in such suspect components. The first is temporal proximity to the observed failure [12]. That is, typical design errors are expected to be excited only a few cycles before the failure is observed, since they can quickly propagate to observation points in most cases. Second, these locations are expected to exhibit low toggling frequency measured between consecutive excitation cycles [9]. The argument behind the latter is that typical RTL errors are relatively "easy" to excite in the majority of cases.

Once all suspects are collected, these two criteria are used to assign various levels of significance to each suspect component with respect to the failure it may be responsible for. The work in [10] does so via a suspect ranking scheme, while the work in [9] adopts a data weighting scheme.

In [10], pairwise similarity between failures F_i and F_j, denoted $s(i, j)$, is computed as a weighted version of the Jaccard Index [2]. Particularly, $s(i, j)$, is given as:

$$s(i,j) = -\left(1 - \frac{|S_i \cap S_j|}{|S_i \cup S_j|}\right) \times \pi_{ij} \quad (1)$$

In Eq. 1, the factor $\left(1 - \frac{|S_i \cap S_j|}{|S_i \cup S_j|}\right)$ quantifies mutuality between the suspect sets of F_i and F_j, while π_{ij} is a measure of discrepancy between the ranks of mutual suspects in these sets [10]. The product is negated to abide to similarity semantics. As it becomes apparent, the similarities generated by Eq. 1 do not respect the triangle inequality, and thus this method operates in a non-metric space.

On the other hand, the authors in [9] use a feature-based representation for verification failures. Specifically, if $s_1, s_2, \ldots, s_M$ are all the distinct suspect components in $\bigcup_{i=1}^{N} S_i$, then failure F_i is represented by a real-valued feature vector $\vec{F}_i = [x_1^i, x_2^i, \ldots, x_M^i]$, where each feature x_j^i obtains the weight (significance) of suspect s_j with respect to failure F_i, if it appears in S_i, or takes the value 0

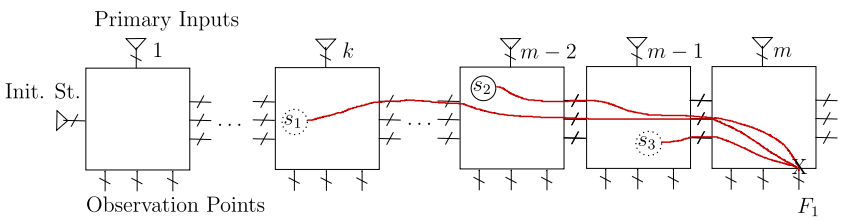


Fig. 2 Error trace and suspect components

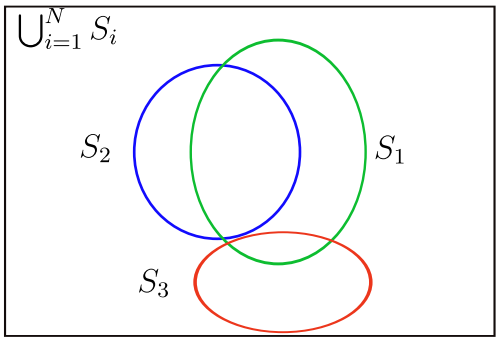


Fig. 3 Suspect set overlap

otherwise. Specifically, a feature is a scalar quantifying some attribute of a suspect component, such as its temporal proximity to the failing observation point and the toggle rate of its input signals [9]. Failures are then mapped into a metric space, where similarity $s(i, j)$ is defined as the negated Euclidean distance between $\mathbf{F_i}$ and $\mathbf{F_j}$:

$$s(i, j) = -||\mathbf{F_i} - \mathbf{F_j}|| \quad (2)$$

Figure 3 illustrates a hypothetical example of three failures F_1, F_2, F_3. Suppose that the corresponding suspect sets S_1, S_2 and S_3 overlap as shown in Fig. 3. Failures, such as F_1 and F_2, that have suspect sets with proportionally large overlap are expected to be strongly related and vice versa. Of course, in [10] suspect ranks adjust the contribution of this overlap accordingly, based on Eq. 1. In [9] the contribution of the overlap and the suspect weights is implicitly represented into a metric space, as shown in Fig. 4. Note that pairwise similarities $s(i, j)$ are non-positive real values. In both cases, the larger $s(i, j)$ is, the stronger the relation between F_i and F_j is considered to be.

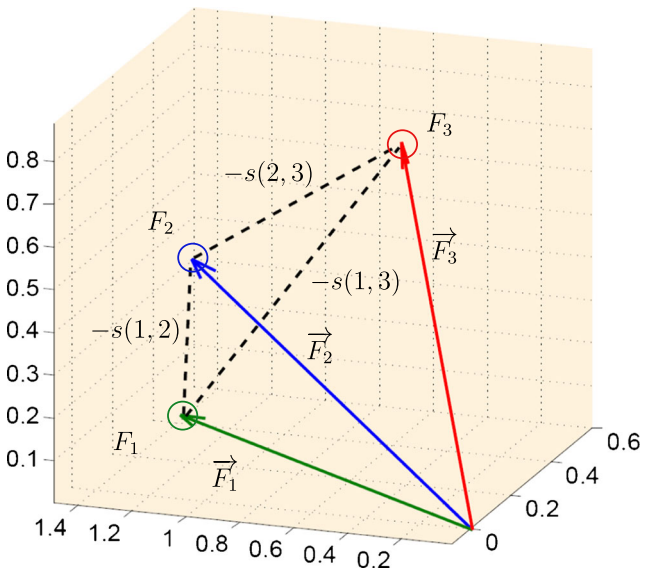


Fig. 4 Failures mapped into a metric space

Once failure similarity is determined, failure binning is performed through clustering algorithms. However, depending on whether similarities are metric or non-metric only specific classes of algorithms can be applied. The framework in [10] is limited to connectivity-based greedy hierarchical clustering, while in [9] hierarchical clustering is combined with k-means to produce refined failure partitions. In both cases, the number of clusters, K, is "guessed" empirically. In [10] it is based on expected cluster size/density, while in [9] the authors use a threshold applied on the clustering merge cost. These estimates experimentally appear to be the bottleneck in failure binning accuracy for both of the methodologies.

2.2 Failure Bin Distribution

Suppose that failure binning generates K failure clusters, $C_1, C_2, \ldots, C_K$. The goal of failure bin distribution is to allocate each of the K clusters to engineers that are most familiar with failures within that cluster. In past work this allocation is done as follows. For each cluster C_i, suspects across failures in C_i that have a high average rank (or average weight) are identified. Then, cluster C_i is passed to the engineer(s) that are best-suited to analyze these important suspect locations in the design. These suspects essentially correspond to a data point that is close to the centroid (mean) of cluster C_i. However, it is not necessary that a failure in cluster C_i always matches with the cluster mean. In fact, this event is quite rare. Along these lines, it is naturally more suitable to allocate the cluster based on a data point associated with a failure that appears in C_i, rather than a fictional data point close to the cluster mean.

3 Exemplar-based Failure Triage

To overcome the problem of heuristically selecting the number of clusters and to distribute failure bins based on failures that belong to the partitioned set, we formulate triage as an exemplar-based clustering problem. In what follows, we provide the details of our methodology.

3.1 Data Preparation

Before triage commences, it is necessary to collect all the relevant information for each failure generated by regression. To this end, we follow the standard approach of performing SAT-based debugging, and for each failure F_i we generate a suspect set S_i.

If N is the the number of observed failures, then $|\bigcup_{i=1}^{N} S_i| = M$ gives the number of distinct suspects across all failures $F_1, F_2, \ldots, F_N$. Recall that in a feature-based representation, M corresponds to the number of dimensions

of the metric space where failures are mapped. In previous work, feature-based representation has been shown to outperform other existing methods, but it does not perform well when $M >> N$ [9]. This is due to the "curse of dimensionality" when the number of dimensions is much larger than the size of the data set. As such, to determine whether to use a metric space mapping we first compute the ratio N/M and check if $N/M > \gamma$, where $\gamma \leq 0.2$. If $N/M > \gamma$, then similarities $s(i, j)$ are computed based on Eq. 2 in a metric space. Otherwise, similarities are non-metric and are computed based on Eq. 1. However, in both cases, if $S_i \cap S_j = \emptyset$, then $s(i, j)$ is set to $-\infty$, since disjoint suspect sets indicate that failures should have minimum similarity and never be placed into the same cluster. Note that the range of potential values for threshold γ is determined empirically with a bias towards smaller values to avoid using metric mappings for high-dimensions. However, the actual value that we use in our framework is the result of a training process as it will be discussed in Section 4. In both cases, pairwise failure similarities are given in the form of a $N \times N$ similarity matrix $\mathbf{S}$.

3.2 Problem Formulation

Once similarity matrix $\mathbf{S}$ is computed, failure binning takes place. However, unlike prior work, in our methodology we do not treat failure binning separately from bin distribution. Rather, we provide solutions to both problems simultaneously, in a unified sense, by formulating the whole process as exemplar-based clustering.

Exemplar-based clustering not only partitions the data, but also identifies for each cluster its most representative member, also called *exemplar*. A cluster exemplar is the member of the cluster that exhibits maximum overall similarity to other members in the cluster. In the context of failure triage a cluster exemplar can be viewed as a failure that is representative of the erroneous behaviour associated with all other failures that belong to the cluster. Intuitively, this failure-exemplar along with its suspect locations can efficiently determine how to distribute the failure bin.

To find solutions under this formulation we apply an algorithm known as Affinity Propagation (AP) [5], which is derived as an instance of max-product loopy belief propagation [2]. The AP algorithm avoids an explicit search for exactly K clusters and allows for a trade-off between the number of clusters and the within-cluster similarity that is obtained. As such, our technique does not require the number of clusters to be specified or "guessed" a priori. However, the algorithm allows the engineer to specify failures that are believed to be of high importance. To this end, a quantity called *preference*, denoted p_i, is associated with each failure F_i and quantifies our expectation that some failures are more suitable to be exemplars than others. The higher the preference p_i, the more likely failure F_i is to be an exemplar, and vice versa. Preferences are provided as an input to the algorithm in the form of a N-dimensional vector $\mathbf{p} = [p_1, p_2, \ldots, p_N]$. They correspond to values assigned to the $s(i, i)$ similarities (self-similarities) in the diagonal of the similarity matrix, such that $[s(1, 1), s(2, 2), \ldots, s(N, N)] = \mathbf{p}$.

The objective function of exemplar-based clustering, and thus of the AP algorithm, is to maximize the sum of all similarities between data points in the cluster to their exemplar, while also maximizing the total preferences. Suppose the algorithm takes $\mathbf{S}$ and $\mathbf{p}$ as input. Then, a set of N^2 binary random variables $h_{ij} \in \{0, 1\}$ is defined, such that $h_{ij} = 1$ if and only if failure F_i has chosen F_j as its exemplar. Note that $h_{jj} = 1$ indicates that F_j is, in fact, an exemplar. Finally, recall that $s(j, j) = p_j, \forall j \in \{1 \ldots N\}$. The objective function is formulated as a constrained optimization problem, as follows:

$$\max_{\{h_{ij}\}} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} s(i, j) h_{ij} \tag{3a}$$

subject to

$$\sum_j h_{ij} = 1 \qquad \forall i \tag{3b}$$

$$h_{jj} = \max_i h_{ij} \quad \forall j \tag{3c}$$

Equation 3b ensures that each point chooses exactly one other point as its exemplar. Equation 3c guarantees that an exemplar is never assigned to another exemplar. The goal of the AP algorithm is to find settings of $\{h_{ij}\}$ that maximize the quantity in Eq. 3a. The algorithm finds solutions based on an iterative message-passing procedure [5] and, upon convergence, it outputs a set of exemplar failures denoted as $\mathcal{F}_{ex}$:

$$\mathcal{F}_{ex} = \{F_j \in \mathbf{F} : h_{jj} = 1\} \tag{4}$$

Each exemplar defines exactly one cluster of failures, and is itself a member of the cluster. Thus, the number of clusters, K, is equal to the number of exemplars in $\mathcal{F}_{ex}$:

$$K = |\mathcal{F}_{ex}| \tag{5}$$

On the other hand, for each non-exemplar failure F_i there exists an exemplar failure F_j which is the most similar to F_i across all other exemplars in $\mathcal{F}_{ex}$. The set of non-exemplars that are most similar to exemplar F_j, denoted $\mathcal{F}^j_{nex}$ is:

$$\mathcal{F}^j_{nex} = \{F_i \in \mathbf{F} : i = \arg \max_{F_j \in \mathcal{F}_{ex}} s(i, j)\} \tag{6}$$

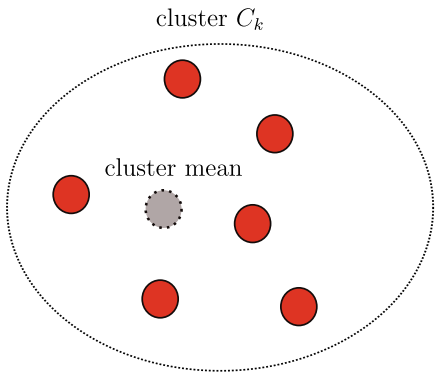
Each non-exemplar failure is then assigned to the same cluster C_k as its most similar exemplar failure F_j. If exemplar $F_j \in C_k$, then:

$$C_k = \{F_j\} \cup \mathcal{F}^j_{nex} \tag{7}$$

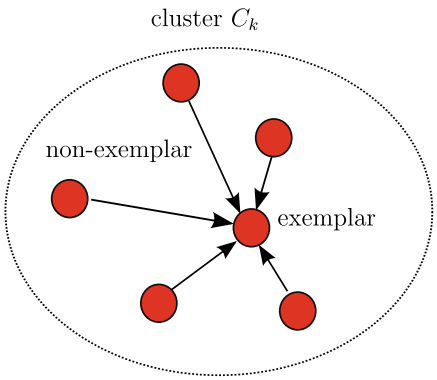
Finally, for the bin distribution step, cluster C_k is assigned to the engineer that is responsible for the suspect locations of failure F_j, where F_j is the exemplar for cluster C_k. This set of design locations is given as:

$$\{s_i : s_i \in S_j \wedge F_j \in C_k \wedge F_j \in \mathcal{F}_{ex}\} \tag{8}$$

Based on the above, the benefits of this formulation are several. First, the process does not make any assumptions about similarities, apart from the fact that they are non-positive real values. They can be either metric or non-metric with no consequences to the formulation. Therefore the process can seamlessly replace existing binning algorithms irrespective of how similarities are generated, and it can leverage the merits of both representations. Further, the number of clusters, K, rises algorithmically from the message-passing procedure, and does not need to be "guessed" beforehand. Finally, as illustrated in Fig. 5, bin distribution is now guided by suspects that correspond to failures-exemplars included in the data set (Fig. 5b), rather than by suspects that correspond to a data point at the cluster mean (Fig. 5a). This allows engineers to analyze each exemplar failure (error trace) and its corresponding suspect set as a whole, instead of examining suspect locations in isolation, even if these locations are significant for a particular cluster.



(a) Standard cluster formation



(b) Exemplar-based formation

Fig. 5 Failure cluster formation in traditional vs. proposed triage

3.3 Triage with Prior Belief

Another important benefit of the proposed methodology is that it offers significant flexibility to the engineer considering various triage scenarios. In the majority of cases, before triage commences it is rather difficult to have an estimate on the number of design errors (number of clusters) responsible for failure set **F**. However, there are cases where engineers based on their intuition can target specific failures around which they wish **F** to be partitioned. That is, failures that are believed should serve as exemplars of erroneous behavior. Along these lines, the proposed method allows triage to be executed with prior belief, both in a uniform and non-uniform setting, as discussed below.

3.3.1 Uniform Setting

In the uniform setting no assumptions are made regarding to what extent a failure should serve as an exemplar. This translates into a triage scenario where even intuitive knowledge around the importance of each failure is missing. In the proposed formulation, this is encoded by simply setting all preferences $p_i \in \mathbf{p}$ into some constant non-positive real number. In practice, the AP algorithm performs as expected and quickly achieves convergence when preferences are fixed to the mean of all similarities:

$$p_k = \frac{1}{N^2} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} s(i, j), \quad k \in \{1 \ldots N\} \tag{9}$$

3.3.2 Non-uniform Setting

In the non-uniform setting the engineer selects specific failures to promote as exemplars a priori. If failure F_k is targeted then p_k is set to 0. Otherwise preference p_k is set to the median of similarities as in Eq. 9. Promoting specific failures as exemplars by setting higher preferences affects the number of clusters to be formed, but this number also emerges from the message-passing process. Therefore, it is not necessary that the number of clusters formed at the end will match then number of promoted failures, if this number does not reflect a reasonable partition based on the constrained optimization problem that is solved. Still, if the "guess" is close to reality then the AP algorithm can be effectively guided. Finally, note that this feature is not offered by any of the existing methodologies. These methods do allow a selection for K before the process begins, but that does not imply that **F** is eventually partitioned around the targeted failures.

3.4 Overall Flow

A flow diagram of failure triage, as it is formulated in this work, is illustrated in Fig. 6. It should be emphasized that the SAT-based debugging step that provides the "signature" suspect sets is performed in the flow whether failure triage takes place or not. Triage begins immediately after this step and preprocesses the data before detailed debug commences, where these suspect sets need to be further examined. As such, this debug step is not added by our methodology but is an inherent part of the overall debug flow in regression mode.

4 Experimental Results

This Section presents experimental results for the proposed triage framework. All experiments are conducted on a single core of an Intel Core i5 3.1 GHz workstation with 8GB of RAM. Four *OpenCores* [8] designs are used for the evaluation (`vga`, `fpu`, `spi` and `mem_ctrl`). The SAT-based debugger used to extract suspect locations is implemented based on [14]. A platform coded in Python is developed to parse debugging and simulation data, calculate the appropriate failure similarities and cluster the failure set through the AP algorithm. For each design, a set of different errors is injected each time by modifying the RTL description. The types of the injected RTL errors resemble typical human-introduced errors (missing pipeline stages, incorrect read pointers, bad stimulus etc.) that lead to non-trivial triage scenarios. In total, twenty regression tests are run, generating various numbers of failures each time, caused by a different set of errors.

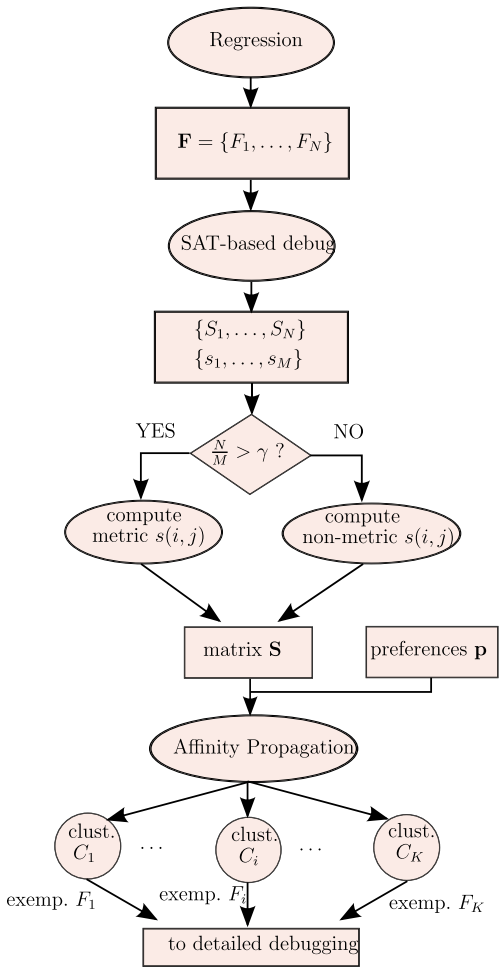


Fig. 6 Proposed triage flow

For each design, a pre-generated set of test sequences is used that is stored in vector files. Each regression run involves hundreds to thousands of input vectors. For the purpose of capturing failures we use end-to-end "golden model" checkers that compare the expected value for various operations, exception checkers and various assertions throughout the designs.

Table 1 summarizes benchmark information and statistics per regression run. From left to right, columns show the circuit name and number of gates, an enumeration for regression runs, the number of input vectors, the number of simultaneous RTL errors, the number of observed failures (N), the number of distinct suspect components (M) generated by SAT-based debugging, and finally the size of the similarity matrix per regression run. SAT-based debugging is performed at the RTL-level and not at the gate-level. This is to avoid large numbers of suspects that would cause the number of dimensions to increase dramatically. Another reason that we maintain this lower debugging resolution is because gate-level suspects offer less insight for coarse-grain debugging and are usually targeted during detailed debugging at later stages [11]. Finally, for evaluation purposes triage is run under the uniform setting described in Section 3.3.1. That is, we assume that there are no targeted failures. This allows us to conduct a fair comparison against existing methods.

To evaluate failure binning accuracy, we use the `Rand Index` (`R.I.`) measure [2]. This metric compares the estimated clustering against a reference failure binning, with the latter corresponding to an ideal partition where all failures are grouped with 100 % accuracy. The metric ranges from 0 to 1 and represents the fraction of correct clustering decisions. Accuracy is hence measured as $100 \times$ `R.I.` %. This fraction is given as $R.I. = \frac{TP+TN}{TP+TN+FP+FN}$, where TP (true positives) corresponds to the number of failure pairs that are caused by the same error and are placed in the same cluster in the binning step, TN (true negatives) is the number of failure pairs that are caused by different errors and are placed in different clusters, FP (false positives) indicates the number of failure pairs that are caused by different errors, but are erroneously placed in the same

Table 1 Benchmarks and regression statistics

Ckt. (# gates)	Test No.	# vectors	# errors	$\|\mathbf{F}\|$ (N)	$\|\bigcup_{i=1}^{N} S_i\|$ (M)	# Matrix entries ($N \times M$)
	1	25206	4	45	36	1620
	2	25206	7	62	40	2480
vga	3	25206	8	97	61	5917
(72292)	4	31870	10	106	129	13674
	5	31870	13	121	155	18755
	6	17365	3	19	28	532
	7	17365	7	30	152	4560
fpu	8	20094	7	55	74	4070
(83303)	9	41759	9	83	60	4980
	10	41759	11	125	111	13875
	11	4573	3	13	38	494
	12	4573	5	28	46	1288
spi	13	4573	6	51	82	4182
(1724)	14	5019	8	39	196	7644
	15	5019	9	72	113	8136
	16	10834	3	17	24	408
	17	10834	5	32	45	1440
mem_ctrl	18	10834	7	31	29	899
(46767)	19	13370	8	66	94	6204
	20	13370	11	95	137	13015

cluster, and finally FN (false negatives) corresponds to the number of failure pairs that are caused by the same error, but are erroneously placed in different clusters.

Further, for all regression runs we fix threshold γ to 0.2. The above value is selected across a range of values in [0.1, 1.0] by conducting 2-fold cross-validation on a training dataset (15 regression runs) for the same circuits using the same regression suites. Table 2 offers some statistics for the training set. From left to right, columns contain the circuit name, the number of RTL errors injected, the number of failures and distinct suspects. These numbers are given as a range (min - max). RTL errors for the training set were injected randomly in the Verilog source. This type of training allows us to maintain large training and test sets, while using each regression run both for training and validation purposes. The resulting selection ($\gamma = 0.2$) offers a stable behavior in the triage flow; we generally desire metric representations and only disallow them in corner-cases when high-dimensionality becomes an impediment in performance. We report that the highest binning accuracy for the validation set is 91 % and is achieved with $\gamma = 0.2$ and $\gamma = 0.3$. The lowest one observed is 79 % when $\gamma = 1.0$, which corresponds to an extensive use of non-metric representations.

Figure 7 shows a comparison between the proposed engine and the methods described in [10] (`non-metric`) and [9] (`metric`), in terms of failure binning accuracy. The proposed triage engine outperforms the framework in [10] in 14/20 regression runs, while it achieves better accuracy in 7/20 cases compared to the engine in [9]. Precisely, across all regression runs, the proposed engine achieves 87 % clustering accuracy on average, compared to 77 % and 88 % accuracy of the non-metric and metric methods, respectively. Note that for corner-cases where $N/M < \gamma$, particularly for test-cases 7 and 14, the proposed method generates non-metric similarities. This has a

Table 2 Training set statistics

Ckt.	# err.	(N)	(M)
vga	3 – 6	23 – 71	16 – 98
fpu	4 – 5	13 – 52	37 – 114
spi	3 – 5	9 – 50	29 – 76
mem_ctrl	3 – 7	10 – 42	21 – 123

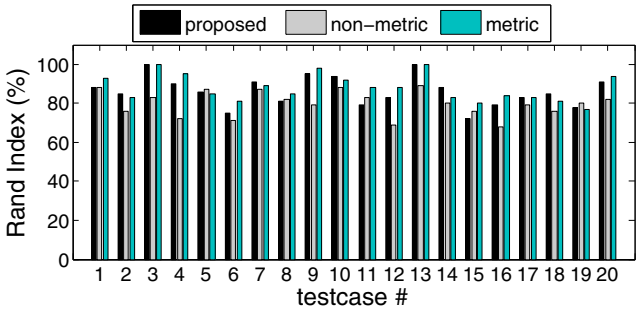


Fig. 7 Engine accuracy vs. existing methods

positive impact on accuracy by avoiding effects of high-dimensionality. The metric approach in these cases exhibits lower accuracy as it also shown by Fig. 7. Finally, it is worth mentioning that under the non-uniform setting the proposed method can achieve up to 96 % average accuracy when targeted failures are carefully chosen. What such high accuracy implies is that the presence of an experienced verification engineer with the insight to select proper preferences (as presented in Section 3.3.2) can lead to very low misclassification rates. Moreover, even when a small number of failures are misplaced and discovered during detailed debugging, the verification engineer has enough context to make a new decision. This is because misplaced failures that have gone through the triage process are not returned as abstract objects. They are instead returned as context-rich entities that include all possible suspect locations in order of significance. In a similar way that such structures can guide detailed debugging they can also guide decisions to rectify any failure misplacement, effectively reducing the manual effort required.

To illustrate how accurate the number of generated clusters, K, is in our framework, Fig. 8 shows how far this prediction is from the number of design errors responsible for the observed failures. We refer to the latter as "true clusters". The prediction error is then given as (K − # true clusters) for all test-cases. In ideal cases, K is equal to

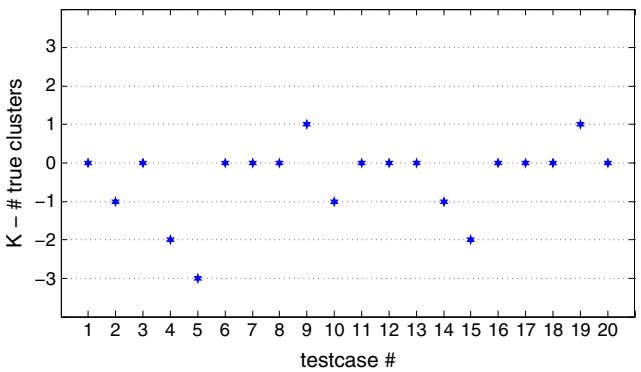


Fig. 8 Cluster prediction error

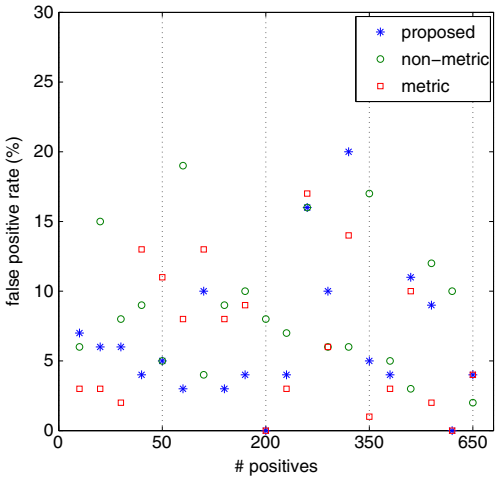


Fig. 9 False positive rates

the number of design errors and the prediction error is 0. Figure 8 shows that in 12/20 regression runs the AP algorithm achieves a perfect prediction, which greatly boosts binning accuracy. Interestingly, in the rest of cases, the number of formed clusters is usually smaller (by 1 to 3 clusters). Only in test-cases 9 and 19 the number of clusters is larger than necessary.

Finally, Figs. 9 and 10 illustrate a comparison between the proposed engine and the methods in [9, 10] in terms of the false positive and false negative rates that are produced as the complexity of binning increases (expressed as the number of positive failure pairs). In the context of failure triage, both false positives and false negatives can prolong time-to-debug. False positives will often be identified when subsequent regression runs expose failures belonging to some failure bin that has been previously marked as fixed. However, they can affect binning quality and bin

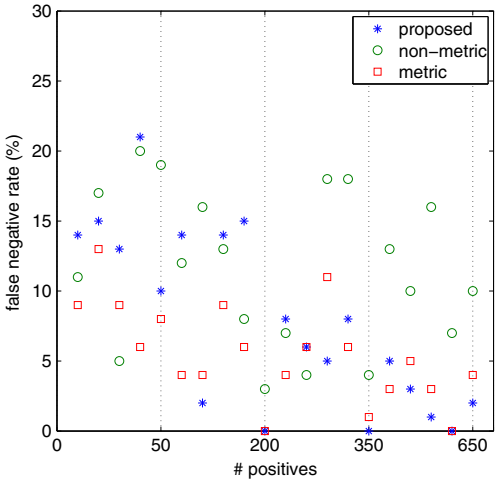


Fig. 10 False negative rates

distribution as they introduce noise in each failure bin. False negatives, on the other hand, can lead to more complex scenarios. A large false negative rate indicates that multiple failures are erroneously allocated across various bins. This can introduce confusion between engineering teams, and it also deteriorates bin quality by producing bins with sparse (missing) information.

As one can observe in Fig. 9, the proposed methodology produces false positives rates similar to those in previous work. This rate does not seem to be clearly affected by the problem complexity. Particularly, the proposed method generates failure binnings with a 6.55 % false positive rate (FPR) on the average, as opposed to 8.85 % average FPR and 6.5 % average FPR produced by the non-metric and metric methods, respectively. Furthermore, in Fig. 10 we observe that the proposed triage engine clearly outperforms the non-metric method in terms of false negative rate (FNR), and is on par with the metric method. Specifically, the average FNR in our method is 7.70 %, whereas the one in [10] is 11.55 % and the one in [9] is 5.75 %. Interestingly, both the proposed method and the metric approach produce smaller FNR as the complexity of the problem increases. This is indicative that a feature-based representation for failures is more effective when failure clusters are dense, leading to even more robust and accurate failure binning steps. Nevertheless, the relatively low FNR of our method indicates that engineers will be receiving failure bins that mainly contain failures with the same root-cause. Identifying those failures that are misplaced can be achieved by means of re-running tests when a fix is performed to see which failures persist, or by means of suspect intersection as a filtering process to identify such failures before a fix is performed [14]. In any scenario, engineers have available ordered suspect sets to guide their decision, reducing manual effort and uncertainty.

As results indicate, the failure binning step in the proposed triage flow demonstrates high accuracy, comparable to the currently most efficient method in [9]. However, the major strength of the proposed flow is its effective exemplar-based bin distribution step. Since bin distribution in this work is performed via exemplar failures, while existing techniques use high-weight suspect locations to guide the process, we need to use a common reference for comparison purposes. To this end, we identify whether the design error responsible for failures in a particular cluster is included in the suspect set of the exemplar failure. If the design error location is indeed in the suspect list, we obtain its rank or weight (significance), which is already computed to generate similarities in the binning step. Finally, we determine if the error location has a high rank or weight compared to other suspect locations for the same exemplar failure. We do this by sorting suspects in order of decreasing weight. Ideally, the error location resides among the top 10-20 % of suspect locations in the ordered list. In that case the exemplar failure and its suspect locations are effectively prioritized.

Table 3 summarizes experimental results regarding the bin distribution step. Column 1 indicates the regression run number. Columns 2 to 4 show what position the responsible design error takes in the sorted list of suspects that is returned to the engineer by bin distribution in [9, 10], and the proposed flow, respectively. The positions are normalized over the size of the suspect list each time. Thus, when the position is low, then the suspect component that includes the design error appears in the first positions of the list, and vice versa. Each row in the table provides the average design error position per regression run. The fifth column shows the improvement that is achieved by the proposed method compared to the bin distribution approach in [9]. Finally, the last three columns give the mean and standard deviation of cluster sizes per regression run.

One observation from Table 3 is that, on average, the proposed bin distribution approach pushes the responsible design error higher in the list compared to existing methods in 12/20 regression scenarios. In total, the average improvement that is achieved compared to the best of the two existing methods (metric) is approximately 21 % across all regression runs. The last three columns of Table 3 reveal another significant finding. In general, it is expected that errors will be responsible for a varying number of failures, far from uniformly distributed and ideally exposing clusters with significantly different sizes. As shown in Table 3, the metric approach in [9] and our methodology capture the above expectation better than the non-metric approach in [10], since they output clusters of smaller size (mean of 8.41 and 7.76) and with greater variance (average σ of 3.08 and 3.12). The non-metric approach tends to form bigger clusters with similar sizes (mean size 9.4 and average σ 2.54), an observation that can explain its inferior accuracy.

As far as time consumption is concerned, it is vastly dominated by the SAT-based debugging step which is performed whether triage takes place or not. This step consumes from approximately 400 to 7000 seconds per regression test-case. The added overhead due to failure binning and bin distribution is negligible and is in the range of 15 to 50 seconds approximately per regression run.

It becomes clear that the scalability of the proposed framework and its applicability for industrial verification settings greatly depends on the size of the design under test and the number of failures exposed. Efficiency with respect to design complexity is affected by the capacity of SAT-based debugging and its limitations regarding state-space explosion. By utilizing state-of-the art formal debugging tools that leverage abstraction and refinement and trace compaction techniques [6, 12], the proposed triage engine can potentially handle larger designs. Any advances in this class of tools can have an immediate positive effect on the

Table 3 Bin distribution performance

Test	avg. normalized error position			Improvement	Cluster size (mean / σ)		
No	Non-metric	Metric	Proposed	(%)	Non-metric	Metric	Proposed
1	0.22	0.13	**0.10**	23	15 / 4.4	11.2 / 7.3	11.2 / 7.3
2	0.26	0.27	**0.16**	41	10.3 / 1.3	10.3 / 2.9	10.3 / 3.3
3	0.19	0.18	0.20	−11	12.1 / 9.3	12.1 / 9.3	12.1 / 9.2
4	0.31	0.34	**0.27**	20	15.3 / 5.5	13.3 / 8.6	13.3 / 8.0
5	0.40	0.18	0.20	−11	17.3 / 2.3	12.1 / 5.0	12.1 / 4.9
6	0.34	0.35	**0.16**	54	6.3 / 4.0	6.3 / 1.2	6.3 / 1.2
7	0.24	0.17	**0.10**	41	6.0 / 3.9	4.3 / 2.6	4.3 / 4.1
8	0.18	0.12	0.19	−58	9.2 / 1.0	7.9 / 2.8	7.9 / 1.4
9	0.27	0.26	**0.23**	12	9.2 / 4.0	8.3 / 4.7	8.3 / 4.4
10	0.51	0.48	**0.32**	33	12.5 / 9.9	13.9 / 13.2	12.5 / 13.1
11	0.30	0.27	0.29	−7	4.3 / 3.5	4.3 / 2.1	4.3 / 1.2
12	0.18	0.23	**0.08**	65	7.0 / 2.9	7.0 / 5.5	5.6 / 5.5
13	0.17	0.15	0.15	0	10.2 / 7.0	8.5 / 7.5	8.5 / 7.5
14	0.33	0.24	**0.16**	33	5.6 / 1.1	4.9 / 1.6	5.6 / 2.0
15	0.21	0.19	0.26	63	12.0 / 2.2	10.3 / 4.7	10.3 / 2.8
16	0.46	0.32	0.33	−3	8.5 / 2.1	5.7 / 4.0	5.7 / 4.5
17	0.10	0.09	0.09	0	6.4 / 5.4	6.4 / 5.7	6.4 / 5.7
18	0.26	0.14	**0.11**	21	4.4 / 1.7	5.2 / 2.1	4.4 / 2.5
19	0.56	0.41	**0.19**	54	6.8 / 2.7	6.8 / 2.8	6.8 / 2.7
20	0.51	0.46	**0.33**	28	9.7 / 6.6	9.7 / 7.5	8.8 / 7.4
AVG	0.300	0.249	**0.196**	**21**	**9.40 / 2.54**	**8.41 / 3.08**	**7.76 / 3.12**

proposed engine. When it comes to regression scenarios that involve thousands of exposed failures, such as microprocessor testing, the AP algorithm can be efficiently used due to its proven applicability for clustering applications that involve tens of thousands of data points [5].

5 Conclusion

To summarize, this work introduces a novel exemplar-based clustering formulation for the growing problem of failure triage in regression design debugging flows. It proposes the use of Affinity Propagation to simultaneously provide solutions to failure binning and bin distribution as a unified constrained optimization problem. Experimental results demonstrate the applicability and efficiency of the proposed triage engine, and indicate that it outperforms existing methods for the important step of bin distribution.

References

1. Berryhill R, Veneris A (2015) Automated rectification methodologies to functional state-space unreachability. In: Proc. Design, automation and test in Europe, pp 1401–1406
2. Bishop CM (2007) Pattern recognition and machine learning (Information Science and Statistics). Springer
3. Chang KH, Wagner I, Bertacco V, Markov IL (2007) Automatic error diagnosis and correction for rtl designs. In: Proceedings International High Level Design Validation and Test Workshop (HLDVT), pp 65–72
4. Foster H (2011) From volume to velocity: the transforming landscape in function verification. In: Proc. Design and Verification Conf
5. Frey BJ, Dueck D (2007) Clustering by passing messages between data points. Science 315:972–976
6. Keng B, Veneris A (2012) Path directed abstraction and refinement in sat-based design debugging. In: Proc. Design Automation Conf
7. Mirzaeian S, Zheng F, Cheng K (2008) Rtl error diagnosis using a word-level sat-solver. In: Proceedings of IEEE international test conference, pp 1–8
8. OpenCores (2007). http://www.opencores.org
9. Poulos Z, Veneris A (2014) Clustering-based failure triage for rtl regression debugging. In: Proceedings IEEE international test conference, pp 1–10
10. Poulos Z, Yang Y, Veneris A (2014) Simulation and satisifiability guided counter-example triage for rtl design debugging. In: Proceedings of IEEE international symposium on quality electronic design, pp 394–399
11. Safarpour S, Keng B, Yang YS, Qin E (2012) Failure triage: the neglected debugging problem. In: Proc. Design and verification conference
12. Safarpour S, Veneris A, Najm F (2010) Managing verification error traces with bounded model debugging. In: Proc. ASP design automation conference, pp 601–606

13. Sarbishei O, Tabandeh M, Alizadeh B, Fujita M (2009) A formal approach for debugging arithmetic circuits. IEEE Trans CAD 28:742–754
14. Smith A, Veneris A, Ali MF, Viglas A (2005) Fault diagnosis and logic debugging using Boolean satisfiability. IEEE Trans CAD 24(10):1606–1621
15. Vasudevan S, Sheridan D, Patel S, Tcheng D, Tuohy B, Johnson D (2010) Goldmine: automatic assertion generation using data mining and static analysis. In: Proc. Design, automation and test in Europe, pp 626–629

Zissis Poulos received the B.E. degree in Electrical and Computer Engineering from the National Technical University of Athens in 2011, and the M.A.Sc degree in Electrical and Computer Engineering from the University of Toronto in 2014. He is currently a Ph.D. candidate at the University of Toronto in the Department of Electrical and Computer Engineering. His research is on formal verification and automated design debugging of digital systems, and extends to applications of data-mining for statistical design debugging.

Andreas Veneris received a Diploma in Computer Engineering and Informatics from the University of Patras in 1991, an M.S. degree in Computer Science from the University of Southern California, Los Angeles in 1992 and a Ph.D. degree in Computer Science from the University of Illinois at Urbana Champaign in 1998. In 1998 he was a visiting faculty at the University of Illinois until 1999 when he joined the Department of Electrical and Computer Engineering and the Depart- ment of Computer Science at the University of Toronto where today he is a Professor. His research interests include CAD for debugging, verification, synthesis and test of digital circuits/systems, and combinatorics. He has received several teaching awards and a best paper award. He is the author of one book and he holds several patents. He is a member of IEEE, ACM, AMC, AAAS, Technical Chamber of Greece, Professionals Engineers of Ontario and The Planetary Society.