Journal of Hardware and Systems Security (2018) 2:201–213
https://doi.org/10.1007/s41635-018-0043-4

The Old Frontier of Reverse Engineering: Netlist Partitioning

Travis Meade[1] · Kaveh Shamsi[2] · Thao Le[3] · Jia Di[3] · Shaojie Zhang[1] · Yier Jin[2]

Received: 11 February 2018 / Accepted: 13 July 2018 / Published online: 10 September 2018

Abstract
Without access to high-level details of commercialized integrated circuits (IC), it might be impossible to find potential design flaws or limiting use cases. To assist in high-level recovery, many IC reverse engineering solutions have been proposed. This paper focuses on a hard problem facing reverse engineering researchers, that of netlist partitioning. To assist in this endeavor, we propose our own methods that focus on signal matching by analyzing fan-in trees. This analysis extends to representing signal's fan-ins numerically by their structural properties. These values go through certain common dimension reducing algorithms; clustering practices are also leveraged to assist in our proposed partitioning process. Adversely researchers have almost never agreed on the metric for evaluating such netlist partitioning methods. To keep our results unbiased, we leverage the Normalize Mutual Information (NMI) to evaluate our proposed partitioning method and compare its results with other techniques that aim to solve the same problem. Lastly, we show how our proposed methods are capable of effectively partition netlists of larger scale than previously proposed schemes.

Keywords Reverse engineering · Gate-level netlist · Hardware Trojan · Structure analysis

1 Introduction

A potentially dangerous reliance on third party resources has as of recent been largely fueled by a crucial need for lower fabrication costs and smaller time-to-market (TTM). This reliance and the ease of which a hardware Trojan can be inserted into a netlist has facilitated an untrust that has grown to epic proportions between intellectual property (IP) vendors and consumers.

✉ Yier Jin
yier.jin@ece.ufl.edu

Travis Meade
travis.meade@ucf.edu

Kaveh Shamsi
kshamsi@ufl.edu

Thao Le
tpl001@mail.uark.edu

Jia Di
jdi@uark.edu

Shaojie Zhang
shzhang@cs.ucf.edu

1 University of Central Florida, Orlando, FL 32816, USA

2 University of Florida, Gainesville, FL 32611, USA

3 University of Arkansas, Fayetteville, AR 72701, USA

Many methods for Trojan detection or removal has been proposed [1, 2], and [3]. Researchers have shown that some of these methods are ineffective at complete prevention and detection [4]. Other Trojan defenses require high level information about the circuit that might not be available to all IP users. Thus, many researchers have moved towards weaker defense models. At the heart of the problem of Trojan detection lies a need for full function recovery. Without complete IC comprehension, it would be impossible to say for certain that an IC is Trojan free.

However, full function recovery has proved over the past few decades to be a hard problem; the amount of research is evidence of this [5–8]. Papers have shown the possibility for accurate function identification on the module level, which can easily allow for full function recovery, but the research community lacks an accurate method for "netlist partitioning"[1] that would allow identification methods to be leveraged [11]. Even just high-level full function recovery can allow any IC producers to more easily spot potential back doors and hardware Trojans.

[1]The netlist partitioning mentioned in the paper involves breaking the signals of a netlist into smaller disjoint subsets that can represent either different words, modules, or even IPs. It should not be confused with the optimization problem presented in [9] or [10].

For the aforementioned hardware Trojan detection capabilities, this paper aims to pin-down an elusive sub-problem for reverse engineering researchers focused on solving full function recover. We plan to assist in laying the ground work for a standardized method for evaluating chip annotation while focusing on the netlist partitioning problem. Past research on netlist partitioning has suffered from two major problems,

- A lack of clarity regarding the problem itself;
- Although a plethora of research has been done in the same thread as netlist partitioning, researchers have either failed to openly and directly define their goals or have not created an appropriate method for evaluating the accuracy of their method.

Upon these challenges, we will re-investigate this "old" problem of netlist partitioning. In addition, we will try to develop metrics for gauging partitioning methods by reintroducing previously developed clustering techniques now for the sake of reverse engineering.

The main contributions of this paper are listed as follows:

- We re-think the definition of netlist partitioning taking into consideration the desire for reverse engineering netlists, by demonstrating how state-of-the-art solutions can fail under certain scenarios.
- We present several heuristic-based approaches for netlist partitioning which aims to address the weaknesses associated with state-of-the-art partitioning methods.
- We demonstrate a more effective method for evaluating netlist word-level partitions and utilize it to provide effective partitioning assessment.

The rest of this paper has the following structure. In Section 2, we discuss precursors to, state-of-the-art methods of, and methods reliant upon, netlist partitioning. We discuss how certain state-of-the-art methods are not capable of, nor have properly displayed the potential of, accurately partitioning netlists. We discuss the reasoning behind our methods and the reason for selecting our evaluation metrics in Section 3. Section 4 presents our various partitioning methods at varying levels of detail and discusses the evaluation metric in greater depth. The results are presented in Section 5. Section 6 discusses many possibilities an extracted netlist provides and concludes the paper with a summary of what was found in this paper.

2 Related Works

A major factor that spurred early development for reverse engineering methods was the threat of hardware Trojans. Many early hardware Trojans were inserted into FSMs at the RTL, which could allow, with certain input patterns, behavior not within the original IC's specification or side-channel activity that could leak sensitive information. Trojan detection methods in the form of logic structures identification were proposed. An exemplary solution used the topology and the knowledge of control signals to help cluster signals into words [12]. The method of [12] was originally designed with the intention of extracting FSM words. Since many Trojans emulate such FSMs, a simple extension to the paper would be to check the effect of such words. The authors determined the method's effectiveness by considering the distribution of the sizes of the generated words. The method showed promise, in that many of the words generated were small, which allows users to quickly determine their functionality. However, the method's accuracy was evaluated in any way which brings into question its ability to correctly group wires. Due to the method's simplicity and justifications, a similar approach is implemented and used for comparison in this paper.

Other works have used information from input/output words to stitch together the data paths that make up the netlist, for example WordRev [13]. Using an idea called forward and backward propagation and a modest signal comparison, signal pairs were checked and merged into large word sets. After which by leveraging function identification methods, a rough high-level IC design was constructed. For analysis, a topology comparison was done manually, and although the number of words found between the different netlists varied the authors claimed the topology was the same across each benchmark. The authors mentioned that in one of the optimization parameters for synthesizing the netlist, only 4-bits of the 6-bit words were found propagated by their method. The authors conjectured the missing bits meant the method was susceptible to changes in optimization parameters.

Some other, less elegant approaches have also used known structures to perform IC slice identification. In [7], the authors examined netlist slices that contained 6 inputs. Using a permutation invariant function matcher, the found bit-slices were grouped into equivalence classes. Then using these classes, components were merged through two major methods: either by common signals, such as ones that could control data flow; or by signal propagation, like those found in adders. The authors also used a QBF solver and a set of known circuits to identify parts that behave in an equivalent manner. The author's methods were evaluated by considering circuit coverage. Perhaps the biggest critique of this method was the failure to show how much of the coverage was correct.

One other notable work used hash of trees to determine if two signals are similar enough to belong to the same word [14]. The method was very simplistic; to their credit their method was much quicker than trying to match via a graph isomorphism. The authors also provided a way to measure their method's accuracy by examining the percentage of full words and fragments found. This can be misleading, and we feel that a more appropriate metric for analyzing clusters should be used. It is also unknown whether the group took into account over grouping, as inappropriately merging two words could have a negative impact overall on a partitioning method.

The methods mentioned above all used some form of netlist partitioning, each with the goal of high-level netlist analysis. Other methods have been proposed to split netlist into smaller parts for identifying where circuit components might be borrowing logic from. Such methods have tried to, with unsupervised learning techniques, broadly classify components based on their graph structure. A recent example of unsupervised circuit labelings is [15]. Their benchmarks were composed of both differing IP and the same IP synthesized via different methods. What the authors found was that there was some IP overlap between circuit component clusters. The authors suggested that the overlap meant either pieces were identified across IPs or there were errors. These "rogue segments" were not thoroughly examined in the paper. The results, although not allowing necessarily for matching each part to the appropriate function, could help detect the presence of known hardware Trojan structures. The major weakness of this method was that novel or unknown structures could be easily mislabeled, especially when learning is done on a small, misrepresented set of netlists.

3 Motivation

Large SoCs can be easily composed of many different IPs, and even worse each IP could have many different modules creating very large, complex structures, which would be difficult to reverse engineer as a whole. To reduce the required effort for analyzing circuits a different approach can be taken. Rather than attempting to determine the IC's functionality all at once, researchers, such as those that authored WordRev [13], try to analyze pieces of the netlist and then after figuring out the components try to determine the full picture. To begin this process, it is required that accurately or at least meaningfully breaking a circuit into pieces can be done. As mentioned previously, there are many abstractions and layers that can compose a large SoC, as can be seen by Fig. 1. Partitioning can be attempted at each of these levels and function/module matching could be performed for each of these resulting partitions. This paper will focus on word-level partitioning in gate-level netlists. To address the issues plaguing other word partitioning methods mentioned in Section 2, we develop a formal procedure based on well-documented clustering techniques that leverages the exact word-level information of the original netlist. That is the resulting partition of our methods will be compared to the original design's intended word sets.

As shown earlier, there exist a number of ways to numerically evaluate a partitioning method. However, many methods introduce bias that allows certain possibly flawed methods to appear to work well. For example, counting the number of complete words found can be very misleading. If words A and B are found by merging all bits of A and B, we could argue that A and B, were found. However, information of the words' separation gets lost by this partition, so although by the metric the method might look efficient, the partitioning method does not work perfectly. In a similar line of thought, we need to be aware that the coarse and fineness of the ground truths structure should be taken into consideration when evaluating a method at partitioning.

Due to the vast differences in hierarchical structure between ICs, it becomes difficult to develop an evaluation that does not favor certain methods. To prevent this bias, the method of normalized mutual information (NMI) is leveraged, which although has roots deep within information theory domain, has been shown to be a modest method for evaluating clustering schemes regardless of the entropy of the ground-truth [16].

Another difficulty associated with evaluation of such partitioning methods is that of multi-interpretation of ground truths. When working with the HDL, or a similar high-level language, the ground truth partitioning becomes even more difficult to infer due to a potential reliance on semantics that could change the ground truth without

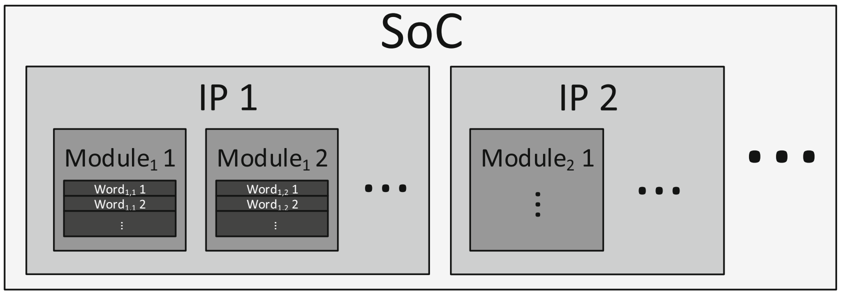

Fig. 1 Simplified hierarchical view of an SoC

changing the observed netlist. Barring the potential for multi-interpretation, we can assume that there will be only one unique ground truth partition per gate-level netlist.

The last potential problem pertaining to netlist partitioning covered in this paper is that of multi-membership. Such a situation occurs when a signal is shared between words. A simple example could be generated by circuit reduction using the circuit's synthesis tool. It is a possibility that in a large circuit redundant signals could be merged or removed to improve IC performance. Multiple membership could also occur when, based on certain control signals, a wire has different behavior. Probably the worst situation for multi-membership to appear is when there is a mistake in the high-level code that somehow gets propagated through to the resulting gate-level netlist.

In short, the most prevalent problems for researchers involved in netlist partitioning are:

– Meaningfully evaluating the effectiveness of a partitioning scheme such that one method can be compared to others;
– Handling ground truths where words can overlap;
– Deciding the correct ground truths where multiple possible outcomes could be considered correct.

4 Methodology

In this section, we formalize our set of methods for clustering. We also present the chosen method for evaluating partitioning solutions.

4.1 Logic Identification Metrics for Partitioning - REWIND

In previous work, a scoring method was leveraged to distinguish logic verse data by using a recursive comparison of fan-in trees. The proposed similarity function, f, generated a score for pairs of signals that estimated the matching substructures (RELIC) [17]. These resulting similarity scores fell in the range of 0 to 1 inclusive, where higher values meant a more similar structure. Originally, the score was used to create an overall similarity score by summing all the scores generated from comparing the original wire with each other wire from the netlist. Logic wires, assumed to be outliers, were selected based on the lowest summations found, since logic should, unlike data signals, have unique structures, the sum of logic signal score should be low. There was a claim that these scores could also be used to find data words as well.

This paper proposes a simple method for retooling the original scores for netlist partitioning. The method we see fit for partitioning uses a near pairwise comparison of signals, while utilizing a representative signal to seed words. A pairwise comparison is the most straightforward and obvious usage, especially since RELIC's original paper leveraged a pairwise comparison for detecting logic. However, direct pairwise comparisons can be quite slow. The work flow for the proposed partitioning scheme, which we called Reverse Engineering Word Identification (REWIND), is as follows. A signal that has not been grouped to a word yet will be selected at random as a seed signal. For each signal within some specified threshold of the seed signal, that has not already been added to a word, we will add it to a current word. After word seeding and growth, the current word is added to the set of known words. Since sometimes a signal might store the negation of the word and involve the logical complement pin of a register, we will allow comparison to be within some negation of the original signal. The described process can be seen in Algorithm 1.

Algorithm 1 Determine the word sets of a netlist with a set of signals S, given a similarity score threshold, t, a similarity score depth d, and a similarity score function f.

```
1:  function PairwiseSimScore($S, t, d$)
2:      $words \leftarrow \emptyset$
3:      $seen \leftarrow \emptyset$
4:      for Random $x \in S \wedge x \notin seen$ do
5:          $seen \leftarrow seen \cup \{x\}$
6:          $X \leftarrow \{x\}$
7:          for $y \in S \wedge y \notin seen$ do
8:              if $(f(x, y, d, \mathbb{T}) > t \vee f(x, y, d, \mathbb{F}) > t)$ then
9:                  $seen \leftarrow seen \cup \{y\}$
10:                 $X \leftarrow X \cup \{y\}$
11:             end if
12:         end for
13:         $words \leftarrow words \cup X$
14:     end for
15:     return $words$
16: end function
```

4.2 Bus Recovery via Similarity Score - REBUS

A direct follow-up to the original logic classification paper was a method that leveraged data flow to extract the data bus of a netlist (REBUS) [17]. The method does not present formal results. The method used the same scores generated by the logic classification method, and along with the concept of forward propagation in [13], the follow-up method tried to extract the data path in a netlist. The method could, due to the reduced number of comparisons, have a significantly better runtime than that of the original logic classification method. This paper compares the results of the bus-based method by examining both the time and accuracy of the method on various types of netlists.

Aside from the standard inputs required from the original logic classification method, the bus-based method needs a set of input words to seed the word set. An extension to the method could utilize a set of output words to potentially find correct word pairs starting from the output with the use of backwards propagation. Regardless the method adds in the pairs of known word signals to a queue. While there are unresolved pairs within the queue, the proposed bus-based partitioning method will try to find new word pairs. For forward propagation, the fan-outs are examined from the known word pair. Each pair of fan-out wires is selected, and if the found fan-out signals do not currently belong to the same word, the score between the two signals is evaluated. If the resulting score is above certain threshold, the signals have their words merged. Any new word signal pairs created will be added to the queue, and evaluation will continue. The pseudo-code for the process can be seen in Algorithm 2.

Algorithm 2 Determine the word sets of a netlist with a set of signals S, given a similarity score threshold, t, a similarity score depth d, a set of input words W, and a similarity score function f.

```
1: function BusBasedPartition($S, t, d, W$)
2:     $words \leftarrow W \cup \{\{x\} | \forall w \in W (x \in S \wedge x \notin w)\}$
3:     $q \leftarrow \emptyset$
4:     for $w \in W$ do
5:         for $x, y \in w \wedge x \neq y$ do
6:             $q.append((x, y))$
7:         end for
8:     end for
9:     for $pair \in q$ do
10:        $x \leftarrow pair.first$
11:        $y \leftarrow pair.second$
12:        for $x_o \in fanout(x)$ do
13:            for $y_o \in fanout(y)$ do
14:                if $(f(x_o, y_o, d, \mathbb{T}) > t \vee f(x_o, y_o, d, \mathbb{F}) > t)$ then
15:                    $X \leftarrow X \cup \{y\}$
16:                    $w_x \leftarrow (w \in words \wedge x_o \in w)$
17:                    $w_y \leftarrow (w \in words \wedge y_o \in w)$
18:                    for $x_1 \in w_x$ do
19:                        for $y_1 \in w_y$ do
20:                            $q.append((x_1, y_1))$
21:                        end for
22:                    end for
23:                    $words \leftarrow words \backslash \{w_x\}$
24:                    $words \leftarrow words \backslash \{w_y\}$
25:                    $words \leftarrow words \cup \{w_x \cup w_y\}$
26:                end if
27:            end for
28:        end for
29:    end for
30:    return $words$
31: end function
```

4.3 Principal Component Analysis based Partitioning - REPCA

A previous approach used for finding possible similarities between signals within words used comparison of signal graph information [7]. A major detriment to such methods is the redundancy of certain graph or structural information. This redundancy is caused by the fact that synthesization is typically performed by a deterministic protocol that optimizes a netlist structure. The synthesis process has a high chance of incorporating similar structure types due to a user's desire to optimize for some design parameters, because of the described automation process, the variance in structure across several variables might be poor.

In [7], dimension/information reduction was simply done by leveraging graph dot products. This reduction could allow a more accurate matching by eliminating extraneous information that could lead to a misclassification while still preserving the potentially distinguishing information. However, for our work, we plan to use a more commonly used method. One of the most common statistical method for dimension reduction is principal component analysis (PCA). The flexibility of such a technique led to the development of Reverse Engineering Word sets using PCA (REPCA)

Each signals' initial numerical information for the PCA based netlist partitioner was derived via certain structural data, examples of which are fan-in set sizes and fan-out set sizes both at various depths in the gate-level netlist. It is possible that the same signal can belong to multiple sets (see Fig. 2 for an example). Other fields consisted of gate type (OR, AND, XOR), and temporal logic type (Flip-Flop or not). We also kept information regarding the number of clock cycles for primary input to affect the gate and the number of clock cycles for the gate to affect a primary output. Other netlist distances leveraged include the closest flip-flop in the fan-in tree and closest flip-flop in the fan-out tree, both of which are measured by the number of non-buffer/inverter gates between the signal and selected flip-flop.

After generation of the principle components, the first n_{pca} components are used for comparison of signals. For the comparison, a simple distance metric is used to determine membership, where two points are within the same word, if the euclidean distance between the two points are less than pre-determined cutoff. To determine the appropriate distance used for membership cutoff, a random set of edges are selected. A ratio of edges that are included within words

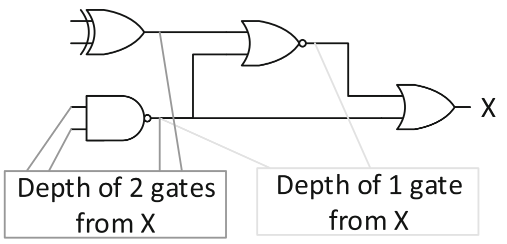

Fig. 2 A simple example of the structural information that can be extracted from a gate-level netlist

compared to the total number of edges is generated by an expected number of words in the final partition. Although it might not be the case, by assuming the inter-word signal distances are always smaller than the intra-word distances and that the number of signals per word are constant, then the ratio can be approximated by the total number of edges divided by the expected number of words.

Based on this expectation and on the desired number of clusters, a distance is selected from the sorted set, which acts as a good edge length cutoff. A sweep is then performed over the set of signals. Like REWIND, when a random signal is found to not be in a word, the signal is used to start a new word. Any signals found to be within a specified estimated distance is then joined to the word. Once each signal has been handled, the word sets are returned. This process is described in Algorithm 3.

Algorithm 3 Determine the word sets of a netlist with a set of signals S, given their principle components, pc, a desired number of words, n_w, a scaling factor α, and a distance metric, d.

```
1: function GetWordSet($S, pc, n_w, \alpha$)
2:     $randDistances \leftarrow \emptyset$
3:     $i \leftarrow 0$
4:     while $i < \alpha \times |S|$ do
5:         $a \leftarrow$Random $x \in S$
6:         $b \leftarrow$Random $x \in S$
7:         $randDistances \leftarrow randDistances$.append($d(a, b)$)
8:         $i \leftarrow i + 1$
9:     end while
10:    sort($randDistances$)
11:    $index \leftarrow \frac{|randDistances|}{n_w}$
12:    $\epsilon \leftarrow randDistances[\lfloor index \rfloor]$
13:    $words \leftarrow \emptyset$
14:    $seen \leftarrow \emptyset$
15:    for Random $x \in S \wedge x \notin seen$ do
16:        $seen \leftarrow seen \cup \{x\}$
17:        $X \leftarrow \emptyset$
18:        for $y \in S \wedge y \notin seen$ do
19:            if $d(pc[x], pc[y]) < \epsilon$ then
20:                $seen \leftarrow seen \cup \{y\}$
21:                $X \leftarrow X \cup \{y\}$
22:            end if
23:        end for
24:        $words \leftarrow words \cup X$
25:    end for
26:    return $words$
27: end function
```

4.4 Evaluation Metrics

As discussed in Section 3, the technique for evaluation of partitioning methods used in this paper is NMI. The method itself is quite simple and has been used frequently for clustering evaluation, which again makes it an obvious choice when selecting an evaluation method considering in some sense netlist partitioning is a form of clustering. As in [18] the formulation of NMI of a partition P and ground-truth T can be expressed as

$$I_{norm}(T, P) = \frac{I(T, P)}{H(T) + H(P)}$$

where $I(T, P)$ is simply the mutual information and $H(X)$ is a type of entropy that normalizes I. Both of which can be calculated by the following equations,

$$I(T, P) = -2 \sum_{i=1}^{C(T)} \sum_{j=1}^{C(P)} \left|T_i^c \cap P_j^c\right| log \left(\frac{\left|T_i^c \cap P_j^c\right| |T|}{\left|T_i^c\right| \left|P_j^c\right|} \right)$$

$$H(X) = \sum_{i=1}^{C(X)} \left|X_i^c\right| log \left(\left|X_i^c\right|\right)$$

where $C(X)$ is the number of classes in partition X, $|T|$ is the total number of element or nodes in the partition, and X_i^c is the set of the i-th class of partition X.

The value returned by the NMI is a real number in the range [0, 1]. The closer to 0 the worse off the partition is to the ground truth, while a NMI of 1 would be an exact match. As an example if a partition where everything is in the same cluster is compared against a ground truth with at least two clusters, then the resulting NMI is 0 as no information is recovered from the partition.

5 Benchmarking Results

In this section, we examine the various performances for different partitioning techniques. To show the techniques generalization, we analyze several different gate-level netlists, with varying amounts of size and logic signals versus data signals ratios. For each netlist, we extract the desired netlist partition based on the register correspondence to the original words in the RTL level code. The partition results are collected from the various heuristic-based method. Finally, the registers within the partition are analyzed against the ground truth.

5.1 Experimental Setup

All simulations were run on an Xeon CPU E5-2690 at 2.90 GHz with 128 GB of RAM. Logic classification and bus-based methods were run with thresholds ranging from 0.75 to 1.00 with a step size of 0.01. A range of depth was used from 2 to 6 with a step size of 1 on bus based, logic classification based, and our PCA-based partitioning. The PCA-based partitioning was run with a varying number of expected words in the range of 200 down to 4. When execution took longer than 2 h, the run was ended. For PCA, the first three components were used for creating the vectors used in partitioning.

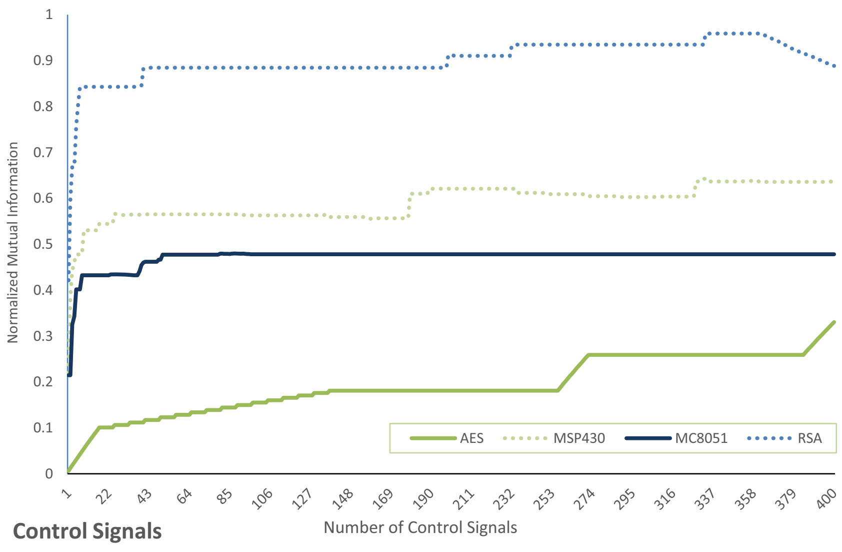

Fig. 3 NMIs found using control signals on RSA, AES, and MC-8051 netlists

The baseline was run with a varying number of control signals from 1 to 400. The NMI for the resulting partition was collected and can be seen in Fig. 3. The maximum and the minimum NMI are plotted with respect to the remaining partitioning methods.

5.2 Normalized Mutual Information

On the results for the AES core, seen in Fig. 4, both REWIND and REBUS were highly stable and neither vary much with the given threshold. REBUS had a better result than the simple classification-based scheme, but both methods are capable of outperforming the control signal based partitioning. The classification method on MSP430 in Fig. 5 had results that varied on both the threshold and the depth parameter. The higher the depth or the higher the threshold, the better the performance, and although REBUS had a higher performance on average, REWIND had the best performance for a certain parameter combination. Also seen in Fig. 5 both REWIND and REBUS were capable of outperforming the control signal based method. However, REWIND only did so with good parameter selection.

The MC-8051 also showed that both REBUS and REWIND could outperform the control signal baseline (see Fig. 6). It should be noted that once again REBUS' partitioning was more consistent, but with certain parameters REWIND was able to overtake the bus scheme in terms of NMI. However, in the last netlist, the RSA core, both

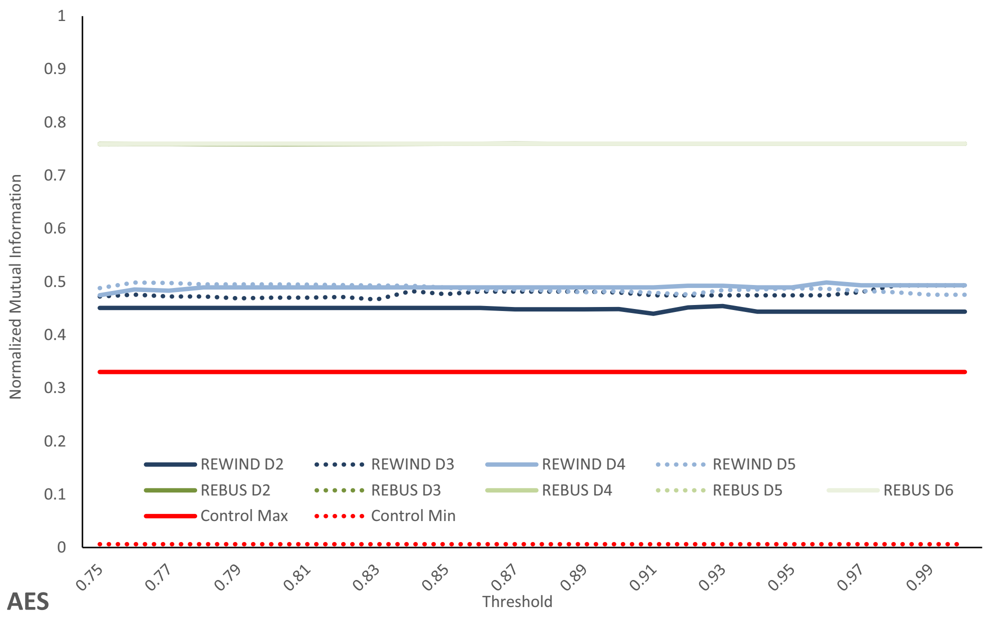

Fig. 4 Cluster Scores found using REBUS and REWIND compared to control signal based matching on the flattened AES-128 netlist

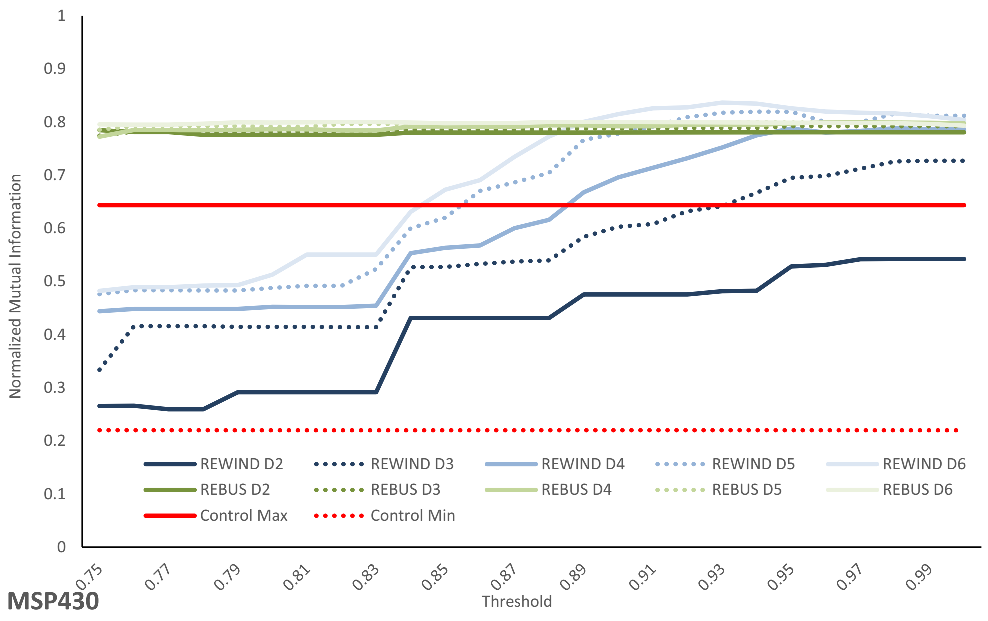

Fig. 5 Cluster scores found using REBUS and REWIND compared to control signal based matching on the MSP430 netlist

REBUS and REWIND had lower maximum NMIs than the control signal based method (see Fig. 7). Not only was control signal based partitioning better, but unlike in the other three netlists REBUS always achieved a lower NMI than the REWIND's simple logic classification method. In short, on the small RSA, control signal-based partitioning performed the best. This might be caused in part by a lack of repetitive structure due to the simplicity/size of the design.

REPCA was also, with a limited parameter set, capable of outperforming the control signal based method in terms of NMI. REPCA, like REWIND and REBUS, was unable to overtake control signal based partitioning as seen in Fig. 11. As one would expect REPCA has its highest accuracy when the expected number of words passed to the program is close to the ground truths number of words. However, in practice, guessing the correct number of words might be difficult, and

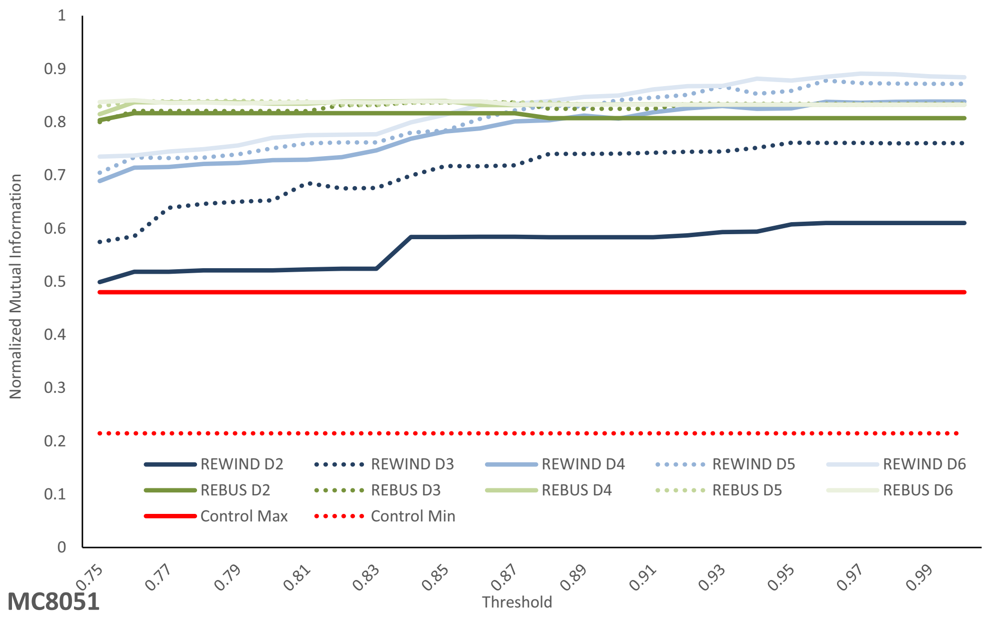

Fig. 6 Cluster scores found using REBUS and REWIND compared to control signal based matching on the MC-8051 netlist

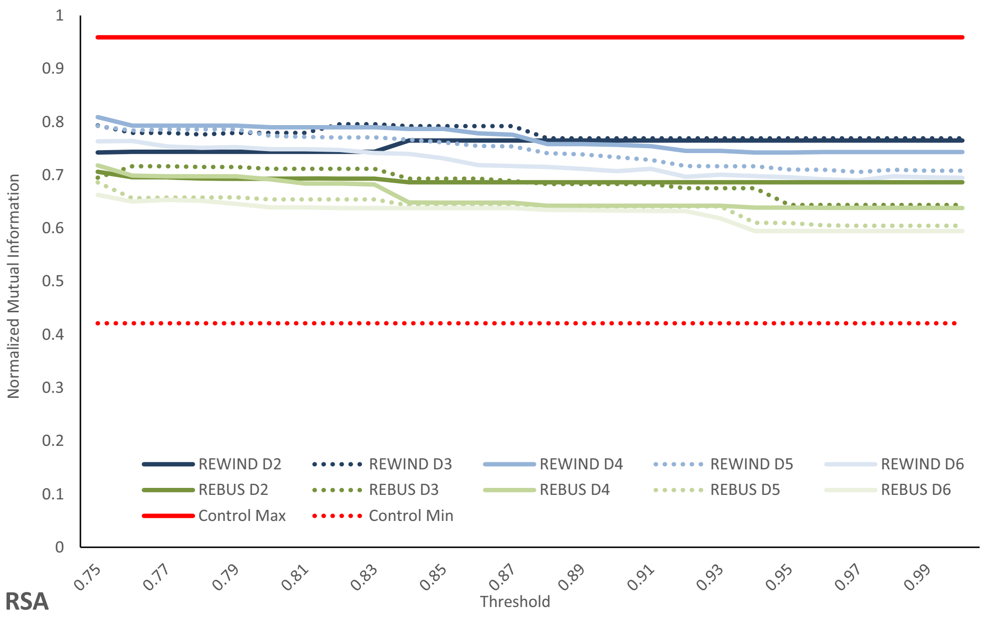

Fig. 7 Cluster scores found using REBUS and REWIND compared to control signal-based matching on the RSA netlist

REPCA's performance suffers when the expected number is too low, as can be seen by Figs. 8, 9, 10, and 11. By leveraging the distributions of distances, the expected number of words might be better inferred, but this is left as a task for future works (Table 1).

Arguments for control signal-based clustering consist of the fact that control signal methods might be more effective on netlists with a higher amount of control logic. It was mentioned in [12] that strongly connected components were used to assist in partitioning the netlist, which would only be useful for netlists that contain large amounts of nets that have some method of recurrent signal propagation. Hence, it might not be fair to utilize each register in the comparison, but to only compare registers that have a feedback signals. The lack of data partitioning leads into a strong argument against [12]; in certain flattened netlists that contain no self-loops (e.g., the flattened AES core used in the experiments). Strongly connected component methods are incapable of performing any partitioning. To reiterate the point, this paper aims to assist in full netlist partitioning, so such analysis on partial netlist recovery is left as a consideration for future work.

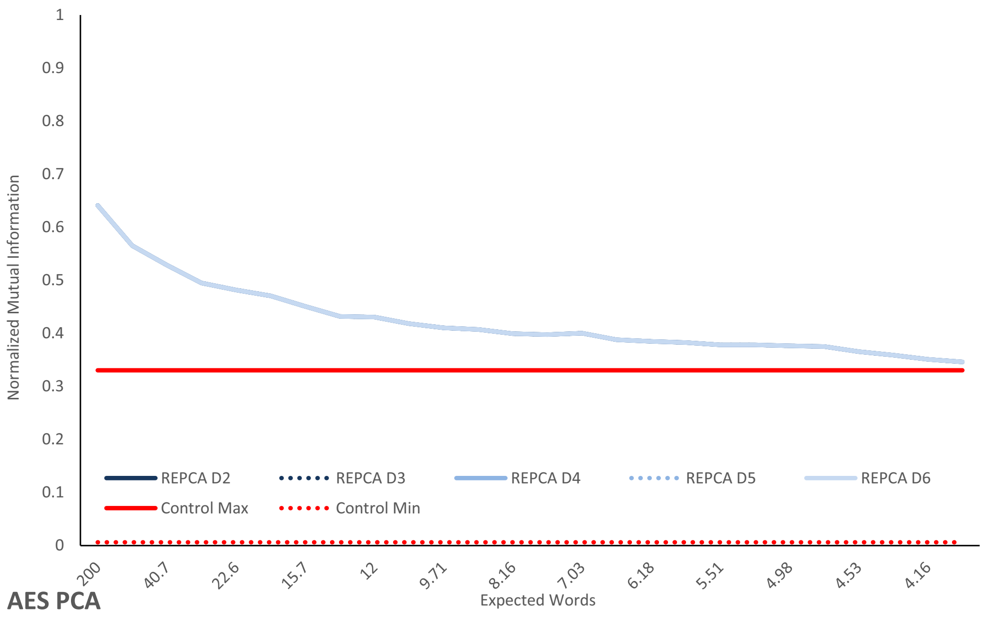

Fig. 8 Cluster scores found using REPCA compared to control signal based matching on the flattened AES-128 netlist

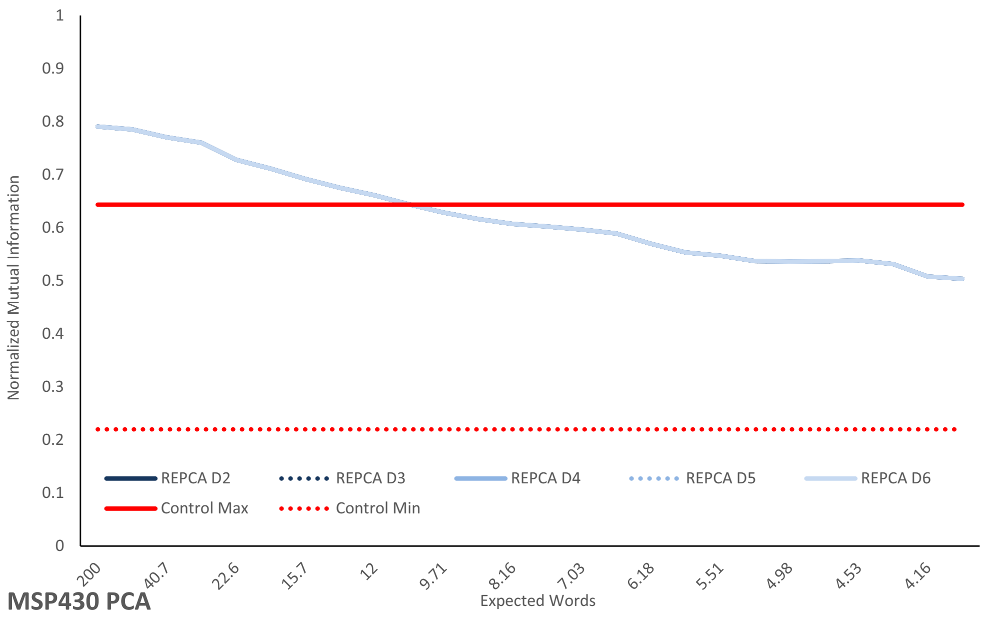

Fig. 9 Cluster scores found using REPCA compared to control signal based matching on the MSP430 netlist

5.3 Runtime

By far the fastest method found was to be the control based signal partitioning (see Table 3). It was capable of running over 100 times faster than any of the other methods on certain netlists and no netlist took longer than a second to execute. While REBUS might not be as accurate, it is capable of running in a fraction of the time compared to slower methods such as REWIND, in part due to its limited comparisons (see Table 2). Due to memory issues among other factors plaguing the scalability of REWIND, the pairwise comparison-based method had a very poor performance on the AES core. Conversely, since the REPCA method has a sizable overhead, its runtime does not get significantly effected by the depth parameter (Table 3).

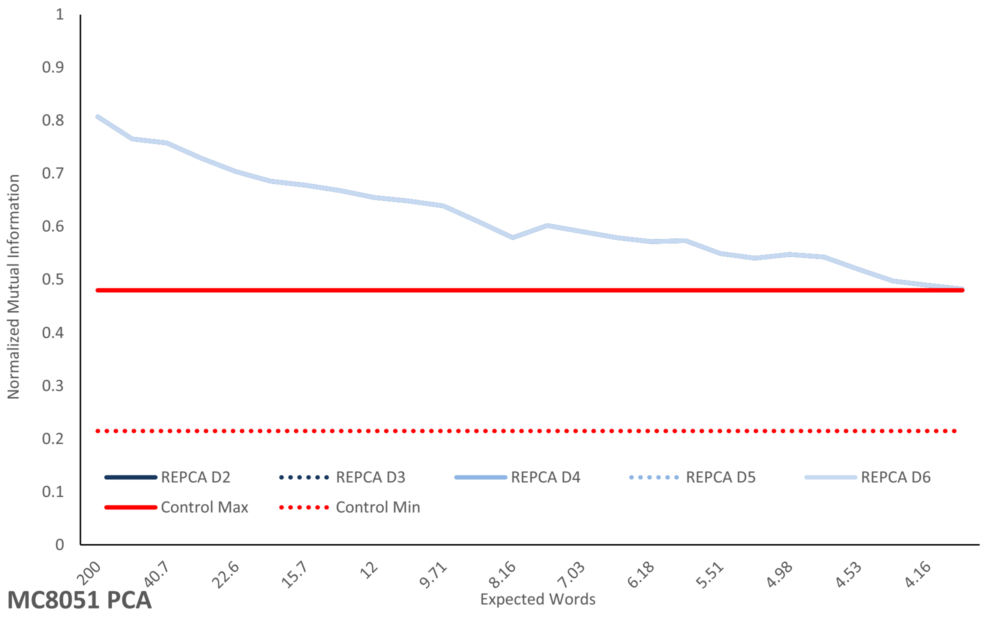

Fig. 10 Cluster scores found using REPCA compared to control signal based matching on the MC-8051 netlist

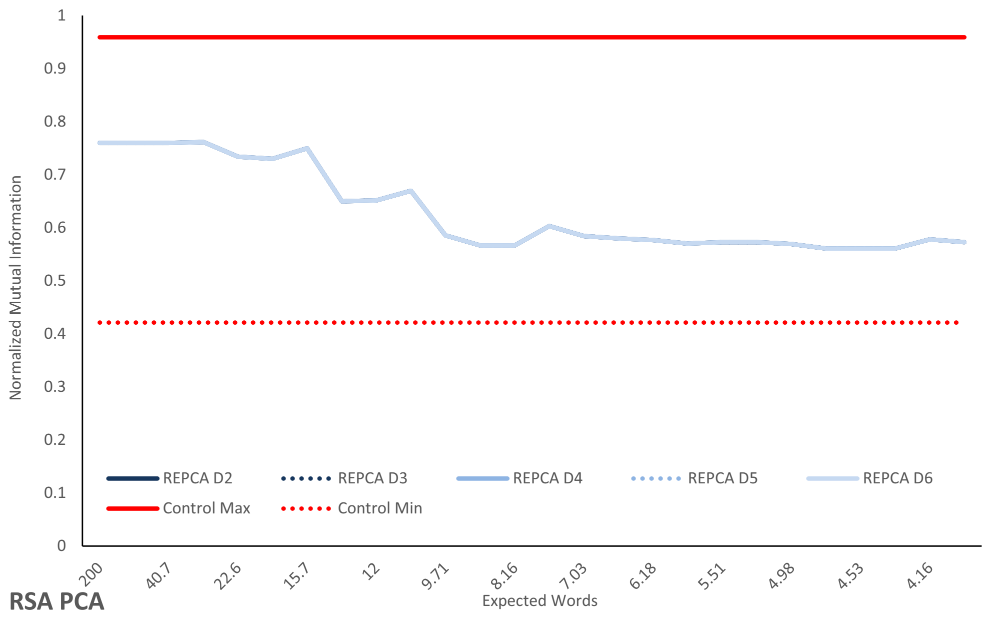

Fig. 11 Cluster scores found using REPCA compared to control signal based matching on the RSA netlist

Table 1 Average time taken for various netlist partitioning methods that relied on the similarity score

Depth	2	3	4	5	6	FF Pins	Ground truth entropy	Ground truth words
AES Logic	146s	385s	1291s	4578s	–	6720	5.36	405
AES Bus	11.4s	9.88s	11.5s	15.1s	19.6s	6720	5.36	405
AES PCA	23.9s	23.8s	23.9s	23.8s	23.9s	6720	5.36	405
MSP430 Logic	4.16s	5.49s	6.82s	7.91s	8.98s	734	4.33	133
MSP430 Bus	0.30s	0.39s	0.48s	0.56s	0.65s	734	4.33	133
MSP430 PCA	0.90s	0.91s	0.89s	0.88s	0.93s	734	4.33	133
MC8051 Logic	1.30s	1.97s	2.65s	3.19s	3.82s	578	4.47	121
MC8051 Bus	0.33s	0.38s	0.42s	0.54s	0.68s	578	4.47	121
MC8051 PCA	1.44s	1.40s	1.53s	1.49s	1.50s	578	4.47	121
RSA Logic	1.29s	1.88s	2.45s	2.64s	2.84s	295	2.53	16
RSA Bus	0.63s	0.70s	0.70s	0.63s	0.68s	295	2.53	16
RSA PCA	0.83s	0.86s	0.87s	0.88s	0.83s	295	2.53	16

Table 2 Average NMI for various netlist partitioning methods that relied on the similarity score

Depth	2	3	4	5	6
AES Logic	0.45	0.48	0.49	0.48	–
AES Bus	*0.76*	*0.76*	*0.76*	*0.76*	*0.76*
AES PCA	0.42	0.42	0.42	0.42	0.42
MSP430 Logic	0.41	0.55	0.62	0.67	0.69
MSP430 Bus	0.78	0.79	0.79	*0.80*	*0.80*
MSP430 PCA	0.62	0.62	0.62	0.62	0.62
MC8051 Logic	0.57	0.71	0.78	0.81	0.82
MC8051 Bus	0.81	0.83	0.83	*0.84*	*0.84*
MC8051 PCA	0.62	0.62	0.62	0.62	0.62
RSA Logic	0.76	0.78	0.77	0.75	0.72
RSA Bus	0.69	0.68	0.66	0.64	0.63
RSA PCA	0.63	0.63	0.63	0.63	0.63

Italic values represent the highest average score for the benchmark

Table 3 Time taken and average NMI for various control signal based partitioning

	Minimum	Maximum	Average time	Average NMI
AES	0.09s	0.46s	0.21s	0.19
MSP430	0.18s	0.71s	0.53s	0.59
MC8051	0.28s	0.89s	0.49s	0.47
RSA	0.06s	0.17s	0.15s	*0.90*

Italic values represent the highest average NMI for the benchmark

6 Conclusion and Future Work

The proposed methods were evaluated and shown to have, for the most part, comparable runtimes. The results lead us to believe that bus-based extraction method using forward propagation are capable of consistently partitioning a netlist with a modest accuracy. While REWIND had a higher accuracy on part of the benchmarks, REBUS was able to outperform REWIND in terms of runtime, and although the control signal baseline method had a higher NMI on smaller netlists, REBUS had a more consistent and on average a higher NMI. Truthfully, the results show that there is a need for more accurate methods. Based on the benchmarking results, we can suggest that:

- For small netlists, the resulting NMI implies that control signal based partitioning will have the best results.
- PCA-based methods work best on larger netlists, when the number of words can be accurately guessed.
- Finally, with no high-level information regarding the circuit, a large netlist can best be recovered by existing netlist reverse engineering methods [19–21].

Further, an efficient netlist partition method opens up opportunities to address many subsequent challenges such as RTL reconstruction, functionality determination, and hardware Trojan detection. After the high-level functionality, often in the format of RTL representation, is reconstructed, the next step is to screen the RTL code for any inserted hardware Trojans or malicious backdoors.

Funding Information The work is partially supported by the National Science Foundation (NSF-1812071, NSF-1703602).

References

1. Hicks M, Finnicum M, King ST, Martin MM, Smith JM (2010) Overcoming an untrusted computing base: detecting and removing malicious hardware automatically. In: 2010 IEEE symposium on security and privacy (SP). IEEE, pp 159–172
2. Banga M, Hsiao MS (2010) Trusted rtl: Trojan detection methodology in pre-silicon designs. In: 2010 IEEE international symposium on hardware-oriented security and trust (HOST). IEEE, pp 56–59
3. Love E, Jin Y, Makris Y (2012) Proof-carrying hardware intellectual property: a pathway to trusted module acquisition. IEEE Trans Inf Forensics Secur 7(1):25–40
4. Sturton C, Hicks M, Wagner D, King ST (2011) Defeating uci: building stealthy and malicious hardware. In: 2011 IEEE symposium on security and privacy (SP). IEEE, pp 64–77
5. Torrance R, James D (2009) The state-of-the-art in ic reverse engineering. In: CHES, vol 5747. Springer, New York, pp 363–381
6. Torrance R, James D (2011) The state-of-the-art in semiconductor reverse engineering. In: Proceedings of the 48th design automation conference. ACM, pp 333–338
7. Subramanyan P, Tsiskaridze N, Li W, Gascón A, Tan WY, Tiwari A, Shankar N, Seshia SA, Malik S (2014) Reverse engineering digital circuits using structural and functional analyses. IEEE Trans Emerging Topics Comput 2(1):63–80
8. Meade T, Zhang S, Jin Y (2016) Netlist reverse engineering for high-level functionality reconstruction. In: 2016 21st Asia and South Pacific design automation conference (ASP-DAC). IEEE, pp 655–660
9. Areibi S, Vannelli A (2000) Tabu search: a meta heuristic for netlist partitioning. VLSI Design 11(3):259–283
10. Buntine WL, Su L, Newton AR, Mayer A (1997) Adaptive methods for netlist partitioning. In: Proceedings of the 1997 IEEE/ACM international conference on computer-aided design. IEEE Computer Society, pp 356–363
11. Dai YY, Brayton RK (2017) Circuit recognition with deep learning
12. Shi Y, Ting CW, Gwee B-H, Ren Y (2010) A highly efficient method for extracting fsms from flattened gate-level netlist. In: Proceedings of 2010 IEEE international symposium on circuits and systems (ISCAS). IEEE, pp 2610–2613
13. Li W, Gascon A, Subramanyan P, Tan WY, Tiwari A, Malik S, Shankar N, Seshia SA (2013) Wordrev: finding word-level structures in a sea of bit-level gates. In: 2013 IEEE international symposium on hardware-oriented security and trust (HOST). IEEE, pp 67–74
14. Tashjian E, Davoodi A (2015) On using control signals for word-level identification in a gate-level netlist. In: Proceedings of the 52nd annual design automation conference. ACM, p 78
15. Couch J, Reilly E, Schuyler M, Barrett B (2016) Functional block identification in circuit design recovery. In: 2016 IEEE international symposium on hardware oriented security and trust (HOST). IEEE, pp 75–78
16. Lancichinetti A, Fortunato S, Kertész J (2009) Detecting the overlapping and hierarchical community structure in complex networks. New J Phys 11(3):033015
17. Meade T, Jin Y, Tehranipoor M, Zhang S (2016) Gate-level netlist reverse engineering for hardware security: control logic register

identification. In: 2016 IEEE international symposium on circuits and systems (ISCAS). IEEE, pp 1334–1337

18. Danon L, Diaz-Guilera A, Duch J, Arenas A (2005) Comparing community structure identification. J Stat Mech: Theory Exp 2005(09):P09008
19. Li W, Wasson Z, Seshia SA (2012) Reverse engineering circuits using behavioral pattern mining. In: 2012 IEEE international symposium on hardware-oriented security and trust, pp 83–88
20. Li W, Gascon A, Subramanyan P, Tan WY, Tiwari A, Malik S, Shankar N, Seshia SA (2013) Wordrev: finding word-level structures in a sea of bit-level gates. In: 2013 IEEE international symposium on hardware-oriented security and trust (HOST), pp 67–74
21. Subramanyan P, Tsiskaridze N, Li W, Gascón A, Tan WY, Tiwari A, Shankar N, Seshia SA, Malik S (2014) Reverse engineering digital circuits using structural and functional analyses. IEEE Trans Emerging Topics Comput 2(1):63–80