Computational Forensic Techniques for Intellectual Property Protection

Jennifer L. Wong[†], Darko Kirovski[‡], Miodrag Potkonjak[†]

[†]Computer Science Department, University of California, Los Angeles, CA 90095
[‡]Microsoft Research, One Microsoft Way, Redmond, WA 98052

Abstract

Computational forensic engineering (CFE) aims to identify the entity that created a particular intellectual property (IP). Specifically, our goal is to identify the synthesis tool or compiler which was used to produce a specific design or program. Rather than relying on watermarking content or designs, the generic CFE methodology analyzes the statistics of certain features of a given IP and quantizes the likelihood that a well known source has created it. In this paper, we describe the generic methodology of CFE and present a set of techniques that, given a set of compilation tools, identify the one used to generate a particular hardware/software design. The generic CFE approach has four phases: feature and statistics data collection, feature extraction, entity clustering, and validation. In addition to IP protection, the developed CFE paradigm can have other potential applications: optimization algorithm selection and tuning, benchmark selection, and source-verification for mobile code.

1 Introduction

The rapid expansion of the Internet, and in particular e-commerce, has impacted the business model of almost all semiconductor and software companies that rely on intellectual property (IP) as their main source of revenues. In such a competitive environment, IP protection (IPP) is a must. Watermarking is currently the most popular form of IPP. To enforce copyrights, watermark protocols rely on detecting a hidden mark specific to the copyright owner. However, watermarking has a number of limitations, in particular when it is applied to hardware and software protection: (i) impact on system performance, (ii) robustness of watermark detection with respect to design modularity, and (iii) the threat of reverse engineering.

Computational forensic engineering (CFE) copes with these problems by trying to identify the tool used to generate a particular IP. In this paper, we present a set of techniques for CFE of design algorithms. The developed CFE technique identifies a tool from a pool of synthesis tools that has been used to generate a particular optimized design. More formally, given a solution S_P to a particular optimization problem instance P and a finite set of algorithms A applicable to P, the goal is to identify with a certain degree of confidence that algorithm A_i has been applied to P in order to obtain solution S_P.

In such a scenario, forensic analysis is conducted based on the likelihood that a design solution, obtained by a particular algorithm, results in characteristic values for a predetermined set of solution properties. Solution analysis is performed in four steps: feature and statistics data collection, feature extraction, clustering of heuristic properties for each analyzed tool, and decision validation.

In order to demonstrate the generic forensic analysis platform, we propose a set of techniques for forensic analysis of solution instances for a set of problems commonly encountered in VLSI CAD: graph coloring and boolean satisfiability. Graph coloring is used for modeling many resource allocation problems. It is widely used in both behavioral synthesis and software compilers for assignment of variables to registers. Boolean satisfiability has equally wide applications for both optimization and constraint satisfaction problems. We have conducted a number of experiments on real-life and abstract benchmarks to show that using our methodology, solutions produced by strategically different algorithms can be associated with their generators with relatively high accuracy.

2 Background Material

2.1 Related Work

We trace the related work along the following lines: forensic engineering in general, copyright infringement policies and law practice, forensic analysis of software and documents, stenography, and code obfuscation.

Forensic analysis is a key methodology in many scientific and art fields, such as anthropology, science, literature, and visual art. For example, forensics is most commonly used in DNA identification. Rudin et al. present the details on DNA profiling and forensic DNA analysis [32]. In literature Thisted and Efron used statistical analysis of Shakespeare's vocabulary throughout his works to predict if a new found poem came from Shakespeare's pen [38]. They provided a high confidence statistical argument for the positive conclusion by analyzing how many new words, words used once, twice, three times and so on would appear in the new Shakespeare's work.

Software copyright enforcement has attracted a great deal of attention among law professionals. McGahn gives a good survey on the state-of-the-art methods used in court for detection of software copyright infringement [30]. In the same journal paper, McGahn introduces a new analytical method, based on Learned Hand's abstractions test, which allows courts to base their decisions on well established and familiar principles of copyright law. Grover presents the details behind an example lawsuit case [19] where Engineering Dynamics Inc., is the plaintiff issuing a judgment of copyright infringement against Structural Software Inc., a competitor who copied many of the input and output formats of Engineering Dynamics Inc.

Forensic engineering has received little attention among the computer science and engineering research community. To the best knowledge of the authors, to date, forensic techniques have been explored for detection of authentic Java byte codes [2] and to perform identity or partial copy detection for digital libraries [7]. Recently, steganography and code obfuscation techniques have been endorsed as viable strategies for content and design protection. Protocols for watermarking active IP have been developed at the physical layout [9], partitioning [22], and behavioral specification [31] level. In the software domain, good survey of techniques for copyright protection of programs has been presented by Collberg and Thomborson [10]. They have also developed a code obfuscation method which performs code transformations such that a program is converted into an equivalent program which is more difficult to reverse engineer.

Integrated circuit reverse engineering techniques and the developed forensic engineering approach have complementary roles in forming an overall intellectual property protection approach. The reverse engineering techniques extract information about specification of the design and forensic engineering establishes the proof of the authorship using this information.

The key difference between the research presented in this paper and all published forensic engineering

research is that our goal is not to identify a copy of a program, text, or design among a large set of entities but to establish the relationship between the design and tool that is used to produce the artifact. Therefore, the previous research is essentially focused on rapid search for an entity with particular properties in a large database. Our goal, on the other hand, is to find the likelihood that a specific program is used to produce a particular solution and therefore the design of interest.

2.2 Other Applications of Forensic Engineering

Computational Forensic engineering has the potential to enable a large variety of applications. For instance, computational forensic engineering can be used for the development of optimization algorithms, as a basis for the development or improvement of existing IPP techniques, for the development of more powerful benchmarking tools, for enabling security, and facilitating runtime prediction.

More specifically, computational forensic engineering can be used to perform optimization algorithm tuning, instance partitioning for optimization, algorithm development, and analysis of algorithm scaling. For example, forensic engineering can analyze the performance of the algorithm on various test cases and pinpoint the types of instances on which the algorithm does not perform well. The technique can also be used to partition instances into components each of which can be processed using algorithms which will perform best with respect to the structure of each part. Furthermore, one can tune parameters of heuristics with respect to the properties of the targeted instance.

Additionally, computational forensic engineering can assist in the development of IPP techniques such as watermarking, obfuscation, and reverse engineering. Watermarking techniques often add a signature in the form of additional constraints into the instance structure. Forensic techniques can be used to determine the proper type of additional constraints to embed in the instance in order to ensure the uniqueness of the watermark without reducing the quality of the obtained solution. Reverse engineering of algorithms can be facilitated by forensic analysis in several ways. For example, the forensic technique can be used to determine which specific instances of the problem to analyze in order to identify the key optimization mechanisms of the algorithm.

Benchmark sets can be built to accurately identify the benefits and limitations of an algorithm with respect to particular type of instance. Forensic analysis can also be used to generate instances with specific structures on which an algorithm does not perform well, performs exceptionally well, or to build a compact, yet diverse benchmark set which fairly tests all competing algorithms/tools.

Security applications include the generation of secure mobile codes and runtime checks of code. For example, one can check using computational forensic techniques whether or not a delivered piece of code was indeed generated using a particular compiler by looking at the register assignment (graph coloring) and scheduling. Lastly, forensic engineering can be used for the runtime prediction of a particular algorithm. Proper resource allocation, such as memory, can be identified using forensic techniques by examining the memory and runtime of an algorithm on instances with similar properties and of similar size.

3 Forensic Engineering: The Generic Approach

Forensic engineering aims at providing both qualitative and quantitative evidence of substantial similarity between the design tool and a solution. The generic problem that a forensic engineering methodology tries to resolve can be formally defined as follows. Given a solution S_P to a particular optimization problem instance P and a finite set of algorithms A applicable to P, the goal is to identify with a certain degree of confidence which algorithm A_i has been applied to P in order to obtain solution S_P.

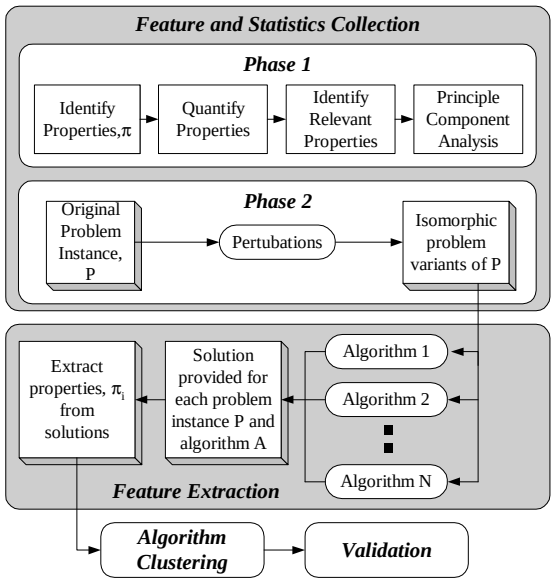


Figure 1: Flow of the Generic Forensic Engineering Approach.

An additional restriction is that the algorithms (their software or hardware implementations) have to be analyzed as black boxes. This requirement is based on two facts: *(i)* similar algorithms can have different executables and *(ii)* parties involved in the ruling are not eager to reveal their IP even in court. Another important assumption is that in order for a distinction to exist between the solutions from different algorithms, the problem instance for which a solution is found must have a large number of solutions of equal or similar quality. Note that this is almost always the case in real life. The global flow of the generic forensic engineering approach, presented in Figure 1, consists of four fully modular phases: feature and statistics collection, feature extraction, algorithm clustering, and validation.

Feature and Statistics Collection

The first phase can be divided into two subphases. The first subphase is to identify and analyze relevant functional and structural properties of the problem solutions. The properties are obtained by analyzing solutions produced by various algorithms and identifying common features in the solutions produced by a particular algorithm. Alternatively, properties can be developed using general intuition presented in Section 4.3. For example, the graph coloring RLF algorithm [28], which is explained in more detail in the next section, is likely to have solutions with a large number of nodes in the graph to be colored with the same color, as well as some colors which only color one or two nodes.

The next step is to quantify properties by abstracting them to their numerical values. The goal is to eventually locate the solutions for each algorithm into n-dimensional space. The dimensions are quantified properties which characterize solutions created by all considered algorithms. For example, for graph coloring solutions, we can find the cardinality of the largest independent set (IS) and normalize all other sets against it. Different coloring algorithms may produce solutions characterized by significantly different probability distribution functions (pdfs) for this feature.

Then, we identify the relevant properties, and discard the ones for which all considered algorithms display equivalent statistics. A property is considered as viable only if at least two algorithms have statistically distinct pdfs under it. For example, in the experimental results, the feature clausal stability for Satisfiability shows histograms which are different for each of the algorithms.

In the fourth step, we conduct the principle component analysis [21, 24, 29]. We attempt to eliminate any subset of features which will provide the same information about the algorithms. The goal is to find the smallest set of features needed to fully classify the algorithms in order to improve efficiency and more importantly, statistical confidence.

The second subphase is instance preprocessing. We make order and lexical perturbations format the instance. This step is performed to eliminate any dependencies an algorithm may have on the naming or form of the input, such as variable labels. We present the details of this phase for both graph coloring and boolean satisfiability in Section 4.

Feature Extraction

We begin by running all the perturbed instances through each of the algorithms, and gathering all the solutions. We apply fast algorithms for extraction of the selected properties from each of the solutions. In some cases, extracting the features from the solutions is trivial. But in other cases, it can be complex and time consuming. Additionally, given that a large number of solutions is produced for each of the algorithms, this process can take a significant amount of time. The gathered property values are then used in the algorithm clustering phase.

Algorithm Clustering

In this step, we begin by selecting the relevant properties. Once the irrelevant properties have been eliminated, we place each of the remaining properties/features into n-dimensional space, and cluster the results. This in itself is a NP-complete problem. The goal is to define areas in the n-dimensional space which distinguishes each of the algorithms. If properties/features which capture each of the algorithms have been used, the space will be divided into single subspaces for each algorithm. However, it is possible that multiple subspaces are found for each algorithm as the result of properties which are not relevant for each and every algorithm.

Validation

Our final step is the application of non-parametric resubstitution software [16] to establish the validity of our ability to distinguish distinct algorithms. Specifically, we run five hundred resubstituitions of 80% of the sample points. When a new solution is available, the generic flow and tools fully and automatically determine which algorithm was used. The details of this phase and the algorithm clustering phase are presented in Section 5.

4 Forensic Engineering: Feature and Statistics Collection

4.1 Graph Coloring

In this section, we demonstrate the developed forensic engineering methodology using the graph K-colorability problem. We first define the problem and provide description of four widely used graph coloring algorithms. The graph coloring problem is a well known optimization task that can be defined in the following way.

PROBLEM: GRAPH K-COLORABILITY
INSTANCE: *Graph $G(V, E)$, positive integer $K \leq |V|$.*

QUESTION: *Is G K-colorable. i.e., does there exist a function $f : V \rightarrow 1, 2, 3, .., K$ such that $f(u) \neq f(v)$ whenever $u, v \in E$?*

In general, graph coloring is an NP-complete problem [17]. Due to its applicability to a wide range of areas, a number of exact and heuristic algorithms for graph coloring have been developed. For the sake of brevity and diversity, in this paper, we focus our attention on a set of algorithms that consists of sequential greedy (SEQ), backtrack DSATUR, MAXIS (RLF-based), and Tabu search.

- **Sequential Greedy (SEQ)** is one of the oldest and simplest constructive algorithms for graph coloring. It was originally proposed by Welsh and Powell [40]. The algorithm colors nodes sequentially, one at the time. The key factor in the effectiveness of the algorithm is the order in which the vertices are colored and by which color. Most commonly, the vertices are ordered by highest degree (number of adjacent vertices) following the popular and effective most-constrained generic heuristic rule. Additionally, each of the potential color classes is numbered. The algorithm colors each vertex, in the selected order, with the lowest numbered color class which is not used by its adjacent vertices.

- **MAXIS** is a recursive, large first (RLF) algorithm based on a approach developed by Bollobas and Thomason [5]. It is an exponential backtrack algorithm which searches and tries to build large maximal independent sets (MISs). Each of these sets represents a set of nodes that can be colored using a single color. In order to build large MIS, the algorithm employs backtrack (depth first) search by selecting single vertices to add to the current independent set (IS). Following the widely used minimally constraining heuristic paradigm, the algorithm always adds the node that has the minimal negative impact on the remaining nodes in the instance which still can be included in the current independent set. After the addition of each vertex, several different choices may arise as possible next selections. In this case, specific search tree pruning rules are determined by the user to eliminate exponential growth. For example, if the graph has more than 300 vertices in it, then only two different options are considered, however if there are 200 vertices, then we would follow three of the branches. Once a large maximal independent set is found, the set of vertices is colored using a single color, removed from the instance, and the process is recursively repeated.

  A basic example of how MAXIS colors graphs is presented in Figure 2. The algorithm begins by building a independent set (IS), which may be maximal. In this case, the starting vertex, vertex 6, is randomly selected. Both vertices 1 and 5 are adjacent, or neighbors, to this vertex and therefore can not be considered to add to the set. Of the remaining vertices, we examine the vertices which are adjacent to the neighbors (1 and 5) of the current IS. In this case, we find vertices 2 and 4, and vertex 4 has the highest degree, and is therefore added to the current IS. We again eliminate all vertices adjacent to the current IS, which leaves vertex 2 as the only remaining option, and is added to the IS. All the nodes in the set are colored black. The second iteration, or IS is built by randomly selecting a new starting node, we chose vertex 1. vertex 5 is the only remaining node not adjacent to vertex 1, we add it to the current IS, and color the vertices using grey color. The only remaining vertex, 3 forms the final IS, and is colored white.

- **Backtrack DSATUR** was originally proposed by Brelaz [6] as a greedy algorithm which aims at coloring denser regions, or regions with high connectivity, of the graph first. The decision mechanism of the algorithm is the saturation degree of a vertex, which is the number of color classes used by adjacent vertices. DSATUR begins by applying Color 1 to the vertex with the largest number of adjacent vertices. At each step the algorithm selects the vertex with the highest saturation degree and colors the vertex with the lowest numbered color which is not used by

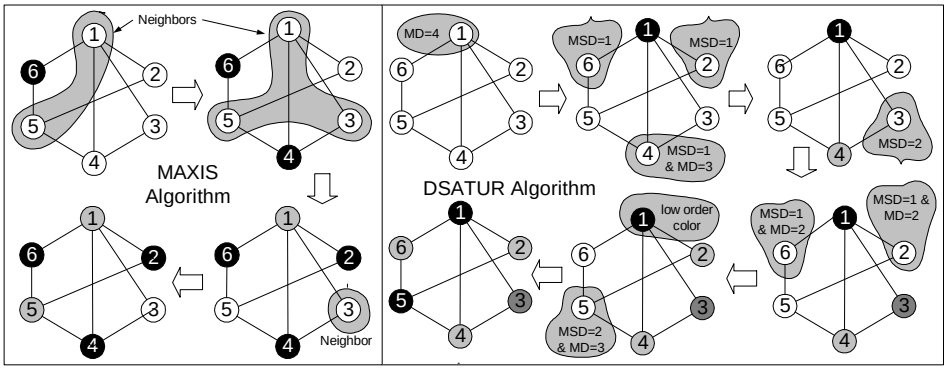


Figure 2: Example of how MAXIS and DSATUR algorithms create their solutions. MD - maximal degree; MSD - maximal saturation degree.

the adjacent vertices. If there is a tie in saturation degree, the vertex with the largest number of uncolored adjacent vertices is selected. If a tie still exists, one of these vertices is selected at random. The backtracking version of the algorithm uses dynamic reordering of the vertices during each backtrack step.

A simple example of the DSATUR algorithm is shown in Figure 2. The algorithm begins by selecting the vertex with the highest degree, or maximum number of adjacent vertices, Vertex 1, and colors it with Color 1 (Black). The saturation degree (SD) of each of the adjacent vertices is calculated. In this situation, vertices 2, 4, and 6 all have the maximal saturation degree (MSD), 1. However, vertex 4 has the highest number of adjacent vertices (3) and therefore is colored with the next available color, Color 2 (light gray). The saturation degree of all the vertices is updated and vertex 3 has the largest, therefore it is selected and colored with the next available non-conflicting color, Color 3 (dark gray). Vertex 5 is selected next, and we color it with the lowest possible color, Color 1. Finally, we color vertex 6 with Color 2.

- **Tabu Search** belongs to a family of probabilistic iterative improvement algorithms. It assigns colors to nodes at random, and then tries to correct the conflicts by reassigning conflicting vertices to different colors [20]. The algorithm selects options from the current state by randomly selecting vertices which are in conflict, i.e. adjacent to other vertices with the same color. For each of the options, the color of the vertex is changed to the color which has the least number of conflicts in the graph. The option which decreases the largest number of conflicts in the graph is selected. A list of k previous moves is kept to assist the algorithm from oscillating and moving out of local minima.

A successful forensic technique should be able to, given a colored graph, distinguish whether a particular algorithm has been used to obtain a particular solution. The key to the efficiency of the forensic method is the selection of the properties used to quantify algorithm-solution correlation. We have developed the following properties that aim at analyzing the structure of the solution for the GC problem:

[π_1] **Color class size.** A histogram of IS cardinalities is used to filter greedy algorithms that focus on coloring graphs constructively (e.g. RLF-like algorithms). Such algorithms tend to create large initial independent sets at the beginning of their coloring process. To quantify this property, we compare the cardinality of the largest IS normalized against the size of the average IS in the solution. Alternatively, as a slight

generalization, in order to achieve statistical robustness, we use 10% of the largest sets instead of only the largest. Interestingly, on real-life applications the first metric is very effective, and on random graphs the second one is strong indicator of the coloring algorithm used.

[π_2] **Number of edges incident to large independent sets.** This property is used to enhance the accuracy of π_1 by excluding easy-to-find large independent sets from consideration in the analysis. Note that large MISs are not necessarily good candidate to be colored with a single color, in particular when many constituent nodes have low degrees. We use k% of the largest sets and measure the percentage of edges leaving the IS.

[π_3] **Number of vertices that can switch color classes.** This criteria, in a sense, analyzes the quality of the coloring. A good (in a sense of being close to a deep local minima) coloring solution will have more nodes that are able to switch color classes. It also characterizes the greediness of an algorithm because greedy algorithms commonly create many color classes that can absorb large portion of the remaining graph at the end of their coloring process . Note that probabilistic algorithms will often create solution that have low value for this property because they will terminate their search as soon as a solution with a given number of colors is found. The percentage of nodes which can switch colors versus the number of nodes is used.

[π_4] **Color saturation in neighborhoods.** This property is calculated using a histogram that counts for each vertex the number of adjacent nodes colored with one color. Greedy algorithms and algorithms that tend to sequentially traverse and color vertices are more likely to have node neighborhoods dominated by fewer colors. We want to know the number of colors in which the neighbors of any node are colored. The Gini coefficient of the histogram is used as well as the average value to quantify to single numbers this property. The Gini coefficient is a measure of dispersion within a group of values, calculated as the average difference between every pair of values divided by two times the average of the sample. The larger the coefficient, the higher the degree of dispersion.

[π_5] **Sum of degrees of nodes included in the smallest color classes.** The analysis goal of this property is similar to π_5 with the exception that it focuses on identifying algorithms that perform neighborhood look ahead techniques [25]. The values are normalized against the average value.

[π_6] **Percent of maximal independent subsets.** This property can be highly effective in distinguishing algorithms that color graphs by iterative color class selection (RLF). Supplemented with property π_3, it aims at detecting fine nuances among similar RLF-like algorithms.

The itemized properties are effective only on relatively large instances, where the standard deviation of histogram values is relatively small. Using standard statistical approaches [14], the function of standard deviation for each histogram can be used to estimate the standard error in the reached conclusion.

Although instances with small cardinalities cannot be a target of forensic methods, for the sake of simplicity and clarity, we use a graph instance in Figure 3 to illustrate how two different graph coloring algorithms tend to have solutions characterized with different properties. The applied algorithms are DSATUR and MAXIS. Specified algorithms color the graph constructively in the order denoted in the figure. If property π_1 is considered, the solution created using DSATUR has a histogram $\chi_{\pi_1}^{DSATUR} = \{1_2, 2_3, 0_4\}$, where histogram value x_y denotes x sets of color classes with cardinality y. Similarly, the solution created using MAXIS results in $\chi_{\pi_1}^{MAXIS} = \{2_2, 0_3, 1_4\}$. Commonly, extreme values point to the optimization goal of the algorithm or characteristic structure property of its solutions. In this case, MAXIS has found a maximum independent set of cardinality $y = 4$, a consequence of the algorithm's strategy to search in a greedy fashion for maximal ISs.

We further illustrate the intuition of how the defined properties contribute to the classification of each of the algorithms in the statistical clustering phase by presenting a brief analysis of each algorithm's solution structure in terms of the properties.

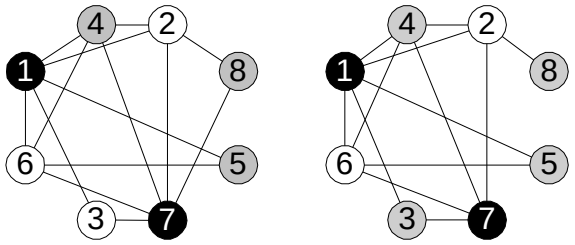


Figure 3: Example of two different graph coloring solutions obtained by two algorithms DSATUR and MAXIS. The index of each vertex specifies the order in which it is colored according to a particular algorithm.

The MAXIS algorithm will build solutions to the graph coloring problem which contain many MISs. Structurally, the algorithm selects vertices with a large number of adjacent vertices in the graph to add to the current IS. As a result the solutions to the MAXIS algorithm will contain a higher number of MISs than any of the other algorithms, and each MIS will have a large number of adjacent edges. Therefore, it is expected that MAXIS solutions will have higher π_6 and π_2 values than the other considered algorithms. Additionally, because MAXIS selects large MIS in the early stages of the algorithm, it is expected that in the later stages, very few vertices will be selected per IS, and therefore will result in many color classes of small size with very few adjacent edges. A larger normalized value is expected for property π_1 as a result.

For the Tabu algorithm, we expect very different solution structures than those found by the MAXIS algorithm. The random initial assignment of the vertices, increases the likelihood that the solution will have color class sizes which are close in size (π_1 value close to 1). As a result, the likelihood of the solution to have small color classes is very low compared to the other approaches. Additionally, as a result of the initial random assignment, we expect that the likelihood of the algorithm finding MISs is slim, resulting in a low π_6 value. Tabu attempts to eliminate conflicts in the graphs by switching the current color of a vertex to the color which has the least conflict. The resulting solutions should contain very few vertices which have the flexibility to change colors. The reasoning is that the initially applied coloring of the adjacent vertices to the conflict vertex are likely to be widely spread. Even though Tabu will attempt to change the color of the node to the color which is causing the least amount of conflict, the final solutions will result in many colors being used by the adjacent vertices, eliminating many options for changing colors (low π_3 value). This also results in the color saturation in neighborhoods of the graph (π_4) to be low.

While the color classes of the Tabu algorithm should be fairly even in size, for the DSATUR algorithm, the color class size is expected to be wide spread (high π_1). DSATUR begins with the vertex which has the highest impact on the graph. And then continues coloring vertices which have a large number of colored adjacent vertices. As a result we expect the colors applied to the early vertices of the graph to be less likely applied than other colors, because they eliminate many adjacent vertices from having the same color. Therefore, it is expected that small color classes will have a large number of adjacent edges (high π_5 value). The sequential coloring of the nodes, i.e. colored with the lowest non-conflicting color, results in a large number of vertices which can be colored with the high colors without conflict (π_3). Only in highly dense areas of the graph will the vertices be unlikely to be able to switch color classes.

It is expected that the greedy algorithm will find the largest MIS as a result of coloring the nodes with the highest degrees first. However, it is unlikely that the algorithm will find many maximal independent subsets (low π_6 value). Again each of the selected nodes is colored with the lowest possible

color resulting in a high π_3 value. We additionally expect that the color class sizes will be widely spread, resulting in a high π_1 value.

According to these properties, we can distinguish each of the algorithms from each other using these properties. We display the distinguishing properties in Figure 4. The distinction between DSATUR/Greedy is difficult because of the similarities between how the algorithms color the vertices with the lowest colors. However, in the majority of cases, properties π_5 and π_6 should distinguish the algorithms. Also, the distinction between DSATUR/MAXIS most often can be distinguished by properties π_1 and π_6, and in various cases by π_2, π_3, and π_5.

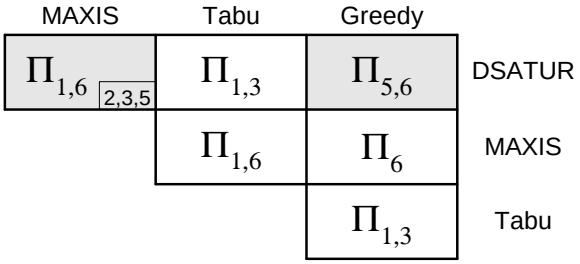


Figure 4: Property distinction between algorithms.

4.2 Boolean Satisfiability

We also present the key ideas for applying the forensic engineering methodology using the boolean satisfiability (SAT) problem. The SAT problem can be formally defined in the following way [17].

Problem: SATISFIABILITY (SAT)

Instance: *A set of variables V and a collection C of clauses over V.*

Question: *Is there a truth assignment for V that satisfies all the clauses in C?*

Boolean satisfiability was the first problem to be defined as NP-complete problem [17]. It has been proven that every other problem in NP can be polynomially reduced to the Satisfiability problem [17]. SAT techniques have been used in testing [37, 26], logic synthesis, and physical design [15]. There are at least three broad classes of solution strategies for the SAT problem. The first class of techniques are based on probabilistic search [36, 35], the second are approximation techniques based on rounding the solution to a nonlinear program relaxation [18], and the third is a great variety of BDD-based techniques [8]. For the sake of brevity, we demonstrate our forensic engineering technology on the following SAT algorithms.

- **WalkSAT** is a local search algorithm which is enhanced by randomization. The algorithm selects with probability p a variable occurring in some unsatisfied clause and flips its truth assignment. Conversely, with probability 1-p, the algorithm performs a greedy heuristic such as GSAT. The optimal value for p was found to be between 0.5 and 0.6 [34]. The GSAT algorithm, also developed by Selman[33, 34], performs a greedy local search. It identifies for each variable v the difference (DIFF) between the number of clauses currently unsatisfied that would be satisfied if the truth value of v were reversed and the number of clauses currently satisfied that would become unsatisfied if the truth value of v were flipped. The algorithm pseudo-randomly flips assignments of variables with the greatest DIFF.

- **NTAB** developed by Crawford [11] consists of two phases. The first phase is local search. The local search algorithm tries to weight the difficulty of clauses, and therefore variables in the clauses.

This is done by applying a greedy approach similar to GSAT. The truth assignment of variables in a random initial solution are flipped to try and increase the number of satisfied clauses in the instance. (Note that this is similar to the greedy approach of GSAT). Once the random assignment shows no additional improvement, the weights of the clauses which remain unsatisfied are increased. The local search is performed multiple times, in order to identify the clauses which are the most difficult. The second phase is the search tree. The search tree preferentially branches on the variables that occur more often in clauses with higher weights.

- **Rel_SAT_rand** is an approach to the boolean satisfiability problem developed by Bayardo and Schrag[3]. It is an enhanced version of the Davis-Putnam proof procedure [13]. The algorithm incorporates CSP look-back techniques, specifically conflict directed backjumping, and learning schemes.

In order to correlate an SAT solution to its corresponding algorithm, we have explored the following properties of the solution structure.

[π_1] **Percentage of non-important variables.** A variable v_i is *non-important* for a particular set of clauses C and satisfactory truth assignment $t(V)$ of all variables in V, if both assignments $t(v_i) = T$ and $t(v_i) = F$ result in satisfied C. For a given truth assignment t, we denote the subset of variables that can switch their assignment without impacting the Satisfiability of C as V_{NI}^t. In the remaining set of properties only functionally significant subset of variables $V_0 = V - V_{NI}^t$ is considered for further forensic analysis.

[π_2] **Clausal stability - percentage of variables that can switch their assignment such that $K\%$ of clauses in C are still satisfied.** This property aims at identifying constructive greedy algorithms, since they assign values to variables such that as many as possible clauses are covered with each variable selection.

[π_3] **Ratio of true assigned variables vs. total number of variables in a clause.** Although this property depends by and large on the structure of the problem, in general, it aims at qualifying the effectiveness of the algorithm. Large values commonly indicate usage of algorithms that try to optimize the coverage using each variable.

[π_4] **Ratio of coverage using positive and negative appearance of a variable.** While property π_3 analyzes the solution from a perspective of a single clause, this property analyzes the solution from the perspective of each variable. Each variable v_i appears in p_i clauses as positively and n_i clauses as negatively inclined. The property quantifies the possibility that an algorithm assigns a truth value to $t(v_i) = p_i \geq n_i$.

[π_5] **The GSAT heuristic.** For each variable v the difference **DIFF=a-b** is computed, where **a** is the number of clauses currently unsatisfied that would become satisfied if the truth value of v were reversed, and **b** is the number of clauses currently satisfied that would become unsatisfied if the truth value of v were flipped. This measure only applies to maximum SAT problems, where the problem is to find the maximum number of clauses which can be satisfied at once.

As in the case of graph coloring, the listed properties demonstrate significant statistical proof only for large problem instances. Instances should be large enough to result in low standard deviation of collected statistical data.

We present a brief analysis of the SAT algorithms and property π_1 and π_3 in order to illustrate how the boolean satisfiability properties assist in the classification of the algorithms in the statistical clustering phase.

For property π_1, we are analyzing the percentage of variables which can be assigned either True or False in the solution without making the instance unsatisfiable. The WalkSAT algorithm builds a

solution for a random initial solution. When a solution is found, it is highly probably that the solution is dependent on variables which would have been non-important, because of the random initial assignment. Therefore, it is expected for WalkSAT to have a lower π_1 value than the other algorithms which build structured solutions. For example, NTAB constructively builds its solution by building a search tree which branches on the variables which appear in clauses which appear to be difficult. As a result, these variables are assigned early in the search tree. When a solution is found, all the variables which were not branched on in the search tree can be either True or False. Therefore, it is expected that the NTAB algorithm will have a large number of non-important variables.

The clausal truth percentage, property π_3, tries to examine if the algorithm tries to satisfy multiple variables in each clause. Because WalkSAT tries to improve on a random solution by flipping variables which increase the number of clauses that are satisfied. When a solution is found, in most cases it will be an inflexible solution, in the sense that very few variables can flip their assignment without making the solution invalid. As a result, of these facts, the clausal truth percentage will be spread over the spectrum. In the case of NTAB, the algorithm will most likely have an median π_3 value. With a high π_1 value, the algorithm will have a large number of variables which are non-important. These variables can then be used to build a stronger satisfiability solution, which can be achieved by assigning the variables such that multiple variables per clauses evaluate to True.

4.3 Generic Property Development

Identification and selection of solution properties is essential for the effectiveness of any forensic engineering technique. While each problem may require one or more unique features, many of properties can be applied to a wide set of problems. More importantly, there is a systematic way to identify the corresponding features in different problems that can be used to identify adequate properties. As a matter of fact, it appears that a small number of features guide this task. Our goal in this subsection is not to present an ultimate set of properties for all possible problems, but more to provide intuition how one can define relevant properties for a specific targeted problem. To make the discussion of the technique complete, we demonstrate its instantiation on several canonical problems, such as scheduling and partitioning.

In order to be self-contained, we briefly introduce the scheduling and partitioning problems. The scheduling problem is defined on a directed graph where each vertex represents an operation and edges indicate execution dependencies between the operations. The objective is to minimize the amount and/or cost of functional units used while scheduling the graph in a given number of clock cycles and keeping all the dependencies satisfied. The partitioning problem aims at dividing all nodes of an undirected graph into k subsets, where the numbers of nodes in each set are as nearly equal as possible and the number of edges between nodes in different sets is minimized.

One can identify at least three types of properties.

[P_1] **Perturbation-based Properties.**

These types of properties try to identify the structure of the solution by analyzing its behavior using local perturbations. The main focus is on perturbation of a solution. The goal is to identify features of the neighborhood of the generated solution for a variety of definitions of neighborhood topology. Under the assumption that the problem instances have many solutions with similar quality, these properties often attempt to determine the strength of the solution with respect to a particular criteria. For example, in the case of the SAT problem the solution only needs to satisfy the each clause using a single variable. If the variable assignment for the solution can handle many changes i.e. flips of variable assignments, we can assume that the solution is resilient on changes and that some implicit effort was placed by the algorithm to produce the solution and to achieve this property.

Definition of perturbation-based properties can be applied to instances in a variety of ways. For example, a specific property of this type can focus on values while preserving the solution or for a given distance from the solution. The entities of focus can be analyzed individually or in groups, and can be equally weighted or compared with respect to their importance (difficulty) in the objective function and constraints.

Examples of this type of property for the SAT and GC problems are SAT π_1, π_2, and π_5 and GC π_3. Each of these properties quantify the amount of flexibility in the solution. If we consider the partitioning problem, an example of a perturbation property is the number of pairs of nodes which can switch partitions without reducing the quality of the solution. Properties such as the number of operations which can change clock cycles (without violating any constraints or a percentage of constraints) can be applied to solutions of the scheduling problem.

[P_2] **Count.** Alternatively, we can focus on the number of occurrences of specific feature in a solution. These properties often correlate well with the tendencies of an algorithm when deciding assignment of the variables. As in the case with the perturbation properties, one can select single entities, pairs, and subsets as scope of consideration. Additionally, this class of properties can be augmented with a variety of statistical measure mechanisms such as average, variance, mean, minimum, or maximum.

GC properties π_1 and π_5 along with SAT π_3 are examples of count properties. Measurements concerning both the variables and the constraints of the problem solutions are illustrated. For the scheduling problem, the average or percentage of operations per cycle can be considered. One could determine the number of saturated clock cycles or the number of operations on the critical path in clock cycles with small operation counts. For the partitioning problem, meaningful properties could be the percentage of nodes with high or low degree in each partition or the percentage of nodes which are in the same partition as all of its neighbors.

[P_3] **Tendency to follow natural algorithm constructs.** Natural algorithm constructs, such as greedy or maximally constrained maximally constraining heuristic (MC/MC), are often as the underlying constructs for combinatorial optimization algorithms. This class of properties has as the goal to identify to what level these constructs were employed by a specific algorithm. For example, in GC property π_2, number of edges incident to large independent sets, tries to quantify the level of greedy optimization principle used by the algorithm to select the largest number of nodes possible which can be colored with a single color. In terms of MC/MC constructs, the GC property π_4 tries to identify algorithms which focus on coloring the neighbors of a node with the least constrained color as possible, implying the same color.

Variable in the SAT problem can have a natural tendency, in the sense that if the variable appears in constraints more often complemented than uncomplemented, a greedy algorithm would prefer to assign this variable to zero satisfying more constraints. The percentage of variables which follow this bias is measured using property π_4.

The MC/MC constructs try to identify if the algorithm groups together all the highly constrained (and often more difficult) parts of the problem or are they spread throughout the solution. For example, in the scheduling problem, one can consider the fan-in and fan-out of the nodes in a single cycle. If the transitive fan-in and fan-out counts of the cycles are either very high or very low, we can assume that the algorithm groups together the most constraining operations into a single cycle. On the other hand, if the counts per clock cycle are often focused around a single value the algorithm aims at balancing the two. In the case of the partitioning problem, the number of nodes with high degrees in each partition can be considered.

Note that we choose not to use runtime as a property due to the fact that we assume that only the instances of the solutions are available for analysis, not the tool itself. However, in cases when the tool is available, runtime can be a key differentiating property.

5 Property Selection and Algorithm Clustering

In this section, we present our approach to algorithm classification. The first step is the identification of relevant properties of solutions. Once the properties are selected, we conduct algorithm clustering. The algorithm clustering task has two objectives: (i) to identify algorithms that are structurally similar, in a sense that their solution have same crucial characteristics indicating that they are using essentially identical mechanisms for optimization and (ii) to provide information required to determine which algorithm is used to produce a particular solution on a particular instance.

A property used for forensic engineering can be irrelevant for one of the following two reasons. The first is that it does not provide resolution capabilities, i.e. it does not facilitate differentiation between the algorithms. The second is that a property should not be considered if it is correlated with another property that provides better resolution capability.

The relevance of an individual property is evaluated in the following way. We use 100 different instances of the problem and at least three different algorithms for this step. We denote the solution of each instance for a given algorithm with a single letter that is unique for that algorithm. Next, we establish an ordering among the obtained values for all instances by sorting them in non-decreasing order. We then identify the leftmost and rightmost instance for each algorithm. The final step is to calculate how many letters that are not identical to the boundary letters are contained in each region that correspond to a given algorithm. We weight each letter that is not identical to the boundary letter by its distance to the closest boundary. If that number is higher than a user defined value, the property is eliminated from further consideration.

The information obtained in the above procedure is also used to establish the level of correlation for two properties. Specifically, we calculate the correlation level for two properties as the extent to which the regions labelled by identical letter are in the same range. The property that has better resolution capability is retained, if the correlation level is above a user specified threshold.

The second step is algorithm clustering. Once statistical data is collected, algorithms in the initial pool are partitioned into clusters. The goal of this partitioning is to join strategically similar algorithms (e.g. with similar properties) into a single cluster. We present this procedure formally using the pseudo-code in Figure 5.

The clustering process is initiated by setting the starting set of clusters to empty $C = \emptyset$. In order to associate an algorithm $A_x \in A$ with the original solution S_P, the set of algorithms is clustered according to the properties of S_P. For each property π_k of S_P we compute its feature quantifier $\pi_k(S_P) \rightarrow \omega_k^{S_P}$ and compare it to the collected pdfs of corresponding features χ_k^i of each considered algorithm $A_i \in A$. The clustering procedure is performed in the following way: two algorithms A_i, A_j remain in the same cluster, if the likelihood $z(A_i, A_j)$ that their properties are not correlated is greater than some predetermined bound $\epsilon \ll 1$.

$$z(A_i, A_j) = \prod_{k=1}^{|\pi|} \frac{2 \cdot \min(\Pr[\pi_k(A_i) \rightarrow \omega_k^i], \Pr[\pi_k(A_j) \rightarrow \omega_k^j])}{\Pr[\pi_k(A_i) \rightarrow \omega_k^i] + \Pr[\pi_k(A_j) \rightarrow \omega_k^j]}$$

The function that computes the mutual correlation of two algorithms takes into account the fact that two properties can be mutually dependent. Algorithm A_i is added to a cluster C_k if its correlation with all algorithms in C_k is greater than some predetermined bound $\epsilon \leq 1$. If A_i cannot be highly correlated with any algorithm from all existing clusters in C then a new cluster $C_{|C|+1}$ is created with A_i as its only member and added to C. If there exists a cluster C_k for which A_i is highly correlated with a subset C_k^H of algorithms within C_k, then C_k is partitioned into two new clusters $C_k^H \cup A_i$ and $C_k - C_k^H$. Finally, algorithm A_i is removed from the list of unprocessed algorithms A. These steps are

```
Given A, C = ∅
For each A_i ∈ A
  For each C_k ∈ C
    Similar = ∅
    For each A_j ∈ C_k
      If z(A_i, A_j) ≤ ε
      Then Similar = Similar ∪A_j
    End For
    /* A_i is similar to all A ∈ C_k */
    If |Similar| == |C_k| Then merge A_i with C_k
    /* A_i is not similar to any A ∈ C_k */
    Else If |Similar| == ∅
    Then create new cluster C_{k+1} with A_i as its only element.
    /* A_i is similar to a subset of A ∈ C_k */
    Else create two new clusters Similar ∪ A_i and C_k − Similar
  End For
End For
```

Figure 5: Pseudo-code for the algorithm clustering procedure.

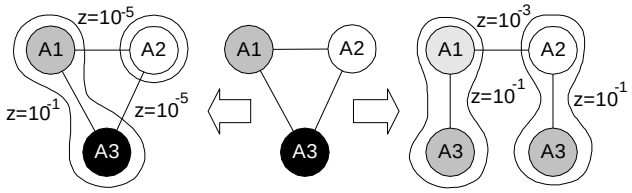


Figure 6: Two different examples of clustering three distinct algorithms. The first clustering (figure on the left) recognizes substantial similarity between algorithms A_1 and A_3 and substantial dissimilarity of A_2 with respect to A_1 and A_3. Accordingly, in the second clustering (figure on the right) the algorithm A_3 is recognized as similar to both algorithms A_1 and A_2, which were found to be dissimilar.

iteratively repeated until all algorithms are processed.

According to this procedure, an algorithm A_i can be correlated with two different algorithms A_j, A_k that are not mutually correlated (as presented in Figure 6). For instance, this situation can occur when an algorithm A_i is a blend of two different heuristics (A_j, A_k) and therefore its properties can be statistically similar to the properties of A_j, A_k. In such cases, exploration of different properties or more expensive and complex structural analysis of algorithm implementations is the only solution to detecting copyright infringement.

Obviously, according to this procedure, an algorithm A_i can be correlated with two different algorithms A_j, A_k that are not mutually correlated (as presented in Figure 6). For instance this situation can occur when an algorithm A_i is a blend of two different heuristics (A_i, A_k) and therefore its properties can be statistically similar to the properties of A_j, A_k. In such cases, exploration of different properties or more expensive and complex structural analysis of algorithm implementations is the only solution to detecting copyright infringement.

Finally, note that a natural way to define similarity between two algorithms A and B in our forensic engineering framework is consider the size of the overlap between their clusters in the property space. Specifically, if we denote the size of cluster C by $S(C)$, the similarity of two algorithms with corresponding clusters A and B is $\frac{S(A \cap B)}{S(A \cup B)}$. Furthermore, note that simultaneous analysis of all clusters yields a statistical estimate of the likelihood that a specific solution is produced using the considered algorithm. Specifically, this estimate for an algorithm A_1 for the forensic decision model built considering algo-

rithms A_i, $i = 1, ..., n$ can be obtain using the following formula: $\frac{S(A_1)}{\sum S(A_i)}$. Standard nonparametric test techniques, such as percentile method, can be used to obtained an interval of confidence for our decision process.

6 Experimental Results

In order to demonstrate the effectiveness of the proposed forensic methodologies, we have conducted a set of experiments on both common benchmarks and real-life problem instances. In this section, we present the obtained results for a large number of graph coloring and SAT instances.

The forensic analysis techniques for classifying algorithm used for creating solutions of SAT instances, have been tested using a real-life and abstract benchmark set of instances adopted from [23, 39]. Parts of the collected statistics are presented in Figure 7 and 8.

Figure 7 shows a histogram of the property values for π_1 for the NTAB, Rel_SAT, and WalkSAT algorithms. The value of the property is represented on the x-axis and the frequency of occurrence on the y-axis. The non-important variables property has multi-modal distribution for the algorithms. Subfigure (b) shows an enlarged portion of distribution. We can see that this property provides a clear distinction between the Rel_SAT and NTAB, as well as the WalkSAT and NTAB algorithms.

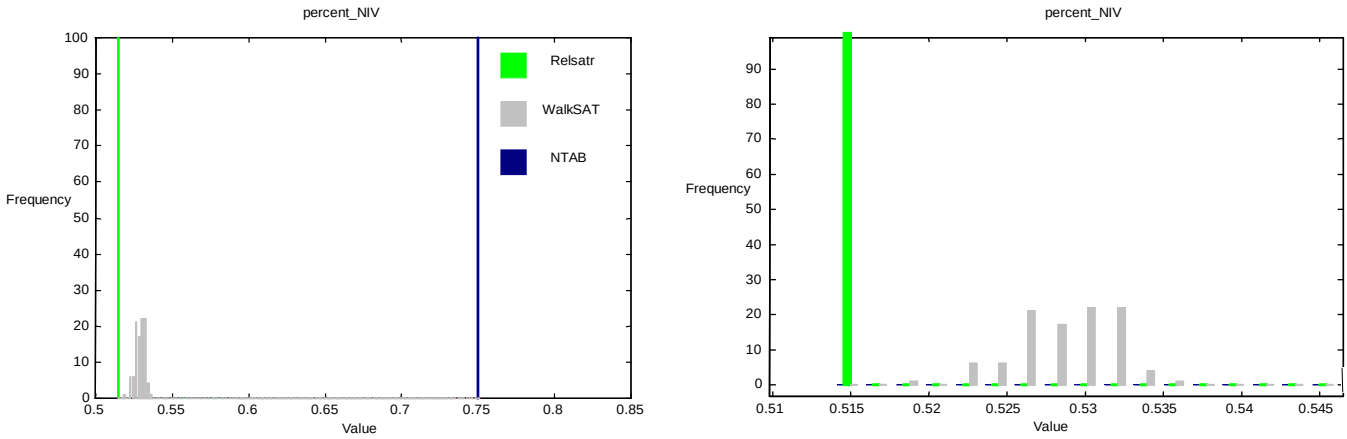
In Figure 8, we present histograms of the property values for π_2 (top row) and π_3 (bottom row) for the NTAB, Rel_SAT, and WalkSAT algorithms respectively. Each histogram presents the property values for 100 different solutions. Each multi-shaded column represents the frequency of the property values for all 100 solutions. For example, consider the histogram in the top left, clausal stability for the WalkSAT algorithm. For property values between 0 and 0.1, the frequency of these values occurring in each of the 100 solutions independently ranged between approximately 675 and 725.

The histograms indicate that solutions produced by NTAB can be easily distinguished from solutions generated by the other two algorithms using any of these properties. However, solutions created by Rel_SAT, and WalkSAT appear to be similar in structure (which is expected because they both use GSAT as the heuristic guidance for their prepositional search). Therefore, in order to distinguish between these two algorithms one must compare the property values for each instance separately. We have had success in distinguishing between the Rel_SAT and WalkSAT solutions on a case by case basis.

For example, in the Figure 7(b) it can be noticed that solutions generated by Rel_SAT have significantly wider range for π_1 and, therefore, according to the histogram, approximately 50% of its solutions can be easily distinguished from WalkSAT's solutions with high confidence. Significantly better results were obtained using another set of structurally different instances (zoomed comparison presented in the Figure 7(c)), where among 100 solution instances no overlap in the value of property π_1 was detected for Rel_SAT, and WalkSAT.

We have focused our forensic exploration of graph coloring solutions on two sets of instances: random (1000 nodes and uniform 0.5 edge existence probability) and register allocation graphs. The graphs in Figure 9 depict the histograms of property value distribution for the following pairs of algorithms and properties: DSATUR with backtracking vs. maxis for π_3, DSATUR with backtracking vs. tabu search for π_7, iterative greedy vs. maxis for π_1 and π_4, and maxis vs. tabu for π_1.

In order to evaluate the effectiveness of the GC properties, we compare the two histograms shown in the center row of the diagram. In this cases, the maxis algorithm is compared with the iterative greedy approach and tabu using the standard deviation of property π_1. In the both graphs, the maxis algorithm (with identical property values) is shown in white. From these two histograms we can conclude that if a given solution has a π_1 property value less than 1.8, the solution most likely was not produced by



(a) (b)

Figure 7: Property π_1 for SAT applied to NTAB, Rel_SAT, and WalkSAT

GC Solvers	bktdsat	maxis	tabu	itrgrdy
bkdsat	998	2	0	0
maxis	3	993	0	4
tabu	1	0	995	4
itrgrdy	1	2	0	997

Table 1: Experimental Results: graph coloring. Statistics for each solver were established. The thousand instances were than classified using these statistical measures.

the maxis algorithm. However if this property has value between 1.8 and 2.1, it is highly unlikely the solution is generated using the tabu algorithm. Therefore, by combining this property with the other properties, we can classify the algorithms with higher accuracy.

Each of the diagrams can be used to associate a particular solution with one of the two algorithms A_1 and A_2 with 1% accuracy (100 instances attempted for statistics collection). For a given property value $\omega_i^{A_j} = x, j = 1, 2$ (x-axis), a test instance can be associated to algorithm A_1 with likelihood equal to the ratio of the pdf values (y-axis) $z(A_1, A_2)$. For the complete set of instances and algorithms that we have explored, as it can be observed from the diagrams, on the average, we have succeeded to associate 99% of solution instances with their corresponding algorithms with probability greater than 0.95. In one half of the cases, we have achieved association likelihood better than $1 - 10^{-6}$.

SAT Solvers	WalkSAT	RelSATR	NTAB
WalkSAT	992	5	3
RelSATR	6	990	4
NTAB	0	2	998

Table 2: Experimental Results: boolean satisfiability. A thousand test cases were used. A thousand test cases were used. Statistics for each solver were established. The thousand instances were than classified using these statistics.

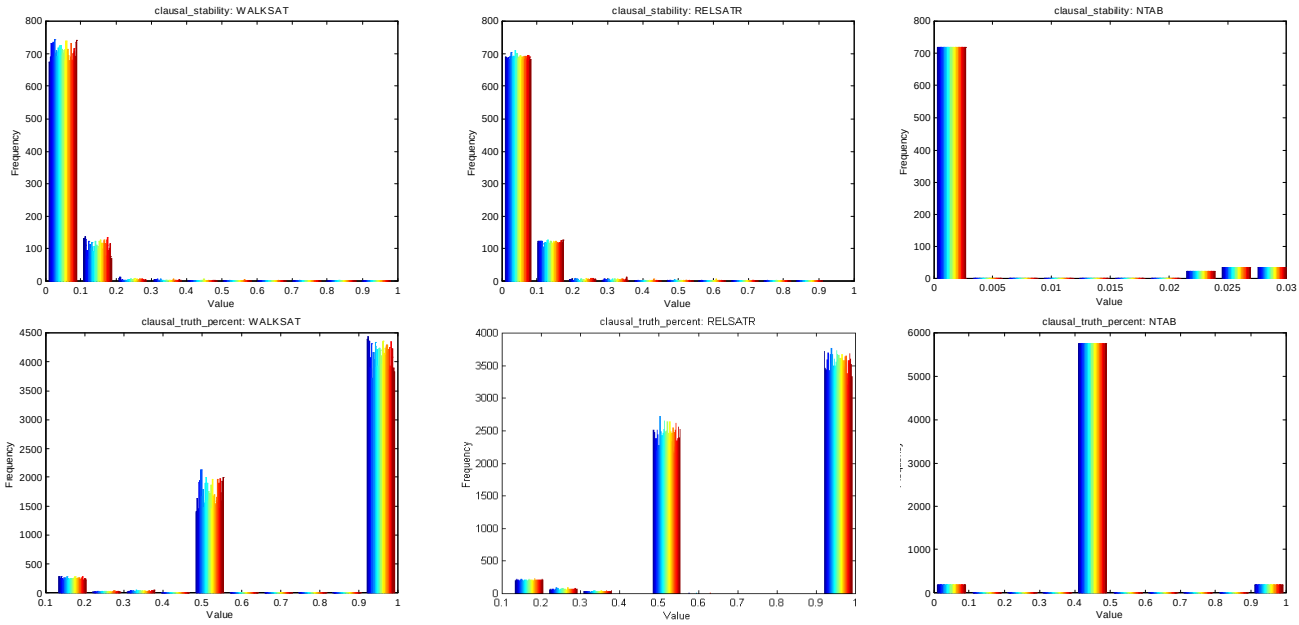


Figure 8: Properties π_2 (top row) and π_3 (bottom row) applied to NTAB, Rel_SAT, and WalkSAT

Using statistical methods, we obtained Table 1 and 2. A thousand test cases were classified using the developed forensic engineering technique. The rows of the tables indicate the solver used to produce the thousand test cases. The columns indicate the classification of the solution using the forensic engineering technique. In all cases more than 99% of the solutions were classified according to their original solvers with probability higher than 0.95. We see that the graph coloring algorithms differ in many of the features, which resulted in very little overlap in the statistics. In the case of boolean satisfiability, both WalkSAT and Rel_SAT_rand are based on the GSAT algorithm which accounts for the slightly higher misclassification rate between the two algorithms.

7 Conclusion

Copyright enforcement has become one of the major obstacles to intellectual property (hardware and software) e-commerce. We propose a forensic engineering technique that addresses the generic copyright enforcement scenario. Specifically, given a solution S_P to a particular optimization problem instance P and a finite set of algorithms A applicable to P, the goal is to identify with certain degree of confidence the algorithm A_i which has been applied to P in order to obtain S_P. The application of the forensic analysis principles to graph coloring and boolean satisfiability has demonstrated that solutions produced by strategically different algorithms can be associated with their corresponding algorithms with high accuracy. Since both graph coloring and boolean satisfiability are common steps in hardware synthesis and software compilation, we implicitly demonstrated the effectiveness of forensic engineering for authorship identification of IP.

References

[1] R. Anderson, M. Kuhn. “Tamper Resistance - a Cautionary Note”, Proceedings of the Second Usenix Workshop on Electronic Commerce, pp. 1-11, 1996.

[2] B.S. Baker and U. Manber. Deducing similarities in Java sources from bytecodes. USENIX Technical Conference, pp.179-190, 1998.

[3] R.J. Bayardo and R. Schrag. Using CSP look-back techniques to solve exceptionally hard SAT instances. Principles and Practice of Constraint Programming, pp.46-60, 1996.

[4] B.C. Behrens and R.R. Levary. Practical legal aspects of software reverse engineering. Comm. of the ACM, vol.41, (no.2), pp.27-29, 1998.

[5] B. Bollobas and A. Thomason. Random graphs of small order. Random Graphs '83. Annals of Discrete Mathematics vol. 28, pp.47-97, 1985.

[6] D. Brelaz. New methods to color the vertices of a graph. Comm. of the ACM, vol.22, (no.4), pp.251-256, 1979.

[7] S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. SIGMOD Record, vol.24, (no.2), pp.398-409, 1995.

[8] R.E. Bryant. Binary decision diagrams and beyond: enabling technologies for formal verification. ICCAD, pp. 236-243, 1995.

[9] E. Charbon and I. Torunoglu. Watermarking layout topologies. ASP-DAC, pp.213-216, 1999.

[10] C.S. Collberg and C. Thomborson. Software Watermarking: Models and Dynamic Embeddings. Symposium on Principles of Programming Languages, pp. 311-324, 1999.

[11] J.M. Crawford. Solving Satisfiability Problems Using a Combination of Systematic and Local Search. Second DIMACS Challenge, 1993.

[12] http://www.cs.ualberta.ca/∼joe

[13] M. Davis, H. Putnam. A computing procedure for quantification theory. Journal of the ACM, pp.201-215, 1960.

[14] M. DeGroot. Probability and Statistics. Addison-Wesley, Reading, 1989.

[15] S. Devadas. Optimal layout via Boolean satisfiability. Internationsl Conference on Computer-Aided Design, pp.294-297, 1989.

[16] B. Efron, R. Tibshirani. An introduction to the bootstrap. 1993.

[17] M.R. Garey and D.S. Johnson. Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, San Francisco, 1979.

[18] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. of the ACM, vol.42, (no.6), pp.1115-1145, 1995.

[19] D. Grover. Forensic copyright protection. Computer Law and Security Report, vol. 14, (no.2), pp.121-122, 1998.

[20] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. Computing, vol.39, no.4, pp.345-351, 1987.

[21] I.T. Jolliffe. Principal component analysis. New York, Springer-Verlag, 1986.

[22] A.B. Kahng et al. Robust IP Watermarking Methodologies for Physical Design. Design Automation Conference, pp. 782-787, 1998.

[23] A.P. Kamath, et al. An interior point approach to Boolean vector function synthesis. Midwest Symposium on Circuits and Systems, pp.185-9, 1993.

[24] H. Karhunen. Ueber lineare Methoden in der Wahrscheinlichkeitsrechnung. Ann. Acad. Sci. Fenn. AI, 37, 1947.

[25] D. Kirovski and M. Potkonjak. Efficient coloring of a large spectrum of graphs. DAC, pp. 427-32, 1998.

[26] H. Konuk and T. Larrabee. Explorations of sequential ATPG using Boolean Satisfiability. IEEE VLSI Test Symposium, pp.85-90, 1993.

[27] J. Kumagai. "Chip Detectives", IEEE Spectrum, Vol. 37, no. 11, pp. 43-49,2000.

[28] F.T. Leighton. A Graph Coloring Algorithm for Large Scheduling Algorithms. Journal of Res. Natl. Bur. Standards, vol.84, pp.489-506, 1979.

[29] M. Loeve. Fonctions Aleatoire de Seconde Ordre. Hermann. Paris. 1948.

[30] D.F. McGahn. Copyright infringement of protected computer software: an analytical method to determine substantial similarity. Rutgers Computer & Technology Law Journal, vol. 21, (no.1), pp.88-142, 1995.

[31] G. Qu and M. Potkonjak. Analysis of watermarking techniques for graph coloring problem. ICCAD, 1998.

[32] N. Rudin, K. Inman, G. Stolvitzky, and I. Rigoutsos. DNA Based Identification. BIOMETRICS personal Identification in Networked Society, Kluwer, 1998.

[33] B. Selman, H.J. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. National Conference on Artificial Intelligence, pp. 440-446, 1992.

[34] B. Selman et al. Local Search Strategies for Satisfiability Testing. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993.

[35] B. Selman. Stochastic search and phase transitions: AI meets physics. IJCAI, pp. 998-1002, vol.1, 1995.

[36] J.P. Marques-Silva and K.A. Sakallah. GRASP: a search algorithm for propositional satisfiability. T. on Computers, vol.48, (no.5), pp. 506-521, 1999.

[37] P. Stephan, et al. Combinational test generation using satisfiability. Transactions on Computer-Aided Design of Intergrated Circuits and Systems, vol.15, (no.9), pp. 1167-1176, 1996.

[38] R. Thisted and B. Efron Did Shakespeare Write a newly discovered Poem? Biometrika, 74, pp. 445-455, 1987.

[39] Y. Tsuji and A. Van Gelder. Incomplete thoughts about incomplete satisfiability procedures. Proceedings of the 2nd DIMACS Challenge, 1993.

[40] D. Welsh and M. Powell. An upper bound for the chromatic number of a graph and its applications to timetabling problems. The Computer Journal, vol.10, pp.85-86, 1967.

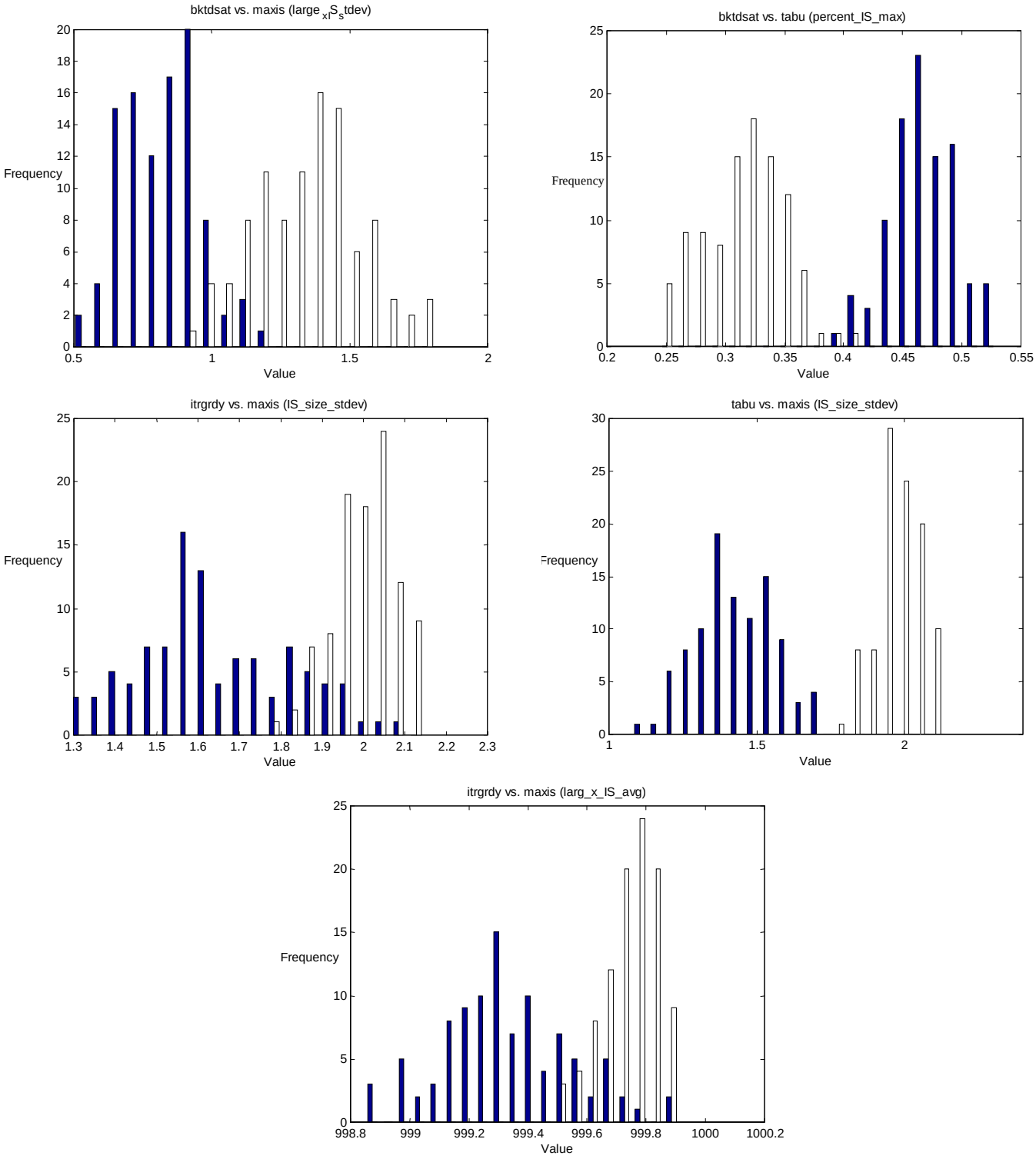


Figure 9: Histograms of property value distribution for Algorithm a (black) vs. Algorithm b (white) and properties: DSATUR with backtracking vs. maxis for π_3, DSATUR with backtracking vs. tabu search for π_7, iterative greedy vs. maxis for π_1 and π_4, and maxis vs. tabu for π_1.