# Computational Forensic Techniques for Intellectual Property Protection

Jennifer L. Wong[1], Darko Kirovski[2], and Miodrag Potkonjak[1]

[1] Computer Science Department, University of California, Los Angeles, CA 90095
[2] 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). 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 pool 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 solutions. The developed CFE tool identifies one 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 Related Work

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 [Rud99]. 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 [Thi87]. 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 [McG95]. In the same journal paper, McGahn introduces a new analytical method, based on Learned Hand's abstractions test, which allows courts to rely their decisions on well established and familiar principles of copyright law. Grover presents the details behind an example lawsuit case [Gro98] 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 [Bak98] and to perform identity or partial copy detection for digital libraries [Bri95]. 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 [Cha99], partitioning [Kah98], and behavioral specification [Qu98] level. In the software domain, good survey of techniques for copyright protection of programs has been presented by Collberg and Thomborson [Col99]. They have also developed

a code obfuscation method which aims at hiding watermarks in program's data structures.

### 2.1 Existing Methods for Establishing Copyright Infringement

In this subsection, we present an overview of techniques used in court to distinguish substantial similarity between a copyright protected design or program and its replica.

The dispositive issue in copyright law is the idea-expression dichotomy, which specifies that any idea (system) of operation (concept), regardless of the form in which it is described, is unprotectable [McG95]. Copyright protection extends only to the expression of ideas, not the ideas themselves. Although courts have fairly effective procedures for distinguishing ideas from expressions [McG95], they lack persuasive methods for quantifying substantial similarity between expressions; a necessary requirement for establishing a case of copyright infringement. Since modern reverse engineering techniques have made both hardware and software [Beh98] vulnerable to partial resynthesis, frequently, plaintiffs have problems identifying the degree of infringement.

Methods used by courts to detect infringement are currently still rudimentary. The widely adopted "iterative approach" enables better abstraction of the problem by requiring: (i) substantial similarity and a proof of copying or access and (ii) proof that the infringing work is an exact duplication of substantial portions of the copyrighted work [McG95]. The test does not address the common case in contemporary industrial espionage, where stolen IP is either hard to abstract from synthesized designs or difficult to correlate to the original because of a number of straightforward modifications which are hard to trace back.

## 3 Forensic Engineering: The Generic Approach

Forensic engineering aims at providing both qualitative and quantitative evidence of substantial similarity between the design original and its copy. 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 $S_P$. 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. The global flow of the generic forensic engineering approach consists of four fully modular phases:

**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. The properties are primarily obtained by analyzing solutions produced by various algorithms and identifying common

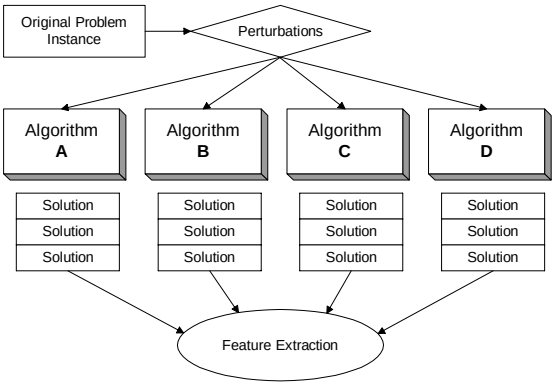


**Fig. 1.** Flow of the Generic Forensic Engineering Approach.

features in solutions produced by a particular algorithm. For example, the Graph Coloring RLF algorithm [Lei79], 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 pdfs for this feature.

Third, we discard features for which all considered algorithms display equivalent statistics. A feature is considered as viable only if at least two algorithms have statistically distinct pdfs for 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 [Loe48, Kar47, Jol86]. 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.

Finally, we apply fast algorithms for extraction of the selected properties. In some cases, extracting the features from the solutions is trivial. But in other cases, it can be complex and time consuming. The second subphase is instance preprocessing. We make order and lexical perturbations of an instance. This is done to avoid any dependencies an algorithm may have on the naming or form of the input, such as variable labels.

**Feature Extraction.** We run all the perturbed instances through each of the algorithms. Once we obtain all the solutions, we extract the features from them.

**Algorithm clustering.** We place the features into $n$-dimensional space, and cluster the results.

**Validation.** Our final step is the application of non-parametric resubstitution software [Efr93] to establish the validity of our ability to distinguish distinct algorithms. Specifically, we run five hundred resubstituitions of 80% of the sample points. Now, when a new solution is available, the generic flow and tools fully and automatically determine which algorithm was used.

Figure 1 demonstrates the processing flow of the generic technique. The first level of flow chart shows the perturbation of a given instance in order to generate unbiased instances of the problem. Next, the instances are run on each of the candidate algorithms A, B, C, and D and the results are collected. From the solutions, we extract relevant features. Given a large number of solutions for each of the algorithm, this process can take significant amount of time. Once the statistical and pattern features have been collected, we perform algorithm clustering.

## 4 Forensic Engineering: Statistics Collection

### 4.1 Graph Coloring

We present the developed forensic engineering methodology using the problem of graph $K$-colorability. It can be formally described 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 [Gar79]. Due to its applicability, a number of exact and heuristic algorithms for graph coloring has been developed to date. For brevity and due to limited source code availability, in this paper, we constrain the pool of algorithms $A$ to a greedy, DSATUR, MAXIS (RLF based), backtrack DSATUR, iterated greedy, and tabu search (descriptions and source code at [Cul99]).

The simplest constructive algorithm for graph coloring is the "sequential" coloring algorithm (SEQ). SEQ sequentially traverses and colors vertices with the lowest index not used by the already colored neighboring vertices. DSATUR [Bre79] colors the next vertex with a color $C$ selected depending on the number of neighbor vertices already connected to nodes colored with $C$ (saturation degree) (Figure 2). RLF [Lei79] colors the vertices sequentially one color class at a time. Vertices colored with one color represent an independent subset (IS) of the graph. The algorithm tries to color with each color maximum number of vertices. Since the problem of finding the maximum IS is intractable [Gar79], a heuristic is employed to select a vertex to join the current IS as the one with the largest number of neighbors already connected to that IS. An example how RLF colors graphs is presented in Figure 2. Node 6 is randomly selected as the first node in the first IS. Two nodes (2,4) have maximum number of neighbors which are also

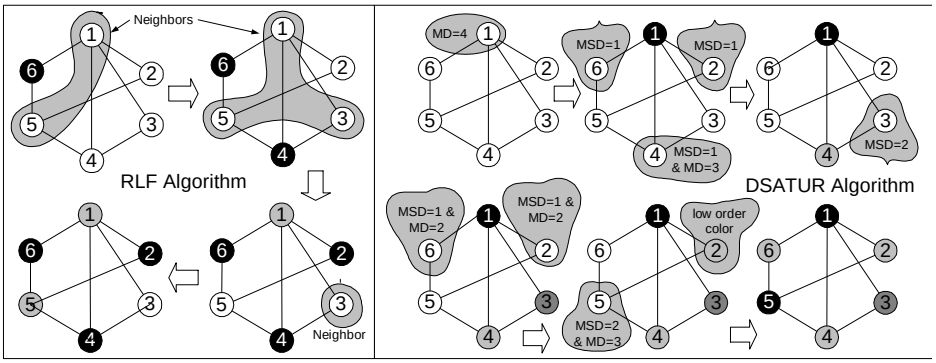


**Fig. 2.** Example of how RLF and DSATUR algorithms create their solutions. $MD$ - maximal degree; $MSD$ - maximal saturation degree.

neighbors to the current IS. The node with the maximum degree is chosen (4). Node 2 is the remaining vertex that can join the first IS. The second IS consists of randomly selected node 1 and the only remaining candidate to join the second IS, node 5. Finally, node 3 represents the last IS.

Iterative improvement techniques try to find better colorings through generating successive colorings by random moves. The most common search techniques are simulated annealing and tabu search [dWe85, Fle96]. In our experiments, we will constrain XIS (RLF based), backtrack DSATUR, iterated greedy, and tabu search (descriptions and source code at [Cul99]).

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

- [$\pi_1$] **Color class size.** 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 take 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 acheive 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 used coloring algorithm
- [$\pi_2$] **Number of edges in large independent sets.** This property is used to aid the accuracy of $\pi_1$ by excluding easy-to-find large independent sets from consideration in the analysis. We use k% of the largest sets and measure the percentage of edges leaving the IS.
- [$\pi_3$] **Number of edges that can switch color classes.** This criteria analyzes the quality of the coloring. Good (in a sense of being close to a local minima) coloring result will have fewer nodes that are able to switch color classes. It also characterizes the greediness of an algorithm because greedy algorithms commonly create at

the end of their coloring process many color classes that can absorb large portion of the remaining graph. The percentage of nodes which can switch colors versus the number of nodes is used.

[$\pi_4$] **Color saturation in neighborhoods.** This property assumes creation of 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 is used as well as the average value to quantify 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.

[$\pi_5$] **Sum of degrees of nodes included in the smallest color classes.** The analysis goal of this property is similar to $\pi_5$ with the exception that it focuses on selecting algorithms that perform neighborhood look ahead techniques [Kir98]. The values are normalized against the average value and both the average value and the Gini Coefficient are used.

[$\pi_7$] **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 $\pi_3$, it aims at detecting fine nuances among similar RLF-like algorithms.

The itemized properties can be effective only on large instances where the standard deviation of histogram values is relatively small. Using standard statistical approaches [DeG89], the function of standard deviation for each histogram can be used to determine the standard error in the reached conclusion.

Although instances with small cardinalities cannot be a target of forensic methods, 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 RLF. Specified algorithms color the graph constructively in the order denoted in the figure. If property $\pi_1$ 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 RLF results in $\chi_{\pi_1}^{RLF} = \{2_2, 0_3, 1_4\}$. Commonly, extreme values point to the optimization goal of the algorithm or characteristic structure property of its solutions. In this case, RLF has found a maximum independent set of cardinality $y = 4$, a consequence of algorithm's strategy to search in a greedy fashion for maximal ISs.

### 4.2 Boolean Satisfiability

We illustrate the key ideas of watermarking-based IPP techniques using the SAT problem. The SAT problem can be defined in the following way [Gar79].

**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 $C$?*

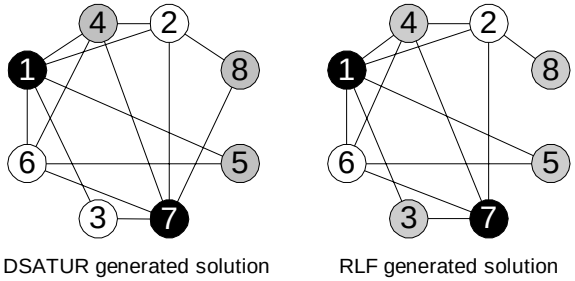


**Fig. 3.** Example of two different graph coloring solutions obtained by two algorithms DSATUR and RLF. The index of each vertex specifies the order in which it is colored according to a particular algorithm.

Boolean Satisfiability is an NP-complete problem [Gar79]. It has been proven that every other problem in NP can be polynomially reduced to Satisfiability [Gar79]. There are at least three broad classes of solution strategies for the SAT problem. For example, SAT techniques have been used in testing [Ste96, Kon93], logic synthesis, and physical design [Dev89]. There are at least three broad classes of solution strategies for the SAT problem. The first class of techniques are based on probabilistic search [Sil99, Sel95], the second are approximation techniques based on rounding the solution to a nonlinear program relaxation [Goe95], and the third is a great variety of BDD-based techniques [Bry95]. For brevity and due to limited source code availability, we demonstrate our forensic engineering technology on the following SAT algorithms.

- **GSAT** 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 [Sel92]. The algorithm pseudo-randomly flips assignments of variables with the greatest DIFF.
- **WalkSAT** 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 [Sel93a].
- **NTAB** performs a local search to determine weights for the clauses, intuitively giving higher weights corresponds to clauses which are harder to satisfy. The clause weights are then used to preferentially branch on variables that occur more often in clauses with higher weights [Cra93].
- **Rel_SAT_rand** represents an enhancement of GSAT with look-back techniques [Bay96].

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

[$\pi_1$] **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 impact on 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.

[$\pi_2$] **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.

[$\pi_3$] **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.

[$\pi_4$] **Ratio of coverage using positive and negative appearance of a variable.** While property $\pi_3$ analyzes the solution from a perspective of a single clause, This property analyzes the solution from a 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 $t(v_i) = p_i \geq n_i$.

[$\pi_5$] **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.

## 5 Algorithm Clustering and Decision Making

Once statistical data is collected, algorithms in the initial pool are partitioned into clusters. The goal of partitioning is to join strategically similar algorithms (e.g. with similar properties) into a single cluster. This procedure is presented formally using the pseudo-code in Figure 4.

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 $\pi_k$ 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 $\chi_k^i$ 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 iteratively repeated until all algorithms are processed.

```
Given A. C = ∅.
For each A_i ∈ A
  For each C_k ∈ C
    add = true; none = true
    For each A_j ∈ C_k
      If z(A_i, A_j) ≤ ε.
      Then add = false Else none = false
    End For
    If add Then merge A_i with C_k
    Else create new cluster C_{|C|+1} with
    A_i as its only element.
    If none Then create two new clusters
    C_k^H ∪ A_i and C_k − C_k^H where C_k^H ∈ C_k
    is a subset of algorithms highly correlated with A_i.
  End For
End For
```

**Fig. 4.** Pseudo-code for the algorithm clustering procedure.

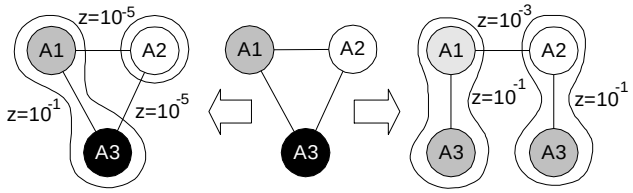


**Fig. 5.** Two different examples of clustering three distinct algorithms. The first clustering (figure on the left) recognizes substantial similarity between algorithms $A_1$ and $A_3$ and substantial dissimilarity of $A_2$ with respect to $A_1$ and $A_3$. Accordingly, in the second clustering (figure 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.

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 5). 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. Once the algorithms are clustered, the decision making process is straightforward:

**If** plaintiff's algorithm $A_x$ is clustered jointly with the defendant's algorithm $A_y$ (e.g. its solution $S_P$)
**and** $A_y$ is not clustered with any other algorithm from $A$ which has been previously determined as strategically different,
**then** substantial similarity between the two algorithms is positively detected at a degree quantified using the parameter $z(A_x, A_y)$.

The court may adjoin to the experiment several slightly modified replicas of $A_x$ as well as a number of strategically different algorithms from $A_x$ in order to validate that the value of $z(A_x, A_y)$ points to the correct conclusion.

## 6 Experimental Results

In order to demonstrate the effectiveness of the proposed forensic methodologies, we have conducted a set of experiments on both abstract and real-life problem instances. In this section, we present the obtained results for a large number of graph coloring and SAT instances. The collected data is partially presented in Figure 6. It is important to stress, that for the sake of external similarity among algorithms, we have adjusted the run-times of all algorithms such that their solutions are of approximately equal quality.

We have focused our forensic exploration of graph coloring solutions on two sets of instances: random (1000 nodes and 0.5 edge existence probability) and register allocation graphs. The last five subfigures in Figure 6 depict the histograms of property value distribution for the following pairs of algorithms and properties: DSATUR with backtracking vs. maxis and $\pi_3$, DSATUR with backtracking vs. tabu search and $\pi_7$, iterative greedy vs. maxis and $\pi_1$ and $\pi_4$, and maxis vs. tabu and $\pi_1$ respectively.

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 $1 - 10^{-6}$.

The forensic analysis techniques, that we have developed for solutions to SAT instances, have been tested using a real-life (circuit testing) and an abstract benchmark set of instances adopted from [Kam93, Tsu93]. Parts of the collected statistics are presented in the first ten subfigures in Figure 6. The subfigures represent the following comparisons: $\pi_1$ and NTAB, Rel_SAT, and WalkSAT and then zoomed version of the same property with only Rel_SAT, and WalkSAT (for two different sets of instances - total: first four subfigures), $\pi_2$ for NTAB, Rel_SAT, and WalkSAT, and $\pi_3$ for NTAB, Rel_SAT, and WalkSAT respectively.

The diagrams clearly indicate that solutions provided by NTAB can be easily distinguished from solutions provided by the other two algorithms using any of the three properties. However, solutions provided 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). We have succeeded to differentiate their solutions on per instance basis. For example, in the second subfigure it can be noticed that solutions provided by Rel_SAT have much wider range for $\pi_1$ and therefore, according to the second subfigure, approximately 50% of its solutions can be easily distinguished from WalkSAT's solutions with high probability. Significantly better results were obtained using another set of structurally different instances where among 100 solution instances no overlap in the value of property $\pi_1$ was detected for 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 statistics.

| 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.

Using statistical methods, we obtained Table 1 and 2. A thousand test cases were classifiesd using the statistical data. The rows of the tables represent the solver in which the thousand test cases originated from. The columns represent the classification of the solution using the statistical methods. In all cases more than 99% of the solutions were classified according to their original solvers with probability higher than 0.95. 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 numbers when classifying 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.

## 8 References


[Bak98] B.S. Baker and U. Manber. Deducing similarities in Java sources from bytecodes. USENIX Technical Conference, pp.179-190, 1998.

[Bay96] 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.

[Beh98] 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.

[Bre79] D. Brelaz. New methods to color the vertices of a graph. Comm. of the ACM, vol.22, (no.4), pp.251-256, 1979.

[Bri95] S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. SIGMOD Record, vol.24, (no.2), pp.398-409, 1995.

[Bry95] R.E. Bryant. Binary decision diagrams and beyond: enabling technologies for formal verification. ICCAD, pp. 236-243, 1995.

[Cha99] E. Charbon and I. Torunoglu. Watermarking layout topologies. ASP-DAC, pp.213-216, 1999.

[Col99] C.S. Collberg and C. Thomborson. Software Watermarking: Models and Dynamic Embeddings. Symposium on Principles of Programming Languages, pp. 311-324, 1999.

[Cra93] J.M. Crawford. Solving Satisfiability Problems Using a Combination of Systematic and Local Search. Second DIMACS Challenge, 1993.

[Cul99] http://www.cs.ualberta.ca/~joe

[DeG89] M. DeGroot. Probability and Statistics. Addison-Wesley, Reading, 1989.

[Dev89] S. Devadas. Optimal layout via Boolean satisfiability. Internationsl Conference on Computer-Aided Design, pp.294-297, 1989.

[Efr93] B. Efron, R. Tibshirani. An introduction to the bootstrap. 1993.

[Fle96] C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. Annals of Operations Research, vol.63, pp.437-461, 1996.

[Gar79] M.R. Garey and D.S. Johnson. Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, San Francisco, 1979.

[Goe95] 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.

[Gro98] D. Grover. Forensic copyright protection. Computer Law and Security Report, vol. 14, (no.2), pp.121-122, 1998.

[Jol86] I.T. Jolliffe. Principal component analysis. New York, Springer-Verlag, 1986.

[Kah98] A.B. Kahng et al. Robust IP Watermarking Methodologies for Physical Design. Design Automation Conference, pp. 782-787, 1998.

[Kar47] H. Karhunen. Ueber lineare Methoden in der Wahrscheinlichkeitsrechnung. Ann. Acad. Sci. Fenn. AI, 37, 1947.

[Kir98] D. Kirovski and M. Potkonjak. Efficient coloring of a large spectrum of graphs. DAC, pp. 427-32, 1998.

[Kon93] H. Konuk and T. Larrabee. Explorations of sequential ATPG using Boolean Satisfiability. IEEE VLSI Test Symposium, pp.85-90, 1993.

[Lei79] F.T. Leighton. A Graph Coloring Algorithm for Large Scheduling Algorithms. Journal of Res. Natl. Bur. Standards, vol.84, pp.489-506, 1979.

[Loe48] M. Loeve. Fonctions Aleatoire de Seconde Ordre. Hermann. Paris. 1948.

[McG95] 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.

[Qu98] G. Qu and M. Potkonjak. Analysis of watermarking techniques for graph coloring problem. ICCAD, 1998.

[Rud99] N. Rudin, K. Inman, G. Stolvitzky, and I. Rigoutsos. DNA Based Identification. BIOMETRICS personal Identification in Networked Society, Kluwer, 1998.

[Sel92] 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.

[Sel93a] B. Selman et al.Local Search Strategies for Satisfiability Testing. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993.

[Sel95] B. Selman. Stochastic search and phase transitions: AI meets physics. IJCAI, pp. 998-1002, vol.1, 1995.

[Sil99] 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.

[Ste96] 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.

[Thi87] R. Thisted and B. Efron Did Shakespeare Write a newly discovered Poem? Biometrika, 74, pp. 445-455, 1987.

[dWe85] D. de Werra. An Introduction to Timetabling. European Journal of Operations Research, vol.19, pp.151-162, 1985.

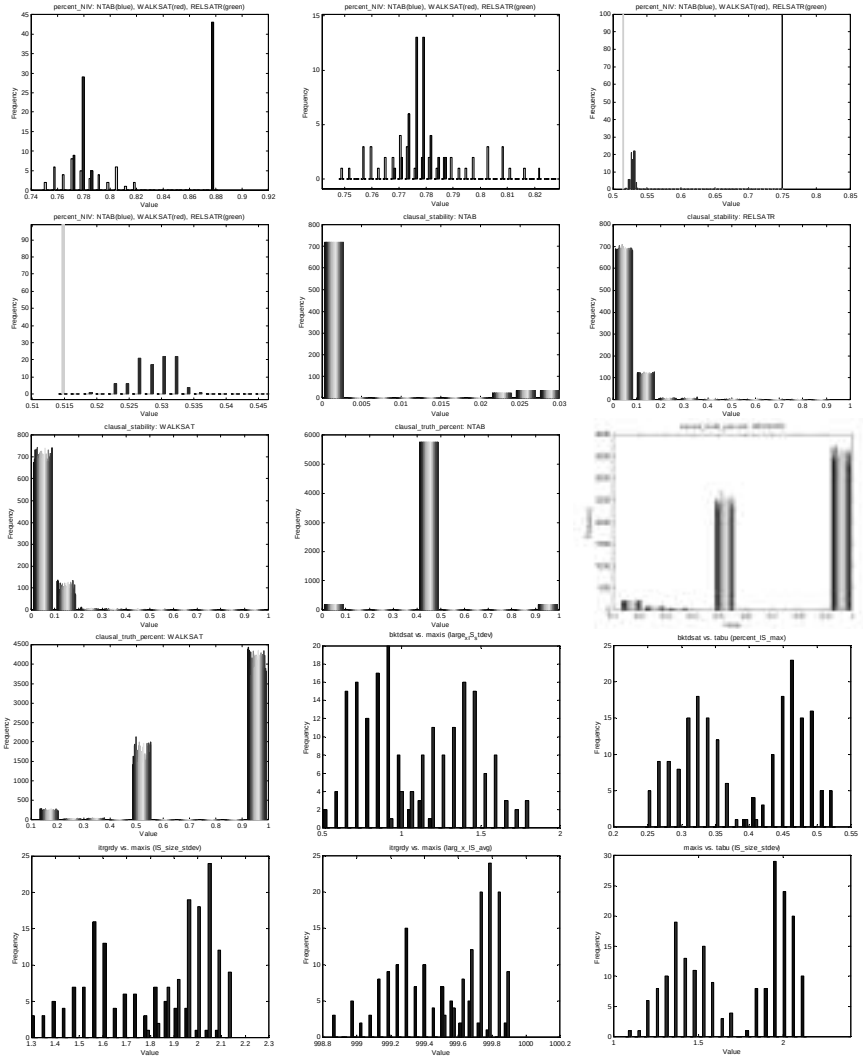


**Fig. 6.** Each subfigure represents the following comparison (from upper left to bottom right): (1,3) $\pi_1$ and NTAB, Rel_SAT, and WalkSAT and (2,4) then zoomed version of the same property with only Rel_SAT, and WalkSAT, (5,6,7) $\pi_2$ for NTAB, Rel_SAT, and WalkSAT, and (8,9,10) $\pi_3$ for NTAB, Rel_SAT, and WalkSAT respectively. The last five subfigures depict the histograms of property value distribution for the following pairs of algorithms and properties: (11) DSATUR with backtracking vs. maxis and $\pi_3$, (12) DSATUR with backtracking vs. tabu search and $\pi_7$, (13,14) iterative greedy vs. maxis and $\pi_1$ and $\pi_4$, and (15) maxis vs. tabu and $\pi_1$.