Forensic Engineering Techniques for VLSI CAD Tools

David Liu, Jennifer Wong, Darko Kirovski, and Miodrag Potkonjak
Computer Science Department, University of California, Los Angeles

Abstract
The proliferation of the Internet has affected the business model of almost all semiconductor and VLSI CAD companies that rely on intellectual property (IP) as their main source of revenues. The fact that IP has become more accessible and easily transferrable, has influenced the emergence of copyright infringement as one of the most common obstructions to e-commerce of IP.

In this paper, we propose a generic forensic engineering technique that addresses a number of copyright infringement scenarios. 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 the algorithm \( A_i \) which has been applied to \( P \) in order to obtain \( S_P \). We have applied forensic analysis principles to two problem instances commonly encountered in VLSI CAD: graph coloring and boolean satisfiability. We have demonstrated that solutions produced by strategically different algorithms can be associated with their corresponding algorithms with high accuracy.

1 Introduction
The emergence of the Internet as the global communication paradigm, has enforced almost all semiconductor and VLSI CAD companies to market their intellectual property online. Currently, companies such as ARM Holdings [Arm99], LSI Logic [Lsi99], and MIPS [Mip99], mainly constrain their on-line presence to sales and technical support. However, in the near future, it is expected that both core and synthesis tools developers place their IP on-line in order to enable modern hardware and software licensing models. There is a wide consensus among the software giants (Microsoft, Oracle, Sun, etc.) that the rental of downloadable software will be their dominating business model in the new millennium [Mc99]. It is expected that similar licensing models become widely accepted among VLSI CAD companies.

Most of the CAD companies planning on-line IP services believe that copyright infringement will be the main negative consequence of IP exposure. This expectation has its strong background in an already "hot" arena of legal disputes in the industry. In the past couple of years, a number of copyright infringement lawsuits have been filed: Cadence vs. Avant! [EET99], Symantec vs. McAfee [IW99], Gambit vs. Silicon Valley Research [GCW99], and Verity vs. Lotus Development [IDG99]. In many cases, the concerns of the plaintiffs were related to the violation of patent rights frequently accompanied with misappropriation of implemented software or hardware libraries. Needless to say, court rulings and secret settlements have impacted the market capitalization of these companies enormously. In many cases, proving legal obstruction has been a major obstacle in reaching a fair and convincing verdict [Mot99, Afc99].

In order to address this important issue, we propose a set of techniques for the forensic analysis of design solutions. Although the variety of copyright infringement scenarios is broad, we target a relatively generic case. The goal of our generic paradigm is to identify 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 three steps: collection of statistical data, clustering of heuristic properties for each analyzed algorithm, and decision making with confidence quantification.

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. 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 corresponding algorithms with relatively high accuracy.

2 Related Work
We trace the related work along the following lines: copyright enforcement policies and law practice, forensic analysis of software and documents, steganography, and code obfuscation. 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 [Mc95]. 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 technology researchers. To the best knowledge of the authors, to date, forensic techniques have been explored for detection of authentic Java bytecodes [Bak98] and to perform identity or partial copy detection for digital libraries [Bir93]. Recently, researchers have endorsed steganography and code obfuscation techniques as viable strategies for content and design protection. Protocols for watermarking active IP have been developed at the physical layout [Cha99, Wai98], partitioning [Kah98], logic synthesis [Ola99, Kir98], partial scan selection [Kit98], and behavioral specification [Qui95, Hon98] level. A routing-level approach for fingerprinting FPGA digital designs has been introduced [Lac98]. 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 pro-
gram's data structures.

Although steganography has demonstrated its potential to protect software and hardware implementations, its applicability to algorithm protection is still an unsolved issue. In order to provide a foundation for associating algorithms with their creations, in this paper, for the first time, we present a set of techniques which aim at detecting copyright infringement by giving quantitative and qualitative analysis of the algorithm-solution correspondence.

3 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 [Tae99] and software [Beh88] vulnerable to partial synthesis, frequently, plaintiffs have problems identifying the degree of infringement.

Methods used by courts to detect infringement are currently still rudimentary. The three most common tests: the "ordinary observer test", the extrinsic/intrinsic test, and the "total concept and feel test" are used in cases when it is easy to detect a complete copy of a design or a program's source code [McG95]. 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]. Obviously, neither of the tests addresses the common case in contemporary industrial espionage, where stolen IP is either performed peephole optimizations can alter a solution to the original problem instance

4 Forensic Engineering: The New Generic Approach

In this section, we introduce generic forensic engineering techniques that can be used to obtain fair rulings in copyright infringement cases. 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 is presented in Figure 1. It consists of three fully modular phases:

Statistics collection. Initially, each algorithm \( A_i \in A \) is applied to a large number of isomorphic representations \( P_j, j = 1 \ldots N \) of the original problem instance \( P \). Note that "isomorphism" indicates pseudo-random perturbation of the original problem instance \( P \). Then, for each obtained solution \( S_{P_j}, j = 1 \ldots |A| \), an analysis program computes the values \( \omega_{\pi_i}^k, k = 1 \ldots L \) for a particular set of solution’s properties \( \pi_i \), \( i = 1 \ldots L \). The reasoning behind performing iterative optimizations of perturbed problem instances is to obtain a valid statistical model on certain properties of solutions generated by a particular algorithm.

Next, the collected statistical data \((\omega_{\pi_i}^k)\) is integrated into a separate histogram \( \chi_{\pi_i}^k \) for each property \( \pi_i \) under the application of a particular algorithm \( A_i \). Since the probability distribution function for \( \chi_{\pi_i}^k \) is in general not known, using non-parametric statistical methods [DeG89], each algorithm \( A_i \) is associated with probability \( p_{A_i} \chi_{\pi_i}^k \) that its solution results in property \( \pi_i \) being equal to \( X \).

Algorithm clustering. In order to associate an algorithm \( A_i \in A \) with the original solution \( S_P \), the set of algorithms is clustered according to the properties of \( S_P \).
The value \( \omega_k^{S_P} \) for each property \( \pi_k \) of \( S_P \) is then compared to the collected histograms \( \chi_i^k, \chi_j^k \) of each pair of considered algorithms \( A_i \) and \( A_j \). Two algorithms \( A_i, A_j \) remain in the same cluster, if the likelihood \( z_{A_i,A_j,\omega_k^{S_P}} \) that their properties are not correlated is greater than some predetermined bound \( \varepsilon \leq 1 \) (\( K \) is the index of the property \( \pi_k \), which induces the highest anti-correspondence between the two algorithms).

\[
z_{A_i,A_j,\omega_k^{S_P}} = \max_{k=1}^{\left| \chi_k \right|} \frac{\text{likelihood}(\pi_k = \omega_k^{S_P})}{[\text{likelihood}(\pi_k = \omega_k^{S_P})] + \text{likelihood}(\pi_k = \omega_{-k}^{S_P})}
\]

It is important to stress that a set of properties associated with algorithm \( A_i \) can be correlated with more than one cluster of algorithms. For instance, this can happen when an algorithm \( A_i \) is a blend of two different heuristics \( A_{ij}, A_{ik} \) and therefore its properties can be statistically similar to the properties of \( A_j, A_k \). Obviously, in such cases exploration of different properties or more expensive and complex structural analysis of programs is the only solution.

**Decision making.** This process is straightforward. If the plaintiff’s algorithm \( A_y \) is clustered jointly with the defendant’s algorithm \( A_y \) and \( A_y \) is not clustered with any other algorithm from \( A_z \), substantial similarity between the two algorithms is positively detected at a degree quantified using the parameter \( z_{A_y, A_y, \omega_k^{S_P}} \). The court may adjourn to the experiment several slightly modified replicas of \( A_y \) as well as a number of strategically different algorithms from \( A_x \) in order to validate that the value of \( z_{A_y, A_y, \omega_k^{S_P}} \) points to the correct conclusion.

Obviously, the selection of properties plays an important role in the entire system. Two obvious candidates are the actual quality of solution and the run-time of the optimization program. Needless to say, such properties may be a decisive factor only in specific cases when copyright infringement has not occurred. Only detailed analysis of solution structures can give useful forensic insights. In the remainder of this manuscript, we demonstrate how such analysis can be performed for graph coloring and boolean satisfiability.

5 **Forensic Engineering: Statistics Collection**

5.1 **Graph Coloring**

We present the developed forensic engineering methodology using the problem of graph \( K \)-colorability. In order to position the proposed approach, initially, we formalize the optimization problem and then survey a number of existing widely accepted heuristics. Finally, we propose a set of heuristic properties that can be used to correlate individual graph coloring solutions to their algorithms.

Since many resource assignment problems can be modeled using graph coloring, its applications in VLSI CAD are numerous (logic minimization, register assignment, cache line coloring, circuit testing, operations scheduling [Cou79]). The problem can be formally described using the following standard format:

**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]. Particular instances of the problem that can be solved in polynomial time are listed in [Gar79]. For instance, graphs with maximum vertex degree less than four, and bipartite graphs can be colored in polynomial time.

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 our research to a few of them. 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 3. 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 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.
successive colorings by random moves. The most common search techniques are simulated annealing [Mor86, Joh91, Mor94] and tabu search [We88, Fl96]. In our experiments, we will 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 [Cu99]).

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.

$\pi_2$: 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.

$\pi_3$: Number of edges that can switch color classes. This criteria analyzes the quality of the coloring. Good 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.

$\pi_4$: 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.

$\pi_5$: Sum of degrees of nodes included in the largest (smallest) color classes. This property aims at identifying algorithms that perform peephole optimizations, since they are not likely to create color classes with high-degree vertices.

$\pi_6$: Sum of degrees of nodes adjacent to the vertices included in the largest (smallest) color classes. The analysis goal of this property is similar to $\pi_4$ with the exception that it focuses on selecting algorithms that perform neighborhood lookahead techniques [Kir95a].

$\pi_7$: 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 [DeG99], the function of standard deviation for each histogram can be used to determine the standard error incorporated in the reached conclusion.

Although instances with small cardinalities cannot be a target of forensic methods, we use a graph instance in Figure 4 to illustrate how two different graph coloring algorithms tend to have solutions characterized with different properties. The applied algorithms are DSATUR and RLF (described earlier in the section). 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 $x^{DSATUR}_{V} = \{1, 2, 3, 0, 1\}$, where histogram value $x_y$ denotes $x$ sets of color classes with cardinality $y$. Similarly, the solution created using RLF results in $x^{RLF}_{V} = \{2, 0, 3, 1\}$. 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.

![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.](image)

5.2 Boolean Satisfiability

We illustrate the key ideas of watermarking-based intellectual property protection 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 $C$?

For instance, $V = \{v_1, v_2\}$ and $C = \{\{v_1, v_2\}, \{v_1\}, \{v_1', v_2'\}\}$ is an instance of SAT for which the answer is positive. A satisfying truth assignment is $t(v_1) = T$ and $t(v_2) = F$. On the other hand, if we have collection $C' = \{\{v_1', v_2'\}, \{v_1\}\}$ there is no satisfying solution.

Boolean satisfiability is an NP-complete problem [Gar79]. It has been proven that every other problem in NP can be polynomially reduced to satisfiability [Coo71, Kar72]. SAT has an exceptionally wide application range. Many problems in CAD are often modeled as SAT instances. For example, SAT techniques have been used in testing [Si87, St96, Ca93, Ko93], logic synthesis, and physical design [De99]. There are at least three broad classes of solution strategies for the SAT problem. The first class of techniques is based on probabilistic search [Gu99, Si99, St95, Da96], the second are approximation techniques based on rounding the solution to a nonlinear program relaxation [Go95], and the third is a great variety of BDD-based techniques [Br95]. 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 $\text{DIFF}$ between the number of clauses currently unsatisfied
that would become 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 [SeI92, SeI93, SeI93a]. 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 [SeI93a].

- **NTAB** performs a local search to determine weights for the clauses (intuitively giving higher weights correspond 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 [Cra96].

- **Rei_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 \( V_{K,t} \). In the remaining set of properties only functionally significant subset of variables \( V_0 = V - V_{K,t} \) 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 \( \pi_4 \) 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 \( n_i \) 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 \( \text{DIFF} = a - b \) 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.

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. Standard deviation impacts the decision making process according to the Central Limit Theorem [DeG89].

### 6 Forensic Engineering: 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) in a single cluster. This procedure is presented 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 \( A_i \in A \) with the original solution \( S_P \), the set of algorithms is clustered according to the properties of \( S_P \). The value \( \omega_{\pi,k}^c \) for each property \( \pi_k \) of \( S_P \) is then compared to the collected histograms \((y_{i,k}, \chi_{i,k})\) of each pair of considered algorithms \( A_i \) and \( A_j \). Two algorithms \( A_i, A_j \) remain in the same cluster, if the likelihood \( z_{A_i, A_j, \omega_{\pi,k}^c} = \max_{k=1}^{\text{likelihood}(\pi_k, \omega_{\pi,k}^c)} \left( \frac{\text{likelihood}(\pi_k, \omega_{\pi,k}^c)}{\text{likelihood}(\pi_k, \omega_{\pi,k}^c)} \right) \)

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 \( C_k \) then a new cluster \( C_{k+1} \) 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 \( C_k' \) of algorithms within \( C_k \), then \( C_k \) is partitioned into two new clusters \( C_k'' \cup A_i \) and \( C_k' \setminus C_k'' \). Finally, algorithm \( A_i \) is removed from the list of unprocessed algorithms \( A \). These steps are iteratively repeated until all algorithms are processed.

![Figure 5: Pseudo-code for the algorithm clustering procedure.](image-url)
Once the algorithms are clustered, the decision making process is straightforward.

- **If** plaintiff’s algorithm $A_p$ is clustered jointly with the defendant’s algorithm $A_d$,
- and $A_d$ 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 $\frac{\gamma \log(\frac{1}{\delta})}{\Delta}$.

The court may adjourn to the experiment several slightly modified replicas of $A_p$ as well as a number of strategically different algorithms from $A$ in order to validate that the value of $\frac{\gamma \log(\frac{1}{\delta})}{\Delta}$ points to the correct conclusion.

\[ A1 \rightarrow A2 \rightarrow A1 \rightarrow A3 \rightarrow A1 \rightarrow A2 \rightarrow A3 \]

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 $A_2$ is recognized as similar to both algorithms $A_1$ and $A_2$, which were found to be dissimilar.

7 Experimental Results (Figure 7)

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 7. 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 [Joh91]) and register allocation graphs. The last five subfigures in Figure 7 depict the histograms of property value distribution for the following pairs of algorithms and properties: DSATUR with backtracking vs. maxis and $\pi_1$, DSATUR with backtracking vs. tabu search and $\pi_1$, iterative greedy vs. maxis 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 $\pi_i = x$ (X-dimension), a test instance can be associated to algorithm $A_1$ with likelihood equal to the ratio of the Y-dimensions of the histogram for $\frac{A_1[x]}{A_2[x]}$. 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 90% of solution instances with their corresponding algorithms with probability greater than 0.95. According to the Central Limit Theorem [DeG89], 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, Tso93]. Parts of the collected statistics are presented in the first ten subfigures in Figure 7. 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 propositional 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 (zoomed comparison presented in the fourth subfigure), where among 100 solution instances no overlap in the value of property $\pi_1$ was detected for Rel_SAT, and WalkSAT.

8 Conclusion

With the emergence of the Internet, intellectual property has become accessible and easily transferable. The improvements in product delivery and maintenance have a negative side-effect: copyright infringement has become one of the most commonly feared obstacles to IP e-commerce. We have proposed a forensic engineering technique that addresses the generic copyright infringement scenario. 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.

9 References

Figure 7: Experimental results: 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 (for two different sets of instances: total: first four subfigures), (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 \( \pi_1 \), (12) DSATUR with backtracking vs. tabu search and \( \pi_1 \), (13,14) iterative greedy vs. maxis and \( \pi_1 \) and \( \pi_4 \), and (15) maxis vs. tabu and \( \pi_1 \).