Computational Forensic Techniques for Intellectual Property Protection

Jennifer L. Wong, Darko Kirovski, and Miodrag Potkonjak

***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: 1) feature and statistics data collection; 2) feature extraction; 3) entity clustering; and 4) 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.**

***Index Terms*—Boolean functions, design automation, intellectual property protection.**

I. Introduction

The rapid expansion of the Internet, and e-commerce, in particular, 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: 1) impact on system performance; 2) robustness of watermark detection with respect to design modularity; and 3) 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: 1) feature and statistics data collection; 2) feature extraction; 3) clustering of heuristic properties for each analyzed tool; and 4) 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 very large scale integrated computer-aided design: graph coloring and Boolean satisfiability.

Manuscript received October 5, 2002; revised May 9, 2003. This paper was recommended by Associate Editor R. Camposano.

J. L. Wong and M. Potkonjak are with the Computer Science Department, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: jwong@cs.ucla.edu).

D. Kirovski is with Microsoft Research, Redmond, WA 98052 USA (e-mail: darkok@microsoft.com).

Digital Object Identifier 10.1109/TCAD.2004.828122

II. 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 [1].

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 [2]. 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 [3] 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, forensic techniques have been explored for detection of authentic Java byte codes [4] and to perform identity or partial copy detection for digital libraries [5]. 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 [6], partitioning [7], [8], and behavioral specification [9] level. In the software domain, a 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 the specification of the design and forensic engineering establishes proof of 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.

III. Forensic Engineering: 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. 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: 1) similar algorithms

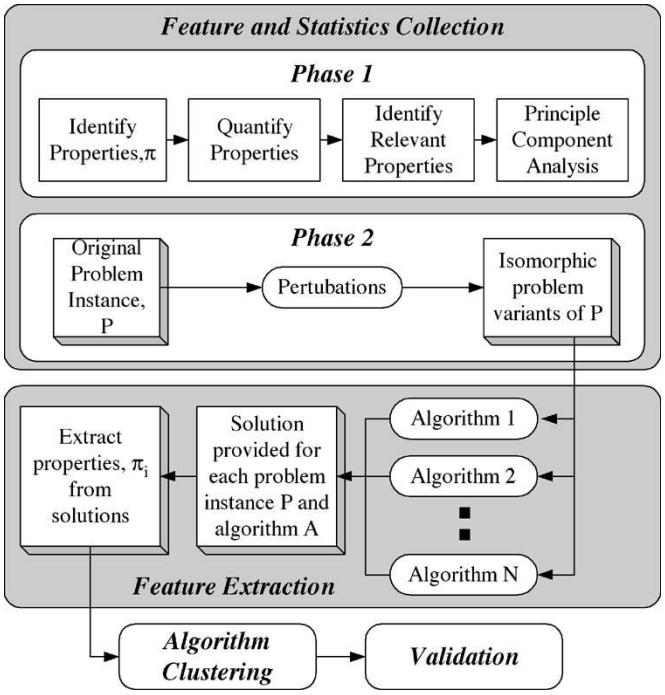

Fig. 1. Flow of the generic forensic engineering approach.

can have different executables and 2) 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 Fig. 1, consists of four fully modular phases: 1) feature and statistics collection; 2) feature extraction; 3) algorithm clustering; and 4) validation.

The first phase, feature and statistics collection, 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 IV-C. For example, the graph coloring RLF algorithm [11], which colors the graph by selecting large independent sets of nodes, 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 each property by abstracting them to their numerical values. The goal is to eventually position the solutions for each algorithm into n-dimensional space. The dimensions are quantified properties which characterize solutions created by all considered algorithms.

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 probability distribution functions (pdfs) under it.

In the fourth step, we conduct principle component analysis [12]. 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 to the format of 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.

In the feature extraction phase, we begin by running all the perturbed instances through each of the algorithms, and gather all the solutions. We apply fast algorithms for extraction of the selected properties from each of the solutions.

In the third phase, algorithm clustering, we begin by placing the relevant properties 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 with little or no error. If the selected properties/features which specifically 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, or unique, for each and every algorithm.

The final step, validation, is the application of nonparametric resubstitution software [13] 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 V.

IV. Forensic Engineering: Feature and Statistics Collection

In this section, we first introduce the graph coloring and Boolean-satisfiability problems. For each of the problems, we illustrate the goals of the feature and statistics collection phase. Additionally, at the end of the section, we introduce a generic notion for solutions properties, and illustrate how these generic notions can be used to formulate solution properties for other optimization problems, such as the scheduling and graph-partitioning problems.

A. Graph Coloring

In this section, we demonstrate the developed feature and statistics collection phase of the forensic engineering methodology using the graph K-colorability problem. We first define the problem and then introduce properties for identifying solutions of 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, $v \in E$?

Graph coloring is an NP-complete problem [14]. 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) [15], backtrack DSATUR [16], MAXIS (RLF-based) [17], and Tabu search [18]. Detailed descriptions of the algorithms can be found in [19] and [20].

A successful forensic technique should be able to distinguish, given a colored graph, whether a particular algorithm has been used to obtain the coloring 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 independent set (IS) cardinalities is used to filter greedy algorithms that focus on coloring graphs constructively (e.g., RLF-based 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 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 candidates to be colored with a single color, in particular when many constituent nodes have low degrees. We use $k\%$ of the largest independent 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 as a quantitative measure.

[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 Ginni coefficient of the histogram is used as well as the average value to quantify this property to a single number.

[π_5] **Sum of degrees of nodes included in the smallest color classes**. The analysis goal of this property is similar to π_4 with the exception that it focuses on identifying algorithms that perform neighborhood look ahead techniques [21]. 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 the histogram values is relatively small. Using standard statistical approaches [22], 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 the graph instance in Fig. 2 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 $\chi_{\pi_1}^{\mathrm{DSATUR}} = \{1_2, 2_3, 0_4\}$, where histogram value x_y denotes x sets of color classes with cardinality y. Similarly, the solution created using MAXIS results in $\chi_{\pi_1}^{\mathrm{MAXIS}} = \{2_2, 0_3, 1_4\}$. 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.

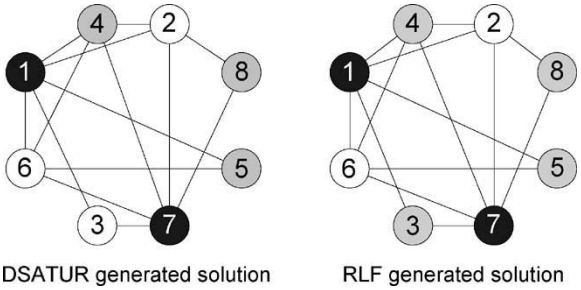

Fig. 2. Example of two different graph coloring solutions obtained by two algorithms DSATUR and MAXIS.

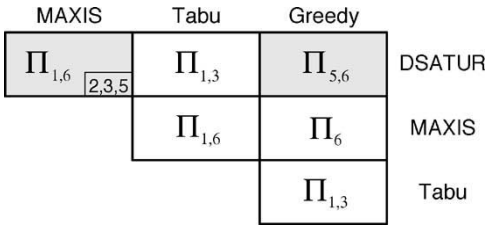

Fig. 3. Property distinction between algorithms.

We can distinguish each of the algorithms from each other using properties $\pi_1 - \pi_6$. We display the distinguishing properties in Fig. 3. 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.

B. 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 [14].

PROBLEM: BOOLEAN 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?

SAT was the first problem to be defined as NP-complete problem [14]. It has been proven that every other problem in NP can be polynomially reduced to the satisfiability problem [14]. SAT techniques have been used in testing [23], [24], logic synthesis, and physical design [25]. There are at least three broad classes of solution strategies for the SAT problem. The first class of techniques are based on probabilistic search [26], [27], the second are approximation techniques based on rounding the solution to a nonlinear program relaxation [28], and the third is a great variety of BDD-based techniques [29]. For the sake of brevity, we demonstrate our forensic engineering technology on the following SAT algorithms: WalkSAT [30], NTAB [31], and Rel_Sat_Rand [32] (more detailed descriptions can be found in [19]).

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 nonimportant variables**. A variable v_i is *nonimportant* 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 versus 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 appearances of a variable**. While property π_3 analyzes the solution from the 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 $\mathbf{DIFF} = \mathbf{a} - \mathbf{b}$ is computed, where $\mathbf{a}$ is the number of clauses currently unsatisfied that would become satisfied if the truth value of v were reversed, and $\mathbf{b}$ 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.

We present a brief analysis of the SAT algorithms and property π_1 in order to illustrate how the SAT 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 probable that the solution is dependent on variables which would have been nonimportant, 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 are determined to be difficult to satisfy. 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 assigned either True or False. Therefore, it is expected that the NTAB algorithm will have a large number of nonimportant variables.

C. Generic Property Development

Identification and the selection of solution properties is essential for the effectiveness of any forensic engineering technique. While each problem may require one or more unique properties, many of the 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 section 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 the edges indicate the 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 aim 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.

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 the assignment of the variables. As in the case with the perturbation properties, one can select single entities, pairs, and subsets as the 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.

[P_3] **Tendency to follow natural algorithm constructs**. Natural algorithm constructs, such as greedy or maximally constrained minimally constraining heuristic (MC/MC), are often used 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, the 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.

Variables 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 fanin and fanout of the nodes in a single cycle. If the transitive fanin and fanout 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.

V. Property Selection and Algorithm Clustering

In this section, we present our approach to algorithm classification. The first step is to analyze the property data and identify relevant properties of the solutions. Once the properties are selected, we conduct algorithm clustering. The algorithm clustering task has two objectives: 1) to identify algorithms that are structurally similar, in the sense that their solutions have the same crucial characteristics indicating that they are using essentially identical mechanisms for optimization and 2) 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 nondecreasing 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 labeled 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 pseudocode in Fig. 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

```
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
```

Fig. 4. Pseudocode for the algorithm clustering procedure.

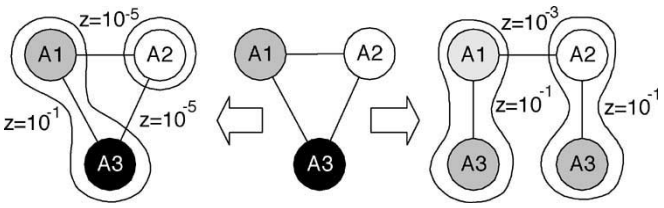

Fig. 5. Two different examples of clustering three distinct algorithms. The first clustering (figure on the left) recognizes substantial similarity between algorithms $\boldsymbol{A}_1$ and $\boldsymbol{A}_3$ and substantial dissimilarity of $\boldsymbol{A}_2$ with respect to $\boldsymbol{A}_1$ and $\boldsymbol{A}_3$. Accordingly, in the second clustering (figure on the right) the algorithm $\boldsymbol{A}_3$ is recognized as similar to both algorithms $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$, which were found to be dissimilar.

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 correlated is smaller than a predetermined bound $\epsilon \ll 1$

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

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 smaller 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.

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 Fig. 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 both 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.

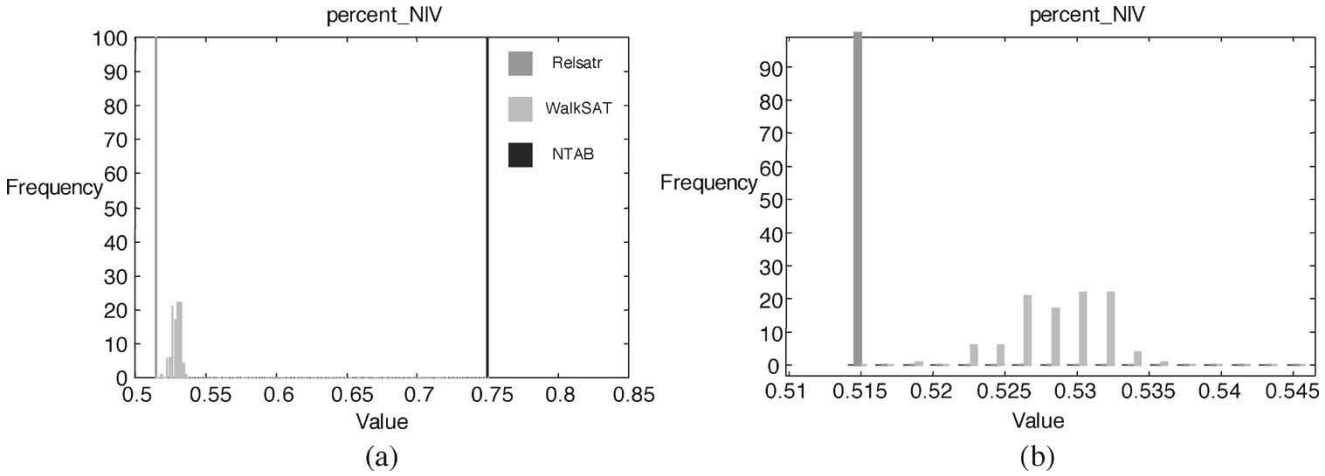

Fig. 6. Property $\boldsymbol{\pi_1}$ for SAT applied to NTAB, Rel_SAT, and WalkSAT.

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 $(S(A \cap B))/(S(A \cup B))$. 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 algorithms A_1, $i = 1, \ldots, n$ can be obtain using the following formula: $(S(A_1))/(\sum S(A_i))$.

VI. 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 algorithms used for creating solutions of SAT instances, has been tested using real-life and abstract benchmark sets of instances adopted from [33] and [34]. Parts of the collected statistics are presented in Fig. 6.

Fig. 6 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 nonimportant variables property has multimodal distribution for the algorithms. Fig. 6(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.

For example, in the Fig. 6(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, 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 Fig. 7 depict the histograms of property value distribution for the following pairs of algorithms and properties: DSATUR with backtracking versus maxis for π_3, DSATUR with backtracking versus tabu search for π_7, iterative greedy versus maxis for π_1 and π_4, and maxis versus 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 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 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 or 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 $1–10^{-6}$.

Even better classification can be obtained using statistical method discussed in Section V, from which we obtained Tables I and II. 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.

VII. 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

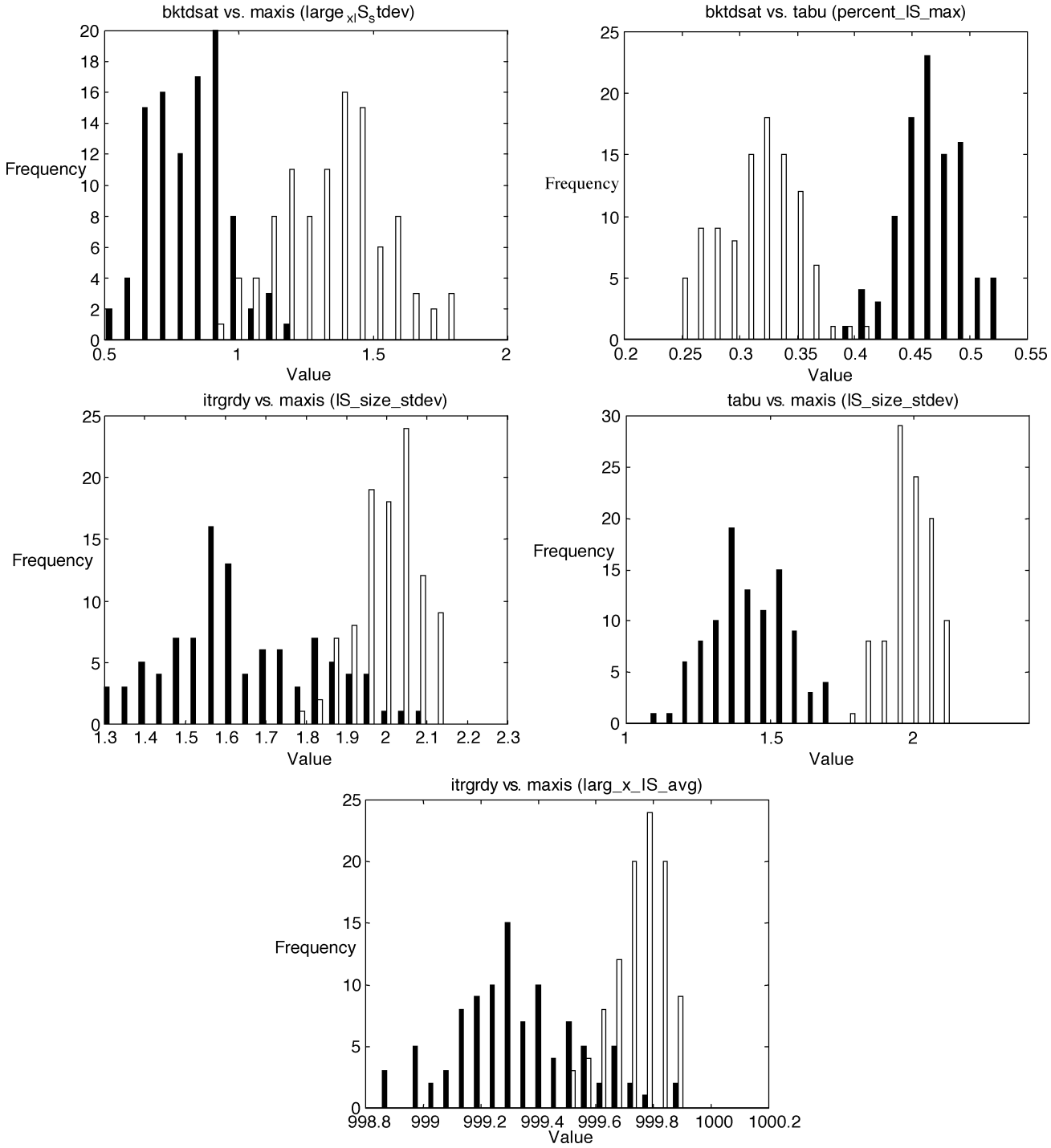

Fig. 7. Histograms of property value distribution for algorithm $\boldsymbol{a}$ (black) versus algorithm $\boldsymbol{b}$ (white) and properties: DSATUR with backtracking versus maxis for $\boldsymbol{\pi}_3$, DSATUR with backtracking versus tabu search for $\boldsymbol{\pi}_7$, iterative greedy versus maxis for $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_4$, and maxis versus tabu for $\boldsymbol{\pi}_1$.

TABLE I
EXPERIMENTAL RESULTS: GRAPH COLORING

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 II
EXPERIMENTAL RESULTS: BOOLEAN SATISFIABILITY

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

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] N. Rudin, K. Inman, G. Stolvitzky, and I. Rigoutsos, *DNA Based Identification*: Kluwer, 1998.

[2] D. McGahn, "Copyright infringement of protected computer software: An analytical method to determine substantial similarity," *Rutgers Comput. Technol. Law J.*, vol. 21, no. 1, pp. 88–142, 1995.

[3] D. Grover, "Forensic copyright protection," *Comput. Law Secur. Rep.*, vol. 14, no. 2, pp. 121–122, 1998.

[4] B. Baker and U. Manber, "Deducing similarities in Java sources from bytecodes," in *Proc. USENIX Tech. Conf.*, 1998, pp. 179–190.

[5] S. Brin, J. Davis, and H. Garcia-Molina, "Copy detection mechanisms for digital documents," in *Proc. ACM SIGMOD Annu. Conf.*, 1995, pp. 398–409.

[6] E. Charbon and I. Torunoglu, "Watermarking layout topologies," in *Proc. Asia South Pacific Design Automation Conf.*, 1989, pp. 213–216.

[7] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking techniques for intellectual property protection," in *Proc. IEEE/ACM Design Automation Conf.*, 1998, pp. 776–781.

[8] G. Wolfe, J. L. Wong, and M. Potkonjak, "Watermarking graph partitioning solutions," *IEEE Trans. Computer-Aided Design*, vol. 21, pp. 486–489, Oct. 2002.

[9] G. Qu and M. Potkonjak, "Analysis of watermarking techniques for graph coloring problem," in *Proc. ICCAD*, 1998, pp. 190–193.

[10] C. Collberg and C. Thomborson, "Software watermarking: Models and dynamic embeddings," in *Proc. Symp. Principles Program. Lang.*, 1999, pp. 311–324.

[11] F. Leighton, "A graph coloring algorithm for large scheduling algorithms," *J. Res. Nat. Bureau Stds.*, vol. 84, pp. 489–506, 1979.

[12] I. Jolliffe, *Principal Component Analysis*. Berlin, Germany: Springer-Verlag, 1986.

[13] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*. London, U.K.: Chapman & Hall, 1993.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: W. H. Freeman, 1979.

[15] D. Welsh and M. Powell, "An upper bound for the chromatic number of a graph and its applications to timetabling problems," *Comput. J.*, vol. 10, no. 85–86, 1967.

[16] D. Brelaz, "New methods to color the vertices of a graph," *Commun. ACM*, vol. 22, no. 4, pp. 251–256, 1979.

[17] B. Bollobas, *Random Graphs*. Norwell, MA: Academic, 1985.

[18] A. Hertz and D. de Werra, "Using tabu search techniques for graph coloring," *J. Comput.*, vol. 39, no. 4, pp. 345–351, 1987.

[19] J. L. Wong, D. Kirovski, and M. Potkonjak, "Computational forensic techniques for intellectual property protection," Univ. of California, Los Angeles, Tech. Rep. TR030 048, 2003.

[20] Joseph culberson's graph coloring resources page Available: http://web.cs.ualberta.ca/~joe/Coloring/index.html [Online]

[21] D. Kirovski, Y.-Y. Hwang, M. Potkonjak, and J. Cong, "Intellectual property protection by watermarking combinational logic synthesis solutions," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1998, pp. 194–198.

[22] M. DeGroot, *Probability and Statistics*. Reading, MA: Addison-Wesley, 1989.

[23] P. Stephan, R. K. Brayton, and A. S. Vincentelli, "Combinational test generation using satisfiability," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 1167–1176, Sept. 1996.

[24] H. Konuk and T. Larrabee, "Explorations of sequential ATPG using Boolean satisfiability," in *Proc. IEEE VLSI Test Symp.*, 1993, pp. 85–90.

[25] S. Devadas, "Optimal layout via Boolean satisfiability," in *Proc. Int. Conf. Computer-Aided Design*, 1989, pp. 294–297.

[26] J. Marques-Silva and K. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, pp. 506–521, May 1999.

[27] B. Selman, "Stochastic search and phase transitions: Al meets physics," in *IJCAI*, vol. 1, 1995, pp. 998–1002.

[28] M. Goemans and D. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *J. ACM*, vol. 42, no. 6, pp. 1115–1145, 1995.

[29] R. Bryant, "Binary decision diagrams and beyond: Enabling technologies for formal verification," in *Proc. Int. Conf. Computer-Aided Design*, 1995, pp. 236–243.

[30] B. Selman, H. Kautz, and B. Cohen, "Local search strategies for satisfiability testing," in *Proc. Cliques, Coloring, Satisfiability: 2nd DIMACS Implementation Challenge*, 1993, pp. 521–532.

[31] J. Crawford, "Solving satisfiability problems using a combination of systematic and local search," in *Proc. 2nd DIMACS Challenge*, 1993, pp. 1–7.

[32] R. Bayardo and R. Schrag, "Using CSP look-back techniques to solve exceptionally hard SAT instances," in *Principles and Practice of Constraint Programming*. New York: Springer, 1996, pp. 46–60.

[33] A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende, "An interior point approach to Boolean vector function synthesis," in *Proc. Midwest Symp. Circuits Syst.*, 1993, pp. 185–189.

[34] Y. Tsuji and A. V. Gelder, "Incomplete thoughts about incomplete satisfiability procedures," in *Proc. 2nd DIMACS Challenge*, 1993, pp. 559–586.