

# Relative Generic Computational Forensic Techniques

Jennifer L. Wong and Miodrag Potkonjak

University of California, Los Angeles

**Abstract.** Computational forensic engineering is the process of identification of the tool or algorithm that was used to produce a particular output or solution by examining the structural properties of the output. We introduce a new Relative Generic Forensic Engineering (RGFE) technique that has several advantages over the previously proposed approaches. The new RGFE technique not only performs more accurate identification of the tool used but also provides the identification with a level of confidence. Additionally, we introduce a generic formulation (integer linear programming formulation) which enables rapid application of the RGFE approach to a variety of problems that can be formulated as 0-1 integer linear programs.

The key innovations of the RGFE technique include the development of a simulated annealing-based (SA) CART classification technique and a generic property formulation technique that facilitates property reuse. We introduce instance properties which enable an enhanced classification of problem instances leading to a higher accuracy of algorithm identification. Finally, the single most important innovation, property calibration, interprets the value for a given algorithm for a given property relative to the values for other algorithms. We demonstrated the effectiveness of the RGFE technique on the boolean satisfiability (SAT) and graph coloring (GC) problems.

## 1 Introduction

Software and hardware piracy resulted in a loss of over \$59 billion globally between 1995 and 2000, and continues to induce an average of \$12 billion each year in the United States alone. Intellectual Property Protection techniques (IPP), such as watermarking and fingerprinting, have been proposed as solutions. These techniques have shown significant potential. However, they introduce additional overhead on each application of the tool and IP, and they cannot be applied to tools which already exist. Computational forensic techniques removes these limitations. At the point of suspected algorithm or tool infringement, forensic engineering can be applied to show that the tool was used to produce the suspected output. The overhead of the technique is only applied once, off-line. Additionally, the technique can be applied to any existing design or software tool. The underlying approach is to examine the structural properties of the output of different tools for a specific problem and use statistical analysis to identify the tool which was used to create a particular output (infringed).

Not only does forensic engineering have application for IPP, but it is important to note that it has a number of other applications, which in many situations have even higher economical impact. For instance, forensic engineering can be used for optimization algorithm development, as a basis for developing or improving other IPP techniques, for the development of more powerful benchmarking tools, for enabling security, and facilitating runtime prediction. More specifically, computational forensic engineering can be used to perform optimization algorithm tuning, algorithm development, and analysis of algorithm scaling.

RGFE can be applied to an arbitrary problem which can be formulated as a 0-1 linear programming problem. In this generic formulation, properties of the problem are extracted and used to analyze the structure of both instances of the problem and the output or solutions of a representative set of tools. Using the information gathered, the RGFE technique builds and verifies a Classification and Regression Tree (CART) model to represent the classification of the observed tools. Once built, the CART model can be used to identify the tool used to generate a particular instance output. This RGFE approach consists of three phases: Property Collection, Modeling, and Validation. The key enabling factors in the property collection phase are the ability to extract properties of a given problem systematically and to conduct calibration of these properties to reflect the differences between solutions generated by the tools. We briefly outline the key novelties of the RGFE technique.

- **Generic Forensic Engineering.** We introduce a generic flow for the RGFE technique that allows it to be applied to a variety of optimization problems with minimal retargeting.
- **Generic Property Formulation.** A systematic way to develop instance and solution properties for different problems allows the generic RGFE technique to be applied to a variety of optimization problems. The generic property formulation is applied to a problem which has been formulated in terms of an objective function and constraints. Special emphasis is placed on the widely used 0-1 ILP formulation.
- **Instance Properties.** Problem instances have varying complexity which is often dependent upon particular structural aspects of the instance. Additionally, different algorithms perform differently depending on the complexity or structure of the problem instance it is presented with. We introduce instance properties which provide a measure for comparing instances and therefore facilitate more accurate analysis and classification of the algorithms.
- **Calibration.** Calibration is performed on both instance and solution properties in order to place the data into the proper perspective. For instance properties, calibration provides a way to scale and classify the instances, while the solution properties for each algorithm are calibrated per instance to place the data into the proper perspective to differentiate the algorithms.
- **One-out-of-any Algorithms.** The technique must be able to classify algorithms not only in terms of the algorithms which have been previously analyzed but also as an unknown algorithm.
- **CART model.** We have developed a new CART model for classification. The key novelty is that the new CART model does not only partition the

solution space so that classification can be conducted but also maximizes the volume of space that indicates solutions that are created by none of the observed algorithms. The CART model is created using a SA algorithm.

## 2 Preliminaries

In this section, we briefly summarize research in the areas which are most directly related: intellectual property protection, forensic analysis, and statistical methods. Additionally, we discuss the relationship between the computation forensic engineering technique and the proposed RGFE approach. Finally, we briefly discuss the generic formulation, the boolean satisfiability problem, algorithms, and the generic formulation for the SAT problem.

Due to the rapidly increasing reuse of intellectual property (IP) such as IC cores and software libraries, intellectual property protection (IPP) has become a mandatory step in the modern design process. Recently, a variety of IPP techniques, such as watermarking, obfuscation, and reverse engineering, have attracted a great deal of attention [1].

We use non-parametric statistical techniques for classification because they can be applied to data which has arbitrary distributions and without any assumptions on the densities of the data [2]. The Classification and Regression Trees (CART) model is a tree-building non-parametric technique widely used for the generation of decision rules for classification. The SA optimization technique originates from statistical mechanics and is often used to generate approximate solutions to very large combinatorial problems [3]. Bootstrapping is a classification validation technique that assesses the statistical accuracy of a model. In the case of nonparametric techniques, bootstrapping is used to provide standard errors and confidence intervals [2].

The Computational Forensic Engineering (CFE) [4] technique identifies an algorithm/tool, which has been used to generate a particular previously unclassified output, from a known set of algorithms/tools. This technique is composed of four phases: feature and statistics collection, feature extraction, algorithm clustering, and validation.

In the feature and statistics collection phase, solution properties of the problem are identified, quantified, analyzed for relevance and selected. Furthermore, preprocessing of the problem instances is done by perturbing the instances - removing any dependencies the algorithms have on the instance format. In the next phase, each of the perturbed instances are processed by each of the algorithms and the solution properties are extracted. The algorithm clustering phase then clusters the solution properties in  $n$ -dimensional space, where  $n$  is the number of properties. The  $n$ -dimensional space is then partitioned into subspaces for each algorithm. The final step validates the accuracy of the partitioned space.

This approach performed well on both the graph coloring and boolean satisfiability problem. However, that was the case only under a number of limiting assumptions. The computational forensic engineering technique performed algorithm classification on one-out-of- $k$  known algorithms, and was tested on a

variety of different instances. However many of these instances had similar instance structures. This forensic engineering technique is problem specific and is not easily generalizable to other problems. Lastly, the CFE technique performed analysis of the techniques in the form of blackboxes.

The RGFE technique eliminates several major limitations of the CFE technique. The technique performs one-out-of-any classification instead on one-out-of- $k$ . In this case, output of an algorithm that was never previously analyzed can be classified as unknown. The key enabler for the effectiveness of the Relative Generic Forensic technique is the calibration of problem instances. By identifying, analyzing, and classifying instances by their properties, the quality of the RGFE classification is expanded to another dimension enabling more statistically sound classification. Additionally, we present a generic formulation and generic property formulation that enables the application of this technique to numerous optimization problems.

The boolean satisfiability (SAT) problem has a variety of applications in many areas such as artificial intelligence, VLSI design and CAD, operations research, and combinatorial optimization [5]. Probably the most well-known applications of SAT in CAD are Automatic Test Pattern Generation [6]. Other applications include logic verification, timing analysis, delay fault testing, FPGA routing, and combinational equivalence checking [5]. Formally, the boolean satisfiability problem can be defined in the following way.

**Problem:** *Boolean Satisfiability*

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

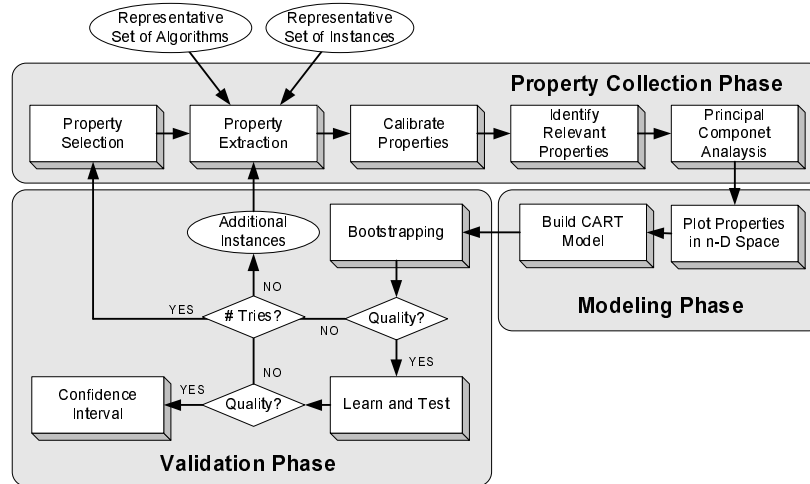
**Question:** *Is there a satisfying truth assignment for  $C$ ?*

Many different techniques have been developed for solving the boolean satisfiability problem. Techniques such as backtrack search [7], local search [8], continuous formulation and recursive learning [9] are among the most popular. Additionally, several public domain software packages are available such as GRASP [9], GSAT [10] and Sato [7].

### 3 Forensic Engineering Flow

In this section, we introduce the RGFE technique. The technique operates on a problem instance in the generic formulation, ILP. Our implementation is restricted to instances that are formulated as 0-1 ILP. The technique consists of two stages: analysis and evaluation. In the analysis stage, the goal is to classify the behavior of algorithms for a specific problem specified using the 0-1 ILP format with a high confidence. The flow of the analysis stage is presented graphically in Figure 1 and using a pseudo code format in Figure 2.

The analysis stage of the RGFE technique, shown in Figure 1, is composed of three phases: property collection, model building, and validation phases. The property collection phase defines, extracts, and calibrates instance and solution properties for the given optimization problem. In the modeling phase, the rele-



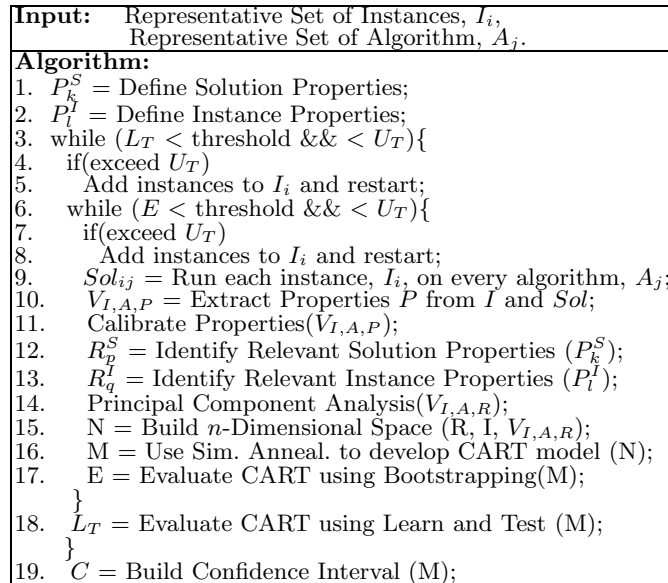
**Fig. 1.** Overall flow of the RGFE technique: Analysis Stage.

vant properties are used to develop a CART classification scheme. The last phase tests and validates the quality of the CART model.

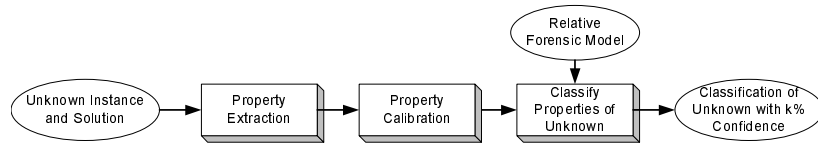
In this phase, the main steps are property selection and property calibration. We begin by selecting generic instance properties that will assist in classifying the characteristics of the given problem. The solution properties are selected to characterize the decisions or the optimization mechanisms used by an algorithm on a particular type of instance. Once a set of properties have been selected for the targeted optimization problem, we proceed to extract each of the properties from our representative sets of instances and algorithms.

In the final steps, relevant properties are identified. Relevant properties are properties which aid in distinguishing the algorithms from each other. If a property yields the same value for all (or a great percentage of) instances or all algorithms, then it is not useful in the classification process, and is excluded from further consideration. Lastly, principal component analysis is performed in order to eliminate the set of properties which provide similar information as other sets of properties. All of the steps in the property collection phase are encapsulated in Figure 2, lines 1, 2, and 9-14.

In the modeling phase, the calibrated properties are used to model the behavior of each of the algorithms. This is accomplished by representing each solution from each of the available algorithms as a point in  $n$ -dimensional space. Each dimension represents either a solution or an instance property. The number of dimensions,  $n$ , is the total number of properties. Once the space is populated with the extracted data, we apply a generalized CART approach (see Section 6). The validation stage is an iterative process which uses the statistical techniques, bootstrapping and learn-and-test, to validate and improve the quality of the CART model.



**Fig. 2.** Pseudo-code for the RGFE technique.



**Fig. 3.** Flow of the Evaluation Stage.

The overall RGFE goal is to be able to correctly classify the output of an unknown algorithm. Once the Analysis stage has completed, this goal of classification is done in the evaluation stage. The evaluation process, shown in Figure 3, begins with property extraction of both instance properties and solution properties from the unknown instance and algorithm output. The properties that are extracted are the set of properties that we used to build the final CART model in the analysis stage. Next, the properties are properly calibrated according to the selected calibration scheme for each property. The calibrated properties of the unknown instance and solution are then evaluated by the CART model. The algorithm that the CART model classifies the output into is the algorithm which produced the solution with the confidence level of the algorithm in the model, which was found at the end of the analysis stage.

Note that the analysis stage of the approach must only be performed once for a set of observed algorithms. However, once the analysis stage is done, the evaluation stage can be applied repeatedly. Only when new algorithms are observed

must the analysis process be repeated. In order to correctly classify the new observed algorithm(s), the properties must be recalibrated to take into account the new algorithm(s). In some cases, it may be necessary to define new properties and to process additional instances on each of the observed algorithms in order to achieve a high confidence level.

## 4 Calibration

Calibration is the mapping of raw data values to values which contain the maximum amount of information to facilitate a particular task, which in this case is algorithm classification. We introduce calibration using an example. Consider two different SAT instances, par8-2-c and par8-1-c, solved using the GRASP and Walksat SAT algorithms. We evaluate the instances and solutions with the solution property of non-important variables. Non-important variables are variables that may switch their assignment in such a way that the correctness of the obtained solution is preserved. For par8-2-c the property values were 0.529 for GRASP and 0.706 for WalkSAT, and for par8-1-c, 0.391 and 0.594 respectively.

Without calibration, by considering only these two instances, we would associate a range of 0.39 to 0.59 to the GRASP solutions and of 0.53 to 0.71 to the Walksat solutions. These two ranges overlap and therefore classification is difficult. The reason is obvious; the two instances have different structure. There is intrinsically many more non-important variable in the instance par8-2-c than in par8-1-c. Calibration can compensate for this difference in instances. For example, we see that in both cases, GRASP has a property value approximately 20% lower than that of Walksat for this property. Calibration of the values with respect to the other algorithms enables proper capturing of the relationships between the algorithms, which is not visible from the raw values.

We have developed two calibration schemes. The first calibration approach is a rank-ordered scheme. For each property value on a particular instance, we rank each of the algorithms. Using these rankings, a collaborative ranking for the property is built by examining the rankings of each of the algorithms on all instances. Additional consideration must be made on how to resolve any ties in ranking, and how to combine rankings for individual instances. One can use either average ranking, median ranking, or some other function of ranking on the individual instances. In our experimentations we used modal ranking - where the ranking of each algorithm is defined as the rank that was detected on the largest number of instances.

Rank-order calibration schemes are simple to implement and are robust against data outliers. However, rank order schemes eliminate the information about the relationship between numerical values for a given property of the algorithms. Additionally, the property after rank order calibration does not provide a mechanism for stating an unpopulated region. Unpopulated regions are necessary for the RGFE technique to classify output from an algorithm which has not be observed or studied in the model. Rank-based property calibration can only classify unobserved areas when multiple properties are consider together.

The second type of calibration mechanism is a scale-based scheme. In these types of techniques, calibration is done by mapping the data values from the initial scale on to a new scale. Possible types of data-mapping are normalization against the highest or lowest value, against the median, or against the average value. We use a scheme where the smallest value on all instances is mapped to value 0, the largest value to value 1, and all other values are mapped according to the formula:  $x_{new} = \frac{x_{init} - x_s}{x_l - x_s}$ , where  $x_{init}$  is the initial value for the property,  $x_s$  is the smallest value, and  $x_l$  is the largest value prior to calibration.

The advantage of a scale-based scheme is, in principle, higher resolution and more expressive power than a rank order scheme. However, these types of approaches can be very sensitive to data outliers - a few exceptionally large or small values. For a scale-based scheme, each of the property values may be plotted on a segment after data-interpretation on the absolute values has been applied. Regions of the segment which are populated by a particular algorithm are defined as classification regions for these algorithms. Regions of the segment which are not populated by any algorithm are specified as unclassified.

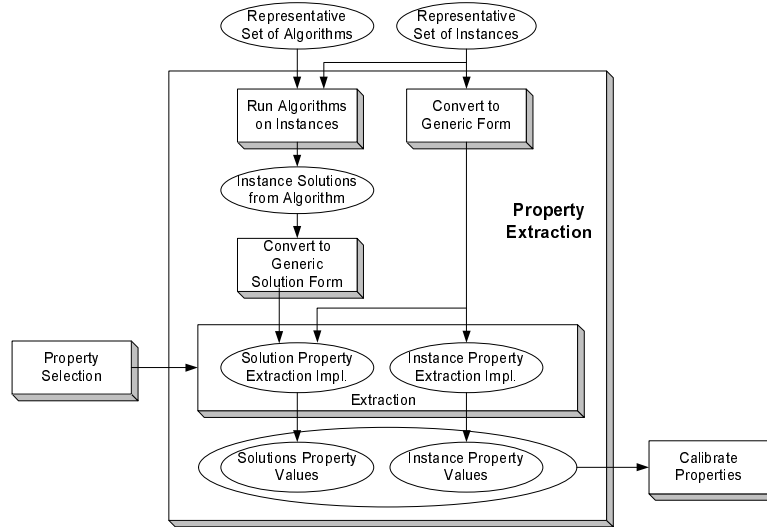
## 5 Properties

There are two key benefits for developing properties in the generic form (0-1 ILP). The first is a conceptual benefit, while the second is software reuse. The conceptual benefit is that treating properties in a generic form greatly facilitates the process of identifying new properties for newly targeted optimization problems. The software reuse benefit lies in the fact that many properties that are developed for one optimization problem can be easily reused for forensic analysis of other optimization problems. Therefore, software for the extraction of these properties need only be written once.

Note that although many problems can be specified using the generic format, often a specific optimization problems have specific features. For example, depending on the problem, the generic formulation may or may not have an objective function. Specifically, the SAT problem contains an empty objective function; as long as all the constraints are satisfied, the value of the objective function is irrelevant. Other properties to consider include the types of variables that appear in the constraints (positive only, negative only, or both), the weights of the variables in the constraints (are they all the same or not), does the objective function contain all of the variables in the problem or only a subset, and so forth. The key is to identify the essential properties of the problem and develop a quantitative way to measure them.

Note that the solution properties can also be extracted in the generic form by mapping the solution output of the algorithms to the generic solution form, then computing the property values. We illustrate the steps in Figure 4.

The representative set of instances for the problem are used both in their standard representation and in generic form. On the right side of the Property Extraction phase, the instances are converted into generic form and then in this form the instance property extraction methods are applied. The instance



**Fig. 4.** Procedure for generic property extraction.

property values are collected and passed on to the calibration step. On the left side of Figure 4, the representative set of algorithms executes each of the instances in the representative set. The solutions for each algorithm on each instance are converted to the generic solution formulation, which is then given to the solution property extraction method along with the original instance in its generic formulation. The solution property values for the given algorithm and instance are collected and the data passes to the calibration phase.

We have developed a number of generic instance properties which we illustrate with their representative meaning for the SAT problem.

[I<sub>1</sub>] **Constraint Difficulty.** Each constraint in the problem formulation contains coefficients for each variable appearing in the constraint and the value (b-value) on the right-hand side of the constraint. The goal of constraint difficulty is to provide a measure of how much effort and attention the algorithm places on a given constraint. For example, in the SAT formulation, each constraint represents a single clause, and therefore all variables have unit weight. The b-value of the constraint is dependent on the number of positive and negative literals in the constraint. Therefore, in this case this generic property summarizes information about the size of the clauses in the instance. The aggregate information about constraints can be expressed using statistical measures such as average and variance, which we actually used in our system.

[I<sub>2</sub>] **Ratio of Signs of Variables.** The key observation is that some variables tend to appear in all constraints in a single form, while others variables will appear in multiple forms and have more balanced appearance counts. For this property, we assume, without loss of generality, that all coefficients b are positive. In the problem formulation, analysis of the positive, negative, and  $x$ -weighted, occurrences

of a variable can be examined with respect to the total number of occurrences of the variable in the instance. In the SAT problem, we can use this property to identify the tendency of a variable in the instance to be assigned true or false. Again, various statistical measures can be used to aggregate this information. We use average and variance.

- [I<sub>3</sub>] **Ratio of Variables vs. Constraints.** This property can be applied to all or a subset of variables in all or a subset of the constraints in the instance. It provides insight into the difficulty of these constraints. A low number of variables in a large number of constraints can imply that the constraints are difficult to satisfy due to the fact that numerous constraints are dependent on the same variables.
- [I<sub>4</sub>] **Bias of a Variable.** We measure the bias of a variable to be assigned to either zero or one, based on the number of constraints which would benefit from the variable being assigned each way.
- [I<sub>5</sub>] **Probability of satisfying constraints.** This property considers the difficulty of satisfying each constraint based on the variables, weights of the variables, and its b-value. We define the probability of the constraint to be satisfied as shown below.

$$P(\text{constraint satisfied}) = 1 - [\bigcup_{v_i} P(\text{variable assigned opposite of constraints benefit})].$$

Solution properties analyze the relationship of the solution and structure of the problem instance. We have developed the following properties.

- [S<sub>1</sub>] **Non-important variables.** This property identifies variables that received an assignment which has no effect on the objective function or on the satisfiability of the constraints. Therefore, the goal is identify variables which are not crucial to the solution of the problem. Constructive and greedy algorithms tend to find solutions which have a high number of variables which are not crucial to the solution.
- [S<sub>2</sub>] **Variable(s) Flip with k% of Constraints still satisfied.** While the non-important variables property aims at identifying variables that have no bearing on the solution of the instance, this property attempts to measure the importance of the variables which impact the objective function and/or the constraints.
- [S<sub>3</sub>] **Constraint Satisfaction.** This property aims at identifying the extent to which the constraint was satisfied. For example, for constraints of the type  $Ax \geq b$ , we can define the property as the value of  $\frac{Ax(sol)-b}{Ax(max)-b}$ . This equation evaluates how much more the constraint was satisfied over the required level by considering the solution's  $Ax$  value less the required value against the maximum possible value less the requirement. This formulation is applied to each constraint in the SAT problem. In the case of SAT, this property translates to the level to which each clause is satisfied by the solution.
- [S<sub>4</sub>] **Variable Tendency.** In many cases, constructive algorithms follow the natural tendencies presented by the instance. More specifically, variables that have all positive coefficients have an intrinsic inclination to be assigned a value of 1, and vice versa. This property tries to quantify to what extent this tendency was followed by a particular type of algorithm. A measure of variable tendency is the number of variables which were assigned according to the ratio of its positive and negative appearances of a variable in the SAT problem.

## 6 Model Building and Validation

In this section, we discuss the modeling phase of the RGFE technique. We begin with a discussion concerning how clustering and classification of the property data for the tools is achieved. The Classification and Regression Trees (CART) model is adopted and generalized for classification, and we analyze the benefits of the model and its application. Finally, we present details on the use of SA to generate the CART model. The starting point for the development of the classification model is that each solution for each algorithm for a given problem is represented as a point in  $n$ -dimensional space, where  $n$  is the number of solution and instance properties. The goal of classification is to partition the space into subspaces that classify regions populated mainly, or in the ideal case only, by solutions produced by a particular algorithm. We define  $A + 1$  classification classes, where  $A$  is the number of algorithms observed. The additional classification class, a unique addition to the standard classification problem, is reserved for subspaces that are not populated by solutions of any of the observed algorithms. If a given output falls into these spaces, the output is classified as produced by an unknown or unobserved tool. Our goal is to perform classification of the data with a model of low Kolmogorov complexity, yet high accuracy [2]. The low Kolmogorov complexity indicates that we did not overtune our model to the given data. Specifically, we follow the principle that for every parameter in the model, we have to have at least five points in the property space.

We adopt the CART model as a starting point for classification for a number of reasons. The model is intuitive, extensively proven to provide accurate classification in many situations, and provides an easy mechanism to enforce the principle of low complexity. The essence of the CART model can be conveniently summarized in the following way. From the geometric point of view, the CART model can be interpreted as partitioning of the  $n$ -dimensional space using  $(n - 1)$ -dimensional hyperplanes that are orthogonal on the axis of the space.

In order to develop the CART model of the  $n$ -dimensional data as required by the RGFE technique, we first define a standard grid. The resolution of the grid is determined by a user specified threshold for misclassification. We keep altering the resolution using binary search until the threshold is not achieved. Next, we examine each of the hypercubes, which the grid defines, for misclassifications, i.e. we identify regions which contain data from multiple algorithms.

Once the  $n$ -dimensional space has been sufficiently divided into hypercubes, the adjacent hypercubes are merged to represent regions of classification for each algorithm. For this task, we use a SA approach. The approach begins with a random classification of the space, which implies random merging of hypercubes into classification regions. Each merged classification solution is evaluated according to the following objective function:  $OF = \alpha N_i + \beta P_m$ , where  $N_i$  is the number of parameters needed to represent the defined classification in the CART model and  $P_m$  is the percentage of misclassification which can occur. The intuition behind the objective function is that the CART model will have smaller representation and complexity if fewer parameters are used. However,

smaller representation implies higher misclassification levels. As a result, we try to balance both components in the objective function.

The SA algorithm requires specification of two additional key components: a basic move generation mechanism and a cooling schedule. The move generation mechanism specifies which changes can be made to the current solution in order to generate a new, so-called neighborhood, solution. In the case of the hypercubes, we define a change as either the movement of a single hyperplane of the hypercube by a single grid unit, or as the replacement of a single hypercube side with a new hypercube side in a different dimension.

The cooling schedule consists of a number of parameters which define the temperature decrement of the SA algorithm. These parameters are initial temperature, stop criterion, temperature decrement between successive stages, and the number of transitions for each temperature value. We define the initial temperature as the temperature on which 50% of the moves result in an increased OF. We stop searching if we visit  $s$  ( $s$  was 3 in our experimentations) temperatures without any improvement in solution. We decrement the temperature using a geometric schedule and at each temperature value, we consider 1000 generations. Validation was conducted using standard learn-and-test and resubstitution validation techniques.

## 7 Experimental Results

In this section, we present the results of the experimental evaluation of the RGFE technique on the boolean satisfiability and the graph coloring problem. The standard approach to solving the register assignment problem is to reduce the problem to the graph coloring problem (GC). We present experimental results for the register assignment problem in the form of graph coloring. For each problem, we discuss the instance and solution properties used for the developed generalized CART model, and the optimization algorithms for which were used to build the model. Furthermore, we provide experimental evidence of the importance of simultaneously considering both solution and instance properties using the SAT problem. Finally, the performance of the overall technique for classifying both observed and unobserved algorithm output for both problems is presented.

The final CART model used by the RGFE technique for the SAT problem used only five generic properties: two instance properties, and three solution properties. The properties which were selected are the following:  $[I_1]$  Weighted Average of “Short” Clauses,  $[I_3]$  Ratio of Variables vs. Number of Constraints,  $[S_1]$  Percentage of Non-important variables,  $[S_2]$  Average Variable Flip with 80% of Constraints still satisfied, and  $[S_3]$  Average Constraint Satisfaction.

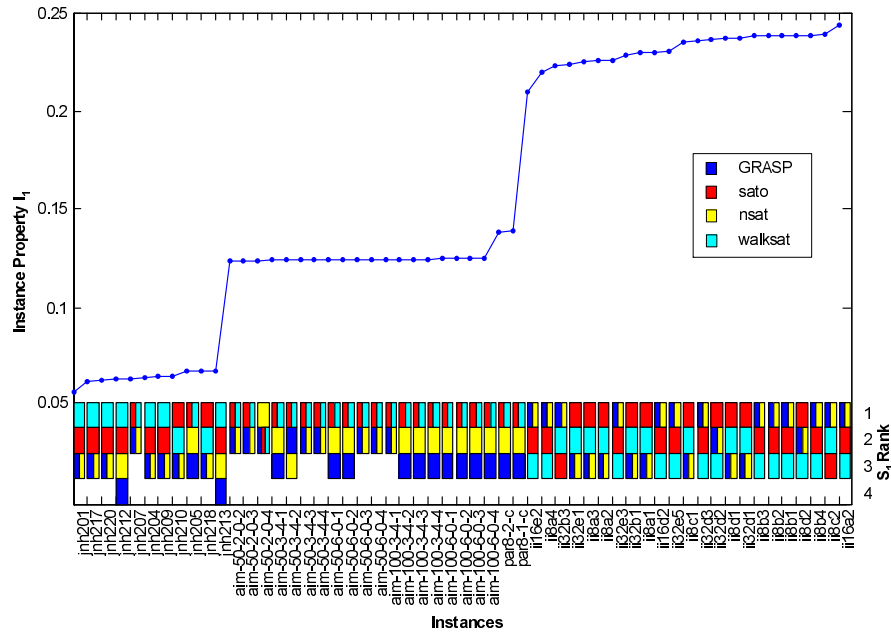
These solution properties were used to analyze the outputs of four SAT algorithms: GRASP, nsat, sato and walksat. The GRASP algorithm [9], is a generic search algorithm which performs non-chronological backtracking with a conflict analysis procedure and tree pruning. The nsat algorithm is a simple backtrack search algorithm. Sato applies the David-Putnam approach and tries to speed up unit propagation [7]. Walksat is a stochastic search algorithm, which is incom-

plete, developed by Kautz and Selman. It is an iterative improvement approach which performs two separate types of moves: a flip of the current variable assignment of a variable appearing in an unsatisfied clause, or a greedy move. The SAT algorithm that we used for testing the classification ability of unobserved algorithms is the Satz algorithm. This approach is a systematic, complete SAT solver with randomized restarts.

The representative set of SAT instances used to build the CART model include instances from the DIMACS standard benchmark set. The first set,  $juh^*$ , are randomly generated instances of the constant density model. The problem distribution of these instances is Random P-SAT. The  $aim^*$  instances are random 3-SAT instances. The third set of instances are propositional versions of parity learning problems,  $par8^*$ . Lastly, the  $ii^*$  set are instances of inductive interference problems. Additional information on these benchmark sets and others can be found at [11].

In order to demonstrate the importance of comparing properties of instances and solutions, we present data and comparisons using the SAT problem. We used a set of 55 SAT instances. In Figure 5, we present the dependency between instance property,  $[I_1]$  weighted average of “short” clauses and solution property  $[S_1]$  non-important variables. The solution property is calibrated using rank-order. A rank of one is given to the algorithm with the highest property value on a given instance, and a rank of 4 is the lowest rank. If two algorithms have the same property value on an instance, they are given the same rank number. The horizontal axis of the figure displays the name of each of the instances. The vertical axis indicates the value of instance property. The piece-wise linear line displays the values of property  $I_1$  for each instance. Note that there are three distinguishable subsets of instances which have distinctly different  $I_1$  values, approximately 0.06, 0.12, and 0.23. For each instance, the calibrated rank of the corresponding solution property values for each algorithm is displayed along the horizontal axis in bar format. For each of the distinguishable regions of  $I_1$  values, a noticeable pattern in the algorithms ranks can be seen. For example, in the first region of  $I_1$  values, 0.06, the GRASP algorithm is always ranked the lowest, the walksat algorithm is ranked highest in nine of the eleven instances, and sato is only ranked second and first in all cases. Similar ranking consistency can be seen in the results for solution property,  $S_2$  in Figure 6. According to this property, for the instances in the highest instance property range, approximately 0.23, walksat is ranked the highest in all but two of the cases, sato is ranked second in all but three cases, and nsat and GRASP are tied in all cases. In the other two regions, patterns also exist.

In order to test the generalized CART model, we ran both new instances and new instances where the order of the inputs (variables) was subject to random permutation. All the instances were from the same class of selected benchmarks, and included instances which were not in the representative set. In total, we used the CART model to classify 1000 different solutions of each algorithm, including the unobserved algorithm *satz*. The results of the classification are shown in Table 7. The rows of the table present the solver used to generate the solution,

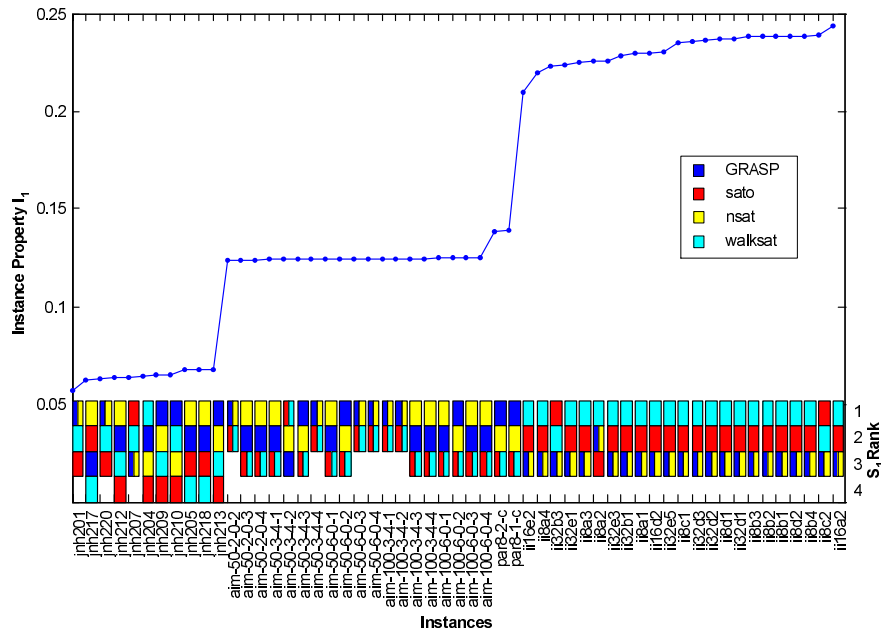


**Fig. 5.** SAT: Dependency between instance property ( $I_1$ ) and solution property ( $S_1$ ).

and the columns represent the algorithm for which the output was classified as by the CART model. The CART model had confidence interval of 92%.

We now present the properties and algorithms used for application of the RGFE technique on the graph coloring problem. The final CART model for classification of algorithms for the GC problem used the following instance and solution properties: $[I_2]$  Variable Appearance vs. Average Variable Appearance,  $[I_3]$ Ratio of Variables vs. Number of Constraints on Edge Constraints,  $[S_1]$  Percentage of Non-important variables - nodes which can change coloring without increasing the number of colors used,  $[S_2]$  Average Variable Flip with 80% of Constraints Still Satisfied - average number of coloring possibilities per node, and  $[S_3]$  Constraint Satisfaction - percentage of nodes which can only be colored with a single color.

For the GC problem, we used four algorithms for building the CART model: dsatur, maxis, itrgrdy, and tabu. Additionally, we use the bkdsat algorithm as the unobserved algorithm. The dsatur algorithm, developed by Brelaz [12], selects the node to color at each step based on the degree of the node, and the number of colors which cannot be used to color the node due to conflicts with previously colored nodes. Nodes with the least number of coloring possibilities are colored first. Maxis is a recursive, large first algorithm (RLF) which applies exponential backtracking. The iterative improvement approach, itrgrdy, uses an iterative local improvement search to improve the current coloring assignment.



**Fig. 6.** SAT: Dependency between instance property ( $I_1$ ) and solution property ( $S_2$ ).

Tabu search is a probabilistic iterative improvement algorithm [13]. The original assignment of colors is applied randomly, and conflicts are attempted to be resolved by reassigning conflicting nodes to different colors. Lastly, the bkdsat algorithm is a greedy algorithm which attempts to color higher density regions of the graph.

For instances of the graph coloring problem, we use graphs from the DIMACS benchmark set, types of which include register allocation, leighton, scheduling, and quasi-random graphs. We select 100 instances to build our CART model. The resulting CART model had a confidence interval of 94%. The results for attempting classification on 50 additional instances and their permutations are

SAT						GC					
Solver	GRASP	nsat	sato	walksat	unobs	Solver	dsatur	maxis	itrgrdy	tabu	unobs
GRASP	993	5	0	0	2	dsatur	992	4	0	1	3
nsat	5	994	0	0	1	maxis	2	996	0	0	2
sato	0	1	995	3	1	itrgrdy	0	2	997	0	1
walksat	2	0	4	992	2	tabu	2	0	1	994	3
satz	3	4	2	1	990	bkdsat	1	3	0	0	996

**Table 1.** Experimental results for SAT and GC.

shown in Table 7. The table displays the classification of the solution generated by each of the algorithms on 1000 permuted instances from the DIMACS set. The model is capable of classifying the output with extreme accuracy.

## 8 Conclusion

We have introduced the RGFE technique for identifying which tool, if any, produced a particular solution to a given optimization problem. The new approach is capable of not only differentiation between outputs of known algorithms, but is able to determine if an unknown algorithm was used to produce the output. The new technique enables rapid retargeting to new optimization problems.

## References

- [1] Qu, G., Potkonjak, M.: Intellectual Property Protection in VLSI Design: Theory and Practice. Kluwer (2003)
- [2] Breiman, L., et al.: Classification and Regression Trees. Chapman and Hall (1993)
- [3] Kirkpatrick, S., et al.: Optimization by simulated annealing. *Science* **220** (1983) 671–680
- [4] Wong, J., Kirovski, D., Potkonjak, M.: Non-parametrical statistical computational forensic techniques for intellectual property protection. In: 4th International Information Hiding Workshop. (2001) 71–86
- [5] Marques-Silva, J., Sakallah, K.: Boolean satisfiability in electronic design automation. In: Cliques, Coloring, and Satisfiability. (2000)
- [6] Hamzaoglu, I., Patel, J.: New techniques for deterministic test pattern generation. In: IEEE VLSI Test Symposium. (1998)
- [7] Zhang, H.: Sato: An efficient propositional prover. In: Conference on Automated Deduction and LNAI 1249. (1997) 272–275
- [8] Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: AAAI. (1992) 440–446
- [9] Marques-Silva, J., Sakallah, K.: Grasp: a search algorithm for propositional satisfiability. *Transactions on Computers* **48** (1999) 506–521
- [10] Selman, B., Kautz, H.: Domain-independent extensions to gsat: Solving large structured satisfiability problems. In: International Conference on Artificial Intelligence. (1993) 290–295
- [11] SATLIB: The satisfiability library. <http://www.satlib.org> (2004)
- [12] Breaz, D.: New methods to color the vertices of a graph. *Comm. of the ACM* **22** (1979) 251–256
- [13] Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. *Journal of Computing* **39** (1987) 345–351