Fingerprinting Intellectual Property Using Constraint-Addition

Gang Qu and Miodrag Potkonjak
Computer Science Department, University of California, Los Angeles, CA 90095

Abstract

Recently, intellectual property protection (IPP) techniques attracted a great deal of attention from semiconductor, system integration and software companies. A number of watermarking-based techniques have been proposed for IPP. One of the key limitations of watermarking is that it does not facilitate tracing of illegally resold intellectual property (IP). Fingerprinting resolves this problem by providing each customer with a unique instance of functionally identical IP. We propose a general technique which enables fingerprinting at all level of design process and is applicable to an arbitrary optimization step. In particular, we address the following fingerprinting problem: How to generate a large number of high quality solution for a given optimization problem by solving the initial problem only once. In addition we also discuss how to select a subset of k solutions from the pool of n solutions so that the solutions are maximally different.

In order to make our discussion concrete we focus on a single NP-complete problem - graph coloring. We test the new fingerprinting on a number of standard benchmarks. Interestingly, while on random graphs it is relatively difficult to produce a large number of solutions without nontrivial quality degradation, on all real-life compilation graphs we are able to generate millions of solution which are all optimal.

1 Introduction

We introduce a new methodology of fingerprinting for the purpose of IP protection. This method is different from the current fingerprinting techniques because of i) its almost zero run-time overhead, and ii) the controllable number of distinct fingerprinted copies it can produce. We have successfully implemented this idea in the case of graph (vertex) coloring (GC) problem.

1.1 Motivation

Today's engineering teams are facing more severe challenges than ever: the shortage of engineering manpower, the soaring design complexity, the growing time-to-market pressure, and the fast rising fabrication cost just to name a few. According to a study of 320 engineering teams in North America by Collett International [15], by the year 2000, the new-design productivity must be doubled and reuse productivity must be improved by a factor of 12. At the same time, design cycle time must drop by 15 percent, team size grows by 36 percent, and reuse increase by 53 percent.

Multi-vendor IP integration is by far the most promising solution to these challenges. From the IP providers' standpoint, the protection of IP remains as one of the most vital concerns[15].

This research was supported in part by NSF under grant CCB-9734166.

DAC 2000, Los Angeles, California

Moreover, IP owners have to protect themselves as well as their legal users. The ownership needs to be protected to recover the high R&D cost. It can be achieved by a newly developed *constraint-based watermarking* technique [9], which embeds the IP provider's signature as additional design constraints during the design and synthesis process to create a rather unique IP.

It is also crucial to distribute IPs with the same functionality but different appearance to different users, because the problem of determining legality of the ownership will become insurmountable if all users get exact same IP and one of them illegally redistributes the IP. However, it is too expensive to build a unique IP for every legal user by applying the same watermarking technique on users' signatures.

Previously, there are techniques to put fingerprints into the IPs. One is to partition the problem into a set of subproblems, and introduce constraints to connect these small problems if necessary, then solve each subproblem independently. This method has very poor performance unless the original problem has specific structure[10].

Another choice [5] is to solve the problem once, then generate a relatively small problem based on this solution. Resolving the small problem will give us possibly new solutions. Cost for solving small instance is usually much lower than is for the original, but when the request for different solutions are huge, this overhead cannot be ignored. Moreover, different solutions are not guaranteed.

1.2 New approach and contributions

We propose a fingerprinting technique to overcome this difficulty illustrated by the GC problem. Figure 1 shows the generic approach of the new methodology. It consists of two phases, first we develop methods for generating as many GC solutions with the smallest overhead, then we provide scheme to distribute these solutions among potential users.

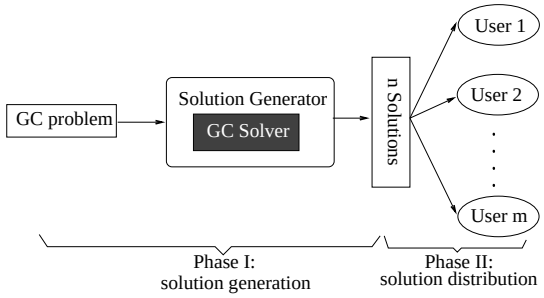


Figure 1: Fingerprinting technique for IP protection: generating n solutions and distributing among m users.

This new approach provides five main benefits:

1. Since we call the solver only once, the run-time overhead for generating many solutions over that for one single solution is almost zero.
2. In three of the four techniques that we have implemented, the number of solutions can be controlled and the solutions are guaranteed distinct.
3. The IP provider's signature can be embedded in the fingerprinting process without additional watermarking techniques.
4. Both symmetric and asymmetric fingerprints can be created by this method.

5. With proper distribution schemes, the techniques are collusion-secure.

In the next section, we review the related efforts in watermarking and fingerprinting. Then we unveil the principles of the new fingerprinting methodology. We use the GC problem to explain the two phases of the technique: generating various number of solution at one time and distributing solutions among users. We present the experimental data and conclude in section 7.

2 Related Work

Both watermarks and fingerprints have been used to discourage piracy of digital objects (text, image, audio, video, multimedia), but almost all of such techniques fail to protect IP (hardware, software, algorithm, architecture, and etc.). We survey the recent efforts on watermarking and fingerprinting techniques for IPP.

Watermarking for the purpose of IPP is a challenge because of the requirement for preserving the IP's correct functionality. The *constraint-based watermarking* method[9] translates the signature into a set of additional constraints during the design and implementation of IP in order to uniquely encode the signature into the IP. The effectiveness of this technique lies in the large solution space of the problem which interprets the IP. Then the signature will be added as extra constraints that eventually cut the solution space. Ownership is proved by the usually exceptionally small probability of obtaining a specific solution from the initial solution space without the signature. This new methodology is mathematically sound and has been applied at the level of algorithm, behavior, logic synthesis and physical design, as well as in FPGA[2, 4, 10, 11, 13].

Fingerprints are the characteristic of an object that is completely unique and incontrovertible. Fingerprint-like marks have also been developed to distinguish other objects from maps and mathematical tables to diamonds and explosives. However, only recently with the proliferation of digital data has it drawn academic and industrial attention. Boneh and Shaw [2] propose the most efficient symmetric fingerprinting schemes in the sense that both the distributor and the user know the fingerprinted copy. Pfitzmann and Schunter[12] introduce asymmetric fingerprints, where only the user knows the fingerprinted data while the distributor can identify the user's information from the data. Biehl and Meyer[1] combine these two and give a construction more suitable for broadcast data.

Like the watermarking techniques for artifact, the fingerprinting techniques developed by far introduce minute errors into each copy of the data. These errors, called *marks*, are not acceptable in the design process of IP, and even if they maintain the correct functionality, a machine can easily detect these changes. One may argue that for each user, we can take his/her signature and apply the constraint-based watermarking techniques proposed by Kahng et al[9]. This method is effective for proof of authorship. However, the computation cost which is linear to the number of users makes their techniques impractical for the purpose of fingerprinting. Moreover, due to the randomness of most of the heuristic algorithms for hard problems, we cannot guarantee that each user will receive a unique copy.

In this paper, we propose techniques that are capable of creating various numbers of different copies, which can be controlled by changing the parameters of the techniques, and all the copies are generated with almost the same expense as for a single copy.

3 Fingerprinting for IP protection

3.1 Background

Fingerprints are characteristics of an object which are sufficient enough to distinguish it from other similar objects. Fingerprinting refers to the process of adding fingerprints to an object and recording them, or process of identifying and recording fingerprints that are already intrinsic to the object[14]. The core idea of fingerprinting is to give each user a copy of the object containing a unique fingerprint, which can be used to identify that user.

One of the most accepted model for fingerprinting[2, 1, 7, 14, 12] can be described as: In the original object, a set of marks is selected probabilistically, where a *mark* is one bit of information that has two slightly different versions. The distributor can choose one of the two versions of each mark to embed either a 0 or a 1 when the object is sold to a user, and thus construct a binary word which becomes the fingerprint of this user. Two general assumptions on the object to be fingerprinted are:

Error-tolerance assumption: the object should remain useful after introducing small errors or marks, and the user cannot detect the marks from the data redundancy. The more errors that the object can tolerate, the more places we can put these marks.

Marking assumption: two or more users may detect a few marks that differ in their copies, but they cannot change the undetected marks without rendering the object useless.

According to a taxonomy given by Wagner[14], the *statistical fingerprinting* is characterized as: given sufficiently many misused objects to examine, the distributor can gain any desired degree of confidence that he has correctly identified the compromised. The identification is, however, never certain. This is one of the fundamentals for many fingerprinting schemes[2, 1].

3.2 Context of fingerprinting for IP protection

Our goal is to protect IP through fingerprinting. The major difference between IPs and the objects mentioned in the previous section is that IPs are usually error-sensitive, which violates the error-tolerance assumption. Therefore, we cannot directly apply the existing fingerprinting techniques for IP protection. For an IP to be protected, it must satisfy the following requirements:

- The IP should be well-interpreted as a problem which has a large solution space. The sole role for the error-tolerance assumption is to guarantee a relatively large valid object space. Introducing errors is one way to create such space, but not the only way.
- The cost to derive the solution space should be negligible comparing to that of inventing the IP. It takes tremendous human and computer resources to design and implement a piece of IP, and we cannot afford to produce different copies by simply repeating the entire design process.
- The existence of algorithms and/or state-of-the-art software which solves the problem. In our experience, these exist for many problems in the field of VLSI CAD.

Before describing the general approach of our constraint-addition based fingerprinting, we list the objectives for any effective fingerprinting techniques:

- The protocol, with the help of the existing problem solver, should be able to create $K >> 1$ solutions at the expense close to that for finding one single solution.
- The protocol, combined with the problem solver, should be capable of generating solutions that are far away from each other, otherwise collusion-secure solution distribution schemes are not possible.
- The fingerprinting protocol should be non-intrusive to the problem solver so that it can be transparently integrated with existing design flows via pre-processing and/or post-processing.

- The fingerprints must provide high credibility to identify each and every user.
- The overhead or degradation of the quality of solution after embedding fingerprints should be minimized.
- The user cannot remove or alter the fingerprints without complete knowledge of the IP or rendering the IP useless.
- The IP provider can, without much difficulty, trace the guilty user(s) from the fingerprints in illegal copies.

3.3 A generic approach of fingerprinting for IP protection using addition of constraints

As we discussed earlier, our fingerprinting procedure consists of two phases: the solution generation phase and the solution distribution phase.

The first concern is: does the problem always have a large solution space and what happens if the solution space is very limited? Since each user will receive a unique copy, we have to construct solution space large enough to accommodate all users or we are in troubles to release copies.

Many hard problems have a sufficient number of solutions in nature. For instance, in the GC problem, isolated nodes can be marked by any colors, and two connected nodes that have the same set of neighbors except themselves can exchange their colors. As another example, in the satisfiability (SAT) problem, flipping over the value of a don't-care variable in a satisfying assignment will give a different solution. Given that the solution space is large, to find k solutions is in general at least as hard as solving the original problem. Moreover, once we have the solution space, we have to maintain a one-to-one mapping from the solution to the user who receives this copy.

In our approach for solving the solution generation problem, we are not attempting to find the whole solution space. Instead, we add a set of extra constraints to the initial problem such that we can easily create (many) new solutions from one solution to the modified problem. In fact, we find a subspace of the solution space, where a base of this subspace can be built from this set of extra constraints, and the solution to the modified problem is a seed.

Once we have a set of solutions generated from a given base, where each solution can be uniquely expressed as a combination of the base. We can map each user's signature to a set of coefficients and assign him/her the corresponding copy of solution. Hence we only need to keep the base and the information for each user.

With a released solution, the user may gain some information about the problem. For example, if the user has a graph colored by 69 colors, then he knows the graph is 69-colorable and a satisfying assignment of a SAT problem tells the user that the original SAT is satisfiable. Since the solutions we created now are not random any more, users may collect different copies, detect their difference and produce new copies differ from their originals. The fingerprinting techniques should be designed to prevent this or make it hard, and allow the owner to be able to trace at least one of the dishonest users with a convincible probability from a forged copy.

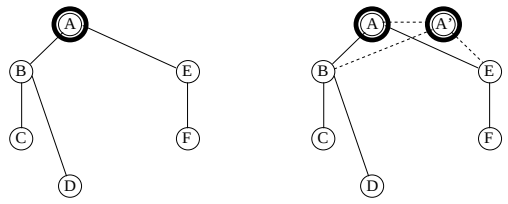
4 Creation of various solutions for GC problem

In this section, we present four techniques to generate solutions for the GC problem: (1) duplicating a selected set of vertices; (2) modifying small cliques; (3) adding edges between unconnected vertices; and (4) post-processing on one solution.

Given a graph $G(V, E)$ and a positive integer $k \leq |V|$, we say that G is k-colorable if we can color the vertices V such that adjacent vertices have different colors. The optimization graph (vertex) coloring problem is to find the minimum number k such that G is k-colorable. This problem is *NP-complete* and many heuristic algorithms have been developed dedicated to this problem (*http://dimacs.rutgers.edu/*).

4.1 Vertex duplication

Given one coloring scheme to a graph, if we know that one vertex can also be colored by another alternative color, then immediately we can have one more solution to the same GC problem. Furthermore, on knowing k vertices each has an second valid color, we are able to create 2^k different solutions with almost no cost. And these k vertices and their associate colors will serve as the base for the solution space we have.



(a) Original graph. (b) New graph.

Figure 2: Duplicating vertex (A) to generate various solutions.

Figures 2 and 3 show this technique and an implementation. The idea is to select a vertex, duplicate it by creating a new vertex and connecting it to all the neighbor's of the selected vertex. Now the selected vertex can be labeled by either its color or the color of its duplication without violating the rules for GC. To guarantee these two vertices receive different colors, we add an edge in between. In Figure 2(b), vertices A and A' will be labeled by two different colors which can both be used to color A in the original graph 2(a).

Input: a graph $G(V, E)$, and an integer n.
Output: a new graph G', such that n different solutions to G can be generated from one solution to G'.

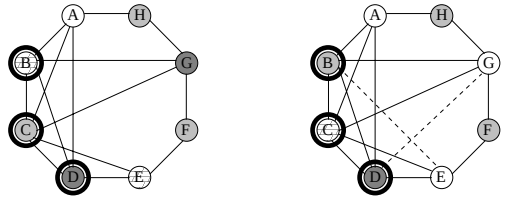
Algorithm:
copy $G(V, E)$ to $G'(V', E')$;
copy V to V'';
repeat ($\lceil \log_2 n \rceil$ times)
{ select a vertex v from V'';
delete v from V'';
add a new vertex v' to V';
add a new edge (v, v') to E';
for all (u such that edge $(u, v) \in E$)
add an edge (u, v') to E';
}
report graph $G'(V', E')$;

Figure 3: Pseudo code for vertex duplication.

4.2 Clique manipulation

In any valid color scheme, vertices from one clique will receive different colors, however, the solution may become invalid if they switch their colors. For example, consider the triangle BCD in Figure 4(a), once the other five vertices' colors are fixed as shown, it is easy to see this is the only solution.

We can add extra constraints to this triangle, as shown in Figure 4(b), and now the three colors for vertices B, C, and D can be assigned arbitrarily. In general, if we choose a clique of size k, and for each vertex, we connect its neighbors to all other vertices in the clique, then based on one solution to the resulting graph, we get



(a) Original graph. (b) New graph.

Figure 4: Manipulating small clique (triangle BCD).

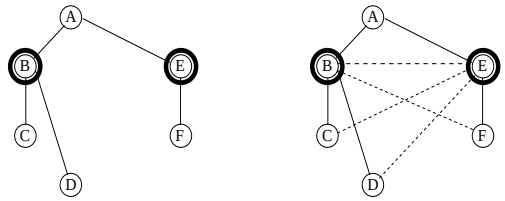
$k!$ solutions to the original GC problem by assigning each of the k different colors to one of the vertices in the clique.

Several cliques can be selected and they combine together forming a base for the solution space.

4.3 Bridge construction

There is no constraint for two vertices that do not have an edge connecting them. In [13], a watermarking technique is proposed where a message is embedded into the graph by adding edges between selected pairs of vertices, and the authorship can be claimed by showing the probability that every pair of vertices receiving different colors, which is not necessarily true in the original graph.

We can exploit the same idea here by selecting a pair of unconnected vertices, connecting one to all the neighbors of the other as well as these two vertices themselves. In Figure 5(b), vertices B and E are selected, and when we color the new graph, B and E will have different colors, say *red* and *green*. Now we can build 4 solutions where B and E are colored as *(red, red)*, *(red, green)*, *(green, red)* or *(green, green)*.



(a) Original graph. (b) New graph.

Figure 5: Constructing bridge (BE) to generate various solutions.

It is worth mentioning here that this method is not restricted to a pair of unconnected vertices. We can select k unconnected vertices (an independent set of size k), create a complete graph over these vertices and connect each node to the neighbors of the others. Obviously, in this way, k^k different solutions can be derived from a single solution.

By constructing bridges, we can make the attacker's job very hard. In Figure 5, if two users detect that vertex B is marked by red and $green$ respectively in their solutions, and provided they know our fingerprinting technique, all the conclusion they may draw is that a bridge has been built between B and a vertex colored by either red or $green$. They have to search through a relatively large space and it will become even worse for them if we are selecting k unconnected vertices.

A hybrid of bridge construction and clique manipulation is practical with additional post-processing. We can choose k vertices (not necessarily unconnected), create a clique of size k and apply the clique manipulation technique. Now since the k selected vertices do not belong to an independent set, an arbitrary combination of their colors may not be valid in the original graph. A trivial procedure has to be conducted before releasing any solution which tests the validity of a given combination.

4.4 Solution post-processing

The last technique we discuss here requires post processing on a given solution.

Suppose we have colored graph $G(V, E)$ by k colors, denote V_i the subset of V that are colored by the i^{th} color. So $V = \cup_{i=1}^{k} V_i$, and $V_i \cap V_j = \phi$ for all $i \neq j$. Now we select l colors and let $V' = \cup_{j=1}^{l} V_{i_j}$. Consider the subgraph $G[V']$ of G that is induced by V', we know this graph is l-colorable. In general, its size is relatively small and we can exhaustively find all the l-color solutions to it. Similarly we may construct another induced subgraph $G[V'']$ such that $V' \cap V'' = \phi$ and recolor it exhaustively. If we find n_1 and n_2 solutions for $G[V']$ and $G[V'']$ respectively, by applying the multiplication principle, we can create $n_1 \times n_2$ solutions to the original graph $G(V, E)$.

4.5 Comparison of the techniques:

One common characteristic for the first three techniques is that they belong to the category of *pre-processing*, where we modify the graph before it is colored by any GC solver (as a "blackbox"). Once a solution to the modified GC instance is returned, many different solutions to the original GC problem can be generated from this "seed" solution easily. The number of solutions can be controlled by tuning the parameters (see Table 1). But if we constrain the original graph too much, we may have some overhead, i.e., using extra colors for the modified graph comparing to that for the initial graph.

In contrast, when we apply "solution post-processing" method, the GC solver will solve exactly the initial GC instance and it will provide us the best solution it can find. And in the post process, we always use the same amount of colors, therefore, there is guaranteed no overhead. However, it is not so easy to create many solutions as we do by the first three techniques, and the number of solutions are not controllable. In our experience, the better is the solver, the less space left for post-processing. For example, in a 85-color solution for a random graph of 1000 nodes, 66 colors are used for maximal independent sets.

To end this section, we summarize these techniques in the following table, for each technique, we list its parameters and the size of the solution space. The base of the solution space can be easily built from the parameters, the overhead will be discussed later by experimental results.

Technique and parameters	Number of solutions from one seed
Vertex duplication: duplicate k vertices	2^k
Clique manipulation: manipulate k cliques of size $n_1, n_2, \ldots, n_k$	$n_1! \times n_2! \times \ldots \times n_k!$
Bridge construction: select k sets of unconnected vertices of size $n_1, \ldots, n_k$	$n_1^{n_1} \times n_2^{n_2} \times \ldots \times n_k^{n_k}$
Solution post-processing: Partition the vertices into k subsets by their colors, in each subset, we find $n_1, n_2, \ldots, n_k$ solutions	$n_1 \times n_2 \times \ldots \times n_k$

Table 1: Summary of the four techniques.

5 Solution distribution scheme

As discussed before, the distributor wishes to give each user a uniquely fingerprinted copy. However, this is impractical for mass produced products like electronic books, software or CD-ROMs. One scheme[2] is to divide the data that a user received into two parts: the *public data* which is common to all users, and the *private*

data which is unique to a particular user. Typically, the private part is small but should be able to provide enough information for the distributor to trace the user.

On the other hand, unlike human fingerprinting, the embedded digital fingerprints may be changed while the object is kept useful or functional correct. Two or more users may easily detect the difference between their copies, and come up with another copy without their fingerprints. In[2], for naive redistribution where a user redistributes his copy of the object without altering it, a *c-secure* code is constructed that can trace at least one of the guilty users from a coalition with size up to c users. For other cases, they construct *c-secure codes with ε-error* which allows an innocent user comes under suspicion with probability ε but requires a code length polynomial to $\log \frac{1}{\varepsilon}$ and $\log n$ (n is the number of potential users).

To avoid computing the problem many times, we create various solutions from one "seed" solution, therefore, similarities can be expected and it may be much easier for pirates to figure out these similarities and forge new valid solutions without their own fingerprints if the solutions are distributed improperly. For example, if we use the vertices duplication method with k vertices, in the seed solution, each of these k vertices will have a primary color and a secondary one. We are able to generate 2^k solutions where the only difference is the colors assigned to these k vertices. Suppose user A receives a copy of all the primary colors, and user B has one with all the secondary colors. Then if users A and B compare their copies, they can discover all the 2^k solutions.

We can discourage this with the aids of carefully designed distribution schemes. Although we cannot force users from redistribution, we can have the copies released in such a way that from a forged copy, we are able to catch at least one user from the coalition. The protocols in [1, 2] are applicable in this case. The basic idea is to select a subset of the solution space generated by the "seed solution" and release only solutions from this subset instead of the entire solution space. This subset should satisfy the following:

- Any combination of solutions cannot create a new solution in this subset, i.e., the innocent user will be protected.
- From any solution created by a combination of solutions from this subset, at lease one of the original solution can be traced. In another word, from an illegal copy, at least one of the guilty users will be caught.

Notice the domino effect of the GC problem (and many other hard optimization problems as well): changing the colors of a few vertices may render the entire solution. This phenomena does not exist in the contexts of fingerprints for classical objects, and our new techniques utilize it to discourage piracy. For example, if we use *clique manipulation* or *bridge construction* techniques, (or a hybrid of these two), it is still possible to find part or all the vertices that have been selected. However, the pirates will have difficult time to find the matching that tells them which clique it belongs to and/or which vertices are connected to it by bridges. And it is unlikely for the users to create new solutions, which are significantly different from the originals, from the copies generated by *solution post-processing*.

6 Experimental Results

We implement the proposed fingerprinting techniques in Section 4 on two types of graphs (available at *http://mat.gsia.cmu.edu/COLOR /instances.html*). The first is standard random graphs with given number of vertices and edges. The other type of graphs is generated from the register allocation problem of variables in real codes. Table 2 shows the parameters for these graphs.

Instance	Vertices	Edges	Optimal Coloring
fpsol2.i.1.col	496	11654	65
fpsol2.i.2.col	451	8691	30
fpsol2.i.3.col	425	8688	30
inithx.i.1.col	864	18707	54
inithx.i.2.col	645	13979	31
inithx.i.3.col	621	13969	31
mulsol.i.1.col	197	3925	49
mulsol.i.2.col	188	3885	31
mulsol.i.3.col	184	3916	31
mulsol.i.4.col	185	3946	31
mulsol.i.5.col	186	3973	31
zeroin.i.1.col	211	4100	49
zeroin.i.2.col	211	3541	30
zeroin.i.3.col	206	3540	30
DSJC1000.5.col.b	1000	249826	N/A

Table 2: Characteristics of benchmark graphs.

6.1 Fingerprinting random graphs

For the random graph DSJC1000.5.col.b, we color it on a Sun ULTRA-5 workstation and get five different solutions. Then we apply the proposed fingerprinting techniques on the original graph and color the resulting graphs again to get 5 solutions. The average and the best number of colors for each test are reported in Table 3. The last column shows the number of solutions can be derived from each single solution, recall that these solutions are guaranteed different. The run-time for coloring the original graph is about 16 hours, and those for the fingerprinted graphs are $14 \sim 19$ hours on the same system.

Tests	Average	Best	Number of Solutions
Original Graph	85.8	85	1
Duplicate 10 Vertices	86.8	86	$2^{10} = 1024$
Duplicate 25 Vertices	87.8	87	$2^{25} \approx 3.355 \times 10^7$
Duplicate 50 Vertices	89.8	89	$2^{50} \approx 1.126 \times 10^{15}$
Create 5 Bridges	87.2	87	$4^5 = 1024$
Create 12 Bridges	89.2	89	$4^{12} \approx 1.678 \times 10^7$
Manipulate 4 Triangles	89.4	89	$6^4 = 1296$
Manipulate 10 Triangles	104	101	$6^{10} \approx 6.047 \times 10^7$

Table 3: Coloring the fingerprinted graph DSJC1000.5.col.b.

Though the run-time overhead can be ignored, the degradation of solution cannot. Graph DSJC1000.5.col.b has similar local structure everywhere by its nature. No matter which fingerprinting technique we use, we will make some part over-constrained and this causes the extra-color overhead.

6.2 Fingerprinting reallife benchmark graphs

To show the effectiveness of our proposed techniques, we fingerprint the reallife benchmark graphs in three different ways, which all promising different solutions to the order of 10^7 (10^{15} for *test2* and *test4*). Both original graphs and fingerprinted graphs can be colored in a few seconds on the same Sun ULTRA-5 workstation. The run-time overhead is negligible.

Table 3 reports the details on coloring the fingerprinted graphs. The first two columns are the instances and their optimal coloring. The next six columns are:

test1: select 25 vertices randomly and duplicate them.

test2: select 50 vertices randomly and duplicate them.

test3: repeat test1 with 25 carefully selected vertices.

test4: repeat test2 with 50 carefully selected vertices.

test5: apply bridge construction on 12 random pair of unconnected vertices.

test6: manipulate 10 random triangles.

Instance	opt.	test1	test2	test3	test4	test5	test6
# of solutions	1	2^{25} 3.3e07	2^{50} 1.1e15	2^{25} 3.3e07	2^{50} 1.1e15	4^{12} 1.6e07	6^{10} 6.0e07
fpsol2.i.1.col	65	66	66	**65**	**65**	**65**	78
fpsol2.i.2.col	30	31	34	**30**	**30**	**31**	42
fpsol2.i.3.col	30	31	32	**30**	**30**	**31**	44
inithx.i.1.col	54	54	56	**54**	**54**	**54**	60
inithx.i.2.col	31	33	33	**31**	**31**	32	41
inithx.i.3.col	31	33	34	**31**	**31**	32	40
mulsol.i.1.col	49	54	58	**49**	**49**	57	57
mulsol.i.2.col	31	37	38	**31**	**31**	36	36
mulsol.i.3.col	31	37	40	**31**	**31**	37	38
mulsol.i.4.col	31	37	39	**31**	**31**	35	40
mulsol.i.5.col	31	38	40	**31**	**31**	36	36
zeroin.i.1.col	49	53	56	**49**	51	53	60
zeroin.i.2.col	30	32	33	**30**	**30**	32	42
zeroin.i.3.col	30	32	32	**30**	**30**	33	43

Table 4: Coloring the fingerprinted benchmark graphs.

In *test1* and *test2*, the overhead is significant, the reason is that we pick the vertices completely randomly. If we choose one from a clique of size k, a new clique of size $k+1$ will be created by duplicating a new vertex which makes the graph over-constrained. On the other hand, selecting isolated vertices only produce trivial solutions. Based on these observations, in *test3* and *test 4*, we avoid isolated vertices and those from large cliques. In all instances but one (*zeroin.i.1.col* with 50 vertices duplicated) there is no extra-color overhead.

The bridge construction method works fine for the *fpsol2* and *inithx* type of graphs, but bring unacceptable overhead to the other two. This is because that the *mulsol* and *zeroin* graphs are relatively small, consequently their solution spaces are small and to have the same amount of solutions, extra colors have to be introduced.

The clique manipulation technique is subtle than the previous ones, but it introduces overhead. When we select small cliques, most likely we will choose one from a large clique and possibly make the clique larger and the graph more difficult to be colored. For example, there are 5 triangles in Figure 6, one is the triangle on the right, the other four are from the clique of size 4. When we choose a triangle, with 80% we will pick one from the clique of size 4.

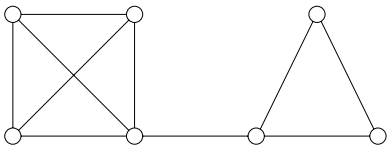


Figure 6: Choosing a triangle from a graph.

For real-life graphs, the local structure of the graph is different from place to place. More specifically, the constraints are not the same. We can exploit this unbalance and select (according to the owner's information if we want to watermark the solution as well) less-constrained part to apply the fingerprinting techniques. The above results show the effectiveness of this approach.

7 Conclusions

Fingerprinting is one of the most powerful and efficient methods to discourage illegal distribution. A fingerprinted IP will not directly prevent misuse of the IP, but will allow the IP provider to detect the source of the redistributed IP and therefore trace the traitor.

The existing fingerprinting IPP techniques have shortcomings: they are either applicable only to a very restricted set of problems[10] or unable to guarantee distinguish copies [5]. We propose a general technique which enables fingerprinting at all level of design process, is applicable to an arbitrary optimization step, and produces numbers of distinct solutions with high quality.

Our key idea is to superimpose additional constraints on the problem formulation so to guarantee that the final solution can be in a straightforward way translated into k different high quality solutions. We implemented this on the NP-complete GC problem and tested on a number of standard benchmarks. Fingerprinting the random graphs introduced overhead, while for graphs generated from real-life register allocation problems, we successfully created millions of distinct optimal solutions with no run-time overhead.

References

[1] I. Biehl, and B. Meyer. *Protocols for Collusion-Secure Asymmetric Fingerprinting.* STACS'97, Proceedings of 14th Annual Symposium on Theoretical Aspect of Computer Science. Reischuk, and Morvan (Eds.), Springer-Verlag pp. 399-412 1997.

[2] D. Boneh, and J. Shaw. *Collusion-Secure Fingerprinting for Digital Data.* Advances in Cryptology - CRYPTO'95, Proceedings of 15th annual International Cryptology Conference. Coppersmith (Ed.), Springer-Verlag, pp. 452-465 1995.

[3] L. Boney, A.H. Tewfik, and K.N. Hamdy. *Digital watermark for audio signals.* International Conference on Multimedia Computing and Systems, pp. 473-480, 1996.

[4] E. Charbon. *Hierarchical Watermarking in IC Design.* IEEE 1998 Custom Integrated Circuits Conference, pp. 295-298, 1998.

[5] A.E. Caldwell, H. Choi, A.B. Kahng, S. Mantik, M. Potkonjak, G. Qu, and J.L. Wong. *Effective Iterative Techniques for Fingerprinting Design IP.* 36th Design Automation Conference Proceedings, pp. 843-848, 1999.

[6] M.T.Chao, and J.Franco. *Probabilistic Analysis of Two Heuristics for the 3-Satisfiability Problem.* SIAM Journal of Computing, Vol.15, No.4 pp. 1106-1118, 1986.

[7] B. Chor, A. Fiat, and M. Naor. *Tracing Traitors.* Advances in Cryptology - CRYPTO'94, Proceedings of 14th annual International Cryptology Conference. Desmedt (Ed.), Springer-Verlag, pp. 257-270, 1994.

[8] I.J. Cox, J. Kilian, T. Leighton, and T. Shamoon. *A secure, imperceptible yet perceptually salient, spread spectrum watermark for multimedia.* Southcon, pp. 192-197, 1996.

[9] A.B. Kahng, J. Lach, W.H. Magione-Smith, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, H. Wang and G. Wolfe. *Watermarking Techniques for Intellectual Property Protection.* 35th Design Automation Conference Proceedings, pp. 776-781, 1998.

[10] J.Lach, W.H.Mangione-Smith, and M.Potkonjak. *FPGA Fingerprinting Techniques for Protecting Intellectual Property.* Proceedings of CICC, 1998.

[11] A.L. Oliveira. *Robust Techniques for Watermarking Sequential Circuit Designs.* 36th Design Automation Conference Proceedings, pp. 837-842, 1999.

[12] B. Pfitzmann, and M. Schunter. *Asymmetric Fingerprinting.* Advances in Cryptology - EUROCRYPT'96, Proceedings of International Conference on the Theory and Application of Cryptographic Techniques. Maurer (Ed.), Springer-Verlag, pp. 84-95, 1996.

[13] G. Qu, and M. Potkonjak. *Analysis of Watermarking Techniques for Graph Coloring Problem.* IEEE/ACM International Conference on Computer Aided Design Proceedings, 1998.

[14] N.R. Wagner. *Fingerprinting.* Proceedings of the 1983 Symposium on Security and Privacy, IEEE Computer Society, pp. 18-22, 1983.

[15] VSI Alliance. *System Chip Letter.* Issue 2, Summer 1998.