Effective Iterative Techniques for Fingerprinting Design IP*

Andrew E. Caldwell, Hyun-Jin Choi, Andrew B. Kahng,
Stefanus Mantik, Miodrag Potkonjak, Gang Qu and Jennifer L. Wong

UCLA Computer Science Dept., Los Angeles, CA 90095-1596

Abstract

While previous watermarking-based approaches to intellectual property protection (IPP) have asymmetrically emphasized the IP provider's rights, the true goal of IPP is to ensure the rights of both the IP provider and the IP buyer. Symmetric fingerprinting schemes have been widely and effectively used to achieve this goal; however, their application domain has been restricted only to static artifacts, such as image and audio. In this paper, we propose the first generic symmetric fingerprinting technique which can be applied to an arbitrary optimization/synthesis problem and, therefore, to hardware and software intellectual property. The key idea is to apply iterative optimization in an incremental fashion to solve a fingerprinted instance; this leverages the optimization effort already spent in obtaining a previous solution, yet generates a uniquely fingerprinted new solution. We use this approach as the basis for developing specific fingerprinting techniques for four important problems in VLSI CAD: partitioning, graph coloring, satisfiability, and standard-cell placement. We demonstrate the effectiveness of our fingerprinting techniques on a number of standard benchmarks for these tasks. Our approach provides an effective tradeoff between runtime and resilience against collusion.

1 Introduction

With rapid deployment of new process technologies, shorter time-to-market design requirements, and advances in CAD tool capability, core-based design and software reuse methodologies have attracted a great deal of industrial and academic interest. *Intellectual property protection* (IPP) techniques are an unavoidable prerequisite for development and adoption of reuse-based system integration business models. In such reuse-based IP business models, as well as the related IPP model, there are two basic types of legal entities involved in an IP transation: *provider* (*seller*, *owner*) and *buyer* (*user*). The goal of IPP is to protect the rights of both the provider and the buyer.

Recently, a number of *watermarking-based* IPP techniques have been proposed [8, 10, 12]. All of these techniques are based on a single fundamental concept, namely, to translate the provider's signature as additional design *constraints* and add these constraints into the design process. The effectiveness of this concept has been shown at various stages of the VLSI design process ranging from behavioral synthesis to physical design [8, 10, 12, 13]. However, no protection for the IP buyer is afforded.

To discourage piracy and unauthorized redistribution, it is not enough to protect the ownership rights of the IP provider: the *buyer's* legal ownership of a given piece of IP must symmetrically be protected as well. The IP provider desires the ability to trace a dishonest buyer from unauthorized resold copies of the IP. This is achieved by embedding the provider's signature into the design, and additionally embedding a unique signature to each realization of the design. Similarly, the IP buyer desires protection from being "framed" by other dishonest buyers working in collusion, or by a dishonest provider who sells extra copies of the IP and then attempts to blame the buyer. The buyer can provide the IP provider with his signature which is encrypted using the buyer's public key. He can easily check whether the purchased design indeed contains this signature. Since the buyer is the only entity who can interpret the signature (using his secret key), he is also protected in the sense that now the provider can not resell the IP without the buyer's permission.

Such symmetric protection of the provider's and buyer's rights is afforded by a *fingerprinting* methodology, whereby the IP provider *fingerprints* and delivers to each buyer a unique copy of functionally identical IP. The difficulty is that the IP provider most often cannot afford to apply a given watermarking technique with each buyer's signature and repeat the entire design process: creating a large number of different high-quality solutions from scratch has a clear time and cost overhead. Therefore, we require protocols that can provide a number of distinct versions of the same IP with reasonable amortized design effort.

The first IP fingerprinting technique in the literature is due to Lach et al. [13]. Their approach is based on solution partitioning. By partitioning an initial solution into a large number of parts and by providing for each part several different realizations, one can realize a fingerprinting scheme with relatively low performance impact for their application (a restricted FPGA mapping problem). However, the technique of [13] cannot be applied to design steps that do not have natural geometric structure and that are sensitive to the cost of the solution. More importantly, the technique has relatively low resilience against collusion attacks since it produces solutions with identical global structure (cf. the work of Boneh and Shaw [4]). Finally, the time overhead associated with creating fingerprinted solutions is relatively high.

Contributions of This Paper

In this paper, we propose a generic fingerprinting methodology that applies to arbitrary optimization/synthesis problems, and that combines existing watermarking techniques and iterative approaches to solving optimization problems. Our approach allows the IP owner/provider to design the IP with his (provider's) watermark embedded, in order to obtain an initial "seed" solution. We may view this initial design as a "from-scratch optimization". Then, for each IP buyer, a new *fingerprinted* optimization instance is created based on the buyer's fingerprint and some knowledge of the current seed solution. Solving this new fingerprinted instance with an "in-

*This research was supported in part by NSF under grant CCB-9734166, and by a grant from Cadence Design Systems, Inc.

cremental optimization" yields a different but functionally identical fingerprinted IP, and is inexpensive because it leverages the design optimization effort that is inherent in the seed solution.

As we will see later, our use of fingerprinted instances together with incremental optimization has several subtleties. (1) Fingerprinting the instance in some sense corresponds to using a smoother, more continuous version of the constraint-based watermarking approach. (2) The approach relies on the property that good solutions of the fingerprinted instance will also be good solutions with respect to the original objective. (3) The one-way nature of the iterative optimization heuristic itself (i.e., given a solution, it is very difficult to determine the exact instance for which the solution is a local minimum) can be useful in the authentication protocol.

In the following section, we review the relevant literature on IP watermarking, fingerprinting, and iterative optimization. Section 3 then sets out the requirements for a viable IP fingerprinting protocol. In Section 4, we propose our new fingerprinting approach, and illustrate its application to four distinct CAD optimizations: hypergraph partitioning, graph coloring, satisfiability, and standard-cell placement. Section 5 presents experimental results showing that the proposed techniques maintain good solution quality while providing an effective tradeoff between runtime and resilience against collusion. We conclude in Section 6 with directions for future research.

2 Related Work

We now review the necessary background for IP watermarking, fingerprinting, and iterative optimization.

2.1 IP Watermarking

A major characteristic of IP watermarking, as distinguished from artifact watermarking, is that it must maintain the correct functionality of the IP. This is the main reason why IP watermarking is not trivially achievable. The *constraint-based watermarking* technique [9] translates a to-be-embedded signature into a set of additional constraints during the design and implementation of the IP, in order to uniquely encode the author's signature into the IP. The effectiveness of this technique lies in the large solution space of the optimization problem that corresponds to the design of the IP: (i) the author's signature is added via extra constraints that reduce the solution space, and (ii) ownership is typically proved via the exceptionally small probability of obtaining a given solution from the initial solution space without the benefit of the signature constraints. The methodology is mathematically sound [16] and has been shown to yield strong proofs of authorship with little or no loss of solution quality, at the level of behavior [8], logic synthesis and physical design [12] [10], as well as in FPGA design [13].[1]

2.2 Fingerprinting

Fingerprints have been used for human identification for a long time because of their uniqueness. Protocols have been developed for adding fingerprint-like marks into digital data to protect both the provider and the buyers [3, 4, 15]. Such marks are made by introducing minute errors to the original copy, with such errors being so insignificant that their effect is negligible. All of these techniques are aimed at protecting *artifacts*, such as digital data, image, and audio/video streams. This is very different from protecting *IP*: since a minor error can change the functionality of the IP and render the entire design useless, IP fingerprinting cannot be achieved in the same way.

To the best of our knowledge, only one published work addresses IP protection using fingerprinting [13]. The approach is to partition the problem into small parts, and impose constraints as needed to make solutions for each part "connectable". Multiple solutions are found independently for each part, and a solution to the original problem can be constructed by mixing and matching these solutions according to the buyer's fingerprint. However, the method is relatively impractical (the problem must have a specific (usually, geometric) structure, the approach can affect solution cost significantly, and it is vulnerable to collusion since fingerprinted solutions all have the same structure). An analogous approach would be to introduce a set of independently relaxable constraints before solving the problem. Then, once a solution is found, relaxing each constraint independently guarantees that a number of distinct solutions can be derived.[2] The runtime overhead is almost zero, but many similarities are expected among fingerprinted solutions, making the approach vulnerable to collusion.

2.3 Iterative Optimization Techniques

An instance of finite global optimization has a finite solution set S and a real-valued cost function $f : S \rightarrow \Re$. Without loss of generality, global optimization seeks a solution $s^* \in S$ which minimizes f, i.e., $f(s^*) \leq f(s)\ \forall s \in S$. This framework applies to most combinatorial domains (scheduling, coloring, partitioning, quadratic assignment, etc.); continuous optimizations can also be discretized to yield finite instances. Many optimization problems are NP-hard [7], and hence heuristic methods are often applied which use an iterative approach broadly described by the iterative global optimization template of Figure 1.

Typically, s' in Line 2 of Figure 1 is generated by a perturbation to s_i, i.e., $s' \in N(s_i)$ where $N(s_i)$ indicates the *neighborhood*, or set of all possible "neighbor" solutions, of s_i under a given neighborhood operator. Example operators include changing a vertex's color in graph coloring; swapping two cells in standard-cell placement; moving a vertex to a different partition in graph partitioning; etc. The collection of neighborhoods $N(s_i)$ implicitly defines a topology over S, which we denote as the *neighborhood structure*, N. Together with N, the cost function f defines a *cost surface* over the neighborhood topology, and iterative optimization searches this surface for (an approximation to) a globally minimum solution. Each iteration of Lines 2 through 4 is a *step* in the algorithm; the sequence of steps from step $i = 0$ until the algorithm terminates in Line 5 is a *run* of the iterative optimization algorithm.

We make two observations:

- Steps 2-4 of Figure 1 can be hierarchically applied to create very complicated metaheuristics. For example, the Kernighan-Lin [11] and Fiduccia-Mattheyses [6] graph partitioning heuristics are both greedy iterative optimizers with respect to a complicated *pass* move that is itself a move-based iterative optimization.[3]

[1] The constraint-based watermarking technique implicitly requires a large solution space, since it relies on transparent introduction of new constraints that are then verified in "binary" fashion (i.e., a given constraint is either satisfied or not satisfied in the final solution). The constraint-based approach is also more suitable to large problem instances, since structural elements of the instance are changed by the constraints (e.g., edges in a graph representation are added or deleted) and we would like such changes to be unobtrusive. One aspect of our proposed fingerprinting methodology, below, is that it is less "binary": it changes the design optimization instance more smoothly and yields a richer set of possible watermarked solutions.

[2] This is similar to the approach of [13] in that a fingerprinted solution is obtained by independently combining elements of the solution (either solutions to sub-parts, or independent relaxations of constraints).

[3] For example, the Fiduccia-Mattheyses algorithm starts with a possibly random solution and changes the solution by a sequence of moves which are organized as *passes*.

Iterative Global Optimization

1. **for** $i = 0$ to $+\infty$
2. Given the current solution s_i, generate a new trial solution s'
3. Decide whether to set $s_{i+1} = s_i$ or $s_{i+1} = s'$
4. **if** a stopping condition is satisfied
5. **return** the best solution found

Figure 1: Basic template for iterative global optimization.

- The complexity of the metaheuristic and its sensitivity to perturbations of the instance can be a vehicle for IPP: given a solution (say, an assignment of vertices to partitions) it is typically extraordinarily difficult to identify the instance (say, the weighted edges of a graph over the vertices) for which a given metaheuristic would return the solution.

3 Fingerprinting Objectives

A fingerprint, being the signature of the buyer, should satisfy all the requirements of any effective watermark:

- **High credibility.** The fingerprint should be readily detectable in proving legal ownership, and the probability of coincidence should be low.
- **Low overhead.** Once the demand for fingerprinted solutions exceeds the number of available good solutions, the solution quality will necessarily degrade. Nevertheless, we seek to minimize the impact of fingerprinting on the quality of the software or design.
- **Resilience.** The fingerprint should be difficult or impossible to remove without complete knowledge of the software or design.
- **Transparency.** The addition of fingerprints to software and designs should be completely transparent, so that fingerprinting can be used with existing design tools.
- **Part Protection.** Ideally, a good fingerprint should be distributed all over the software or design in order to identify the buyer from any part of it.

At the same time, the IPP business model implies that fingerprints have additional mandatory attributes:

- **Collusion-secure.** Different users will receive different copies of the solution with their own fingerprints embedded. These fingerprints should be embedded in such a way that it is not only difficult to remove them, but also difficult to forge a new fingerprint from existing ones (i.e., the fingerprinted solutions should be structurally diverse).
- **Runtime.** The runtime for embedding the fingerprint should be much less than the runtime for solving the problem from scratch.
- **Preserving watermarks.** Fingerprinting should not diminish the strength of the author's watermark.

A move changes the assignment of a vertex from its current partition to another partition. At the beginning of a pass, all vertices are free to move (i.e., they are *unlocked*), and each possible move is labeled with the immediate change in total cost it would cause; this is called the *gain* of the move (positive gains reduce solution cost, while negative gains increase it). Iteratively, a move with highest gain is selected and executed, and the moving vertex is *locked*, i.e., is not allowed to move again during that pass. Since moving a vertex can change gains of adjacent vertices, after a move is executed all affected gains are updated. Selection and execution of a best-gain move, followed by gain update, are repeated until every vertex is locked. Then, the best solution seen during the pass is adopted as the starting solution of the next pass. The algorithm terminates when a pass fails to improve solution quality.

From the above objectives, we extract the following key requirements for fingerprinting protocols:

- A fingerprinting protocol must be capable of generating solutions that are "far away" from each other. If solutions are too similar, it will be difficult for the seller to identify distinct buyers. In most problems, there exist generally accepted definitions for distance or similarity between different solutions.
- A fingerprinting protocol should be non-intrusive to existing design optimization algorithms, so that it can be easily integrated with existing software tool flows.
- The cost of the fingerprinting protocol should be kept as low as possible. Ideally, it should be negligible compared to the original design effort.

4 A New Fingerprinting Approach

To maintain reasonable runtime while producing a large number of fingerprinted solutions, we will exploit the availability of iterative heuristics for difficult optimizations. Notably, we propose to apply such heuristics (i) in an *incremental* fashion, and (ii) to design optimization instances that have been perturbed according to a buyer's signature. The basic approach is as follows.[4]

1. Given an initial instance I_0 containing the provider's watermark, generate a watermarked initial solution S_0 using a *from-scratch* optimization.
2. For $j = 1$ to n (n is the number of buyers)
3. Create a fingerprinted instance I_j with fingerprint F_j added into I_0
4. Starting from S_0 as the initial solution, apply an *incremental* optimization to generate a new fingerprinted solution S_j for the fingerprinted instance I_j.

Given a design instance I_0, our approach starts by generating an initial watermarked solution S_0 using an (iterative) optimization heuristic in "from-scratch" mode. For a given buyer, we embed the buyer's signature into the design as a fingerprint (e.g., by perturbing the weights of graph edges), which yields a fingerprinted instance I_j. Starting from S_0 as an initial solution,[5] we then perform an *incremental* iterative optimization step to obtain solution S_j for instance I_j.

We observe that the iterative optimization heuristic will be applied using a known high-quality solution as a starting point, so that the runtime until the stopping criterion is reached (e.g., arriving at a local minimum) will be much less than that of a from-scratch optimization. Essentially, we leverage the design optimization effort that is inherent in the "seed" solution S_0.

Another important point is that fingerprinting the instance subtly changes its optimization cost surface. There are several advantages to this: (i) changing the cost surface prevents the iterative optimizer from falling into the same local minima as before; (ii) the optimization cost surface together with the iterative heuristic will itself fingerprint the design;[6] and (iii) as noted in the metaheuris-

[4] We do not discuss the mechanics of encoding a buyer's signature as a set of weights or constraints. Such techniques (using, for example, the cryptographic hash function MD5, the public-key cryptosystem RSA, and the stream cipher RC4) have been discussed at length in the recent literature on IPP (e.g., [10]).

[5] Alternatively, we could use S_{j-1} as the initial solution.

[6] As noted above, it is exceedingly difficult to reverse-engineer the particular weighting of the instance for which a given solution is a local minimum. For some fingerprinting protocols, this can be useful for authentication. In the partitioning and standard-cell placement fingerprinting approaches below, which use weights rather than constraints, authentication will entail confirming that the solution IP is a local minimum with respect to a particular weighting (i.e., fingerprinted version) of the instance.

tics literature, such changes can actually lead to improved solution quality.[7]

In the remainder of this section, we develop specific fingerprinting approaches for four classes of VLSI CAD optimizations.

4.1 Partitioning

Given a hyperedge- and vertex-weighted hypergraph $H = (V, E)$, a *k-way partitioning* of V assigns the vertices to k disjoint nonempty partitions. The *k-way partitioning problem* seeks to minimize a given objective function $c(P^k)$ whose arguments are partitionings. A standard objective function is *cut size*, i.e., the number of hyperedges whose vertices are not all in a single partition. Constraints are typically imposed on the partitioning solution, and make the problem difficult. For example, the total vertex weight in each partition may be limited (*balance constraints*), which results in an NP-hard formulation [7]. To achieve flexibility and speed in addressing various formulations, move-based iterative optimization heuristics are typically used, notably the Fiduccia-Mattheyses (FM) heuristic [6]. In our partitioning testbed, we use the recent CLIP FM variant [5] and the net cut cost function.

For a given partitioning instance I_0, we iteratively construct a sequence of fingerprinted solutions according to the following steps.

1. Generate an initial partitioning solution S_0 by finding the best solution out of 40 starts of CLIP FM for instance I_0.
2. Reset all hyperedge weights to 20.
3. According to the j^{th} user's fingerprint, select a subset $E' \subset E$ of size equal to some percentage of the total number of hyperedges in H, and increment the weight of each hyperedge $e \in E'$ by +/- 19 (also according to the user's fingerprint). This yields instance I_j.
4. Partition the hypergraph instance I_j using a single start of CLIP FM, using S_0 (the initial unfingerprinted solution) as the starting solution.[8] This yields the fingerprinted solution S_j.
5. If another fingerprinted solution is needed, return to Step 2.

4.2 Satisfiability

The boolean satisfiability problem (**SAT**) seeks to decide, for a given formula, whether there is a truth assignment for its variables that makes the formula true. We necessarily assume that the SAT instance to be protected is satisfiable and that there is a large enough solution space to accommodate multiple fingerprinted solutions.

Given a formula F on a set of boolean variables V, we iteratively construct a sequence of fingerprinted solutions according to the following steps.

1. Solve F and store the value for each variable of V. This is solution S_0.
2. According to the j^{th} user's fingerprint, select a subset $V' \subset V$.
3. Keep the current assignment for variables in V' and create a new formula F' as follows:
   - Delete from F all clauses that are satisfied by the assignment to V'.
   - Delete from the rest of the formula all literals that are in V'.
   - Apply an existing SAT watermarking technique to embed the user's fingerprint into the remaining formula.
4. Solve F' and get an assignment to all the variables in $V - V'$.
5. Combine the new assignment to $V - V'$ and the old assignment to V'. This is the fingerprinted solution S_j to F; store it as the current assignment.
6. If another fingerprinted solution is needed, return to Step 2.

4.3 Graph Coloring

The NP-hard graph vertex coloring (**GC**) optimization seeks to color a given graph with as few colors as possible, such that no two adjacent vertices receive the same color. Given a graph $G(V, E)$, we iteratively construct a sequence of fingerprinted GC solutions according to the following steps.

1. Apply a graph coloring heuristic to color G, thus obtaining a k-color scheme as solution S_0. I.e., the vertex set V is partitioned into k independent sets, $\{C_1, \ldots, C_k\}$, each marked by a different color.
2. According to the j^{th} user's fingerprint, select $\{C_{i_1}, \ldots, C_{i_l}\} \subset \{C_1, \ldots, C_k\}$.
3. Create a new graph G' as follows:
   - Among all independent sets in $\{C_1, \ldots, C_k\} - \{C_{i_1}, \ldots, C_{i_l}\}$, for each independent set that is not maximal, create a new vertex and connect it to all the vertices that are neighbors of this independent set.
   - Delete vertices in $\{C_1, \ldots, C_k\} - \{C_{i_1}, \ldots, C_{i_l}\}$ along with their associated edges.
   - Apply an existing GC watermarking technique to embed the user's fingerprint and denote the resulting graph by G'.
4. Apply a graph coloring heuristic to color G' and get a coloring solution.
5. Replace the new vertices created in Step 3 by the corresponding independent sets, and append the rest of the maximal independent sets. Report this solution as S_j and retain it as the current solution.
6. If another fingerprinted solution is needed, return to Step 2.

4.4 Standard-Cell Placement

The standard-cell placement problem seeks to place each cell of a gate-level netlist onto a legal site, such that no two cells overlap and the wirelength of the interconnections is minimized. We iteratively construct a sequence of fingerprinted placement solutions according to the following steps (note that our approach is compatible with the LEF/DEF and Cadence QPlace based constraint-based watermarking flow presented in [10]).

1. Given an instance I_0 in LEF/DEF format, apply the placer (Cadence QPlace version 4.1.34) to generate an initial placement solution S_0.
2. Reset the weights of all signal nets to 1.
3. According to the j^{th} user's fingerprint, select a subset $N' \subset N$ of the signal nets in the design, and set the weight of each net in N' to 10. This yields a fingerprinted instance I_j.
4. Incrementally re-place the design, starting from the current solution solution S_{j-1} and using the new net weighting. This is achieved by invoking the *Incremental Mode* of the *QPlace* tool, and yields the fingerprinted placement solution S_j.
5. Save the new placement solution S_j as the current solution.
6. If another fingerprinted solution is needed, return to Step 2.

[7] We refer the reader to the development of *problem-space* and *heuristic-space* methods in the metaheuristics literature [17] [14]. Such methods perturb a given instance to allow a given optimization heuristic to escape local minima. The perturbations induce alternate cost surfaces that one hopes are correlated to the original cost surface (so that good solutions in the new surface correspond to good solutions in the original), yet which have sufficiently different structure (so that the optimization heuristic can move away from the previous local minimum).

[8] We use only one start since our CLIP FM implementation is deterministic; multiple starts from S_0 will yield the same local minimum.

Test Case:	IBM01	IBM02	IBM03
Number of Vertices	12752	19601	23136
Number of Hyperedges	14111	19584	27401

Table 1: Test cases for partitioning experiments.

Test Cases	S_0 Cost		S_0 CPU Time		S_i Cost		S_i CPU Time		Hamming Dist.	
	Max	Ave	Max	Ave	Max	Ave	Max	Ave	Min	Ave
IBM01	308	252.2	261	187.6	307.2	253.8	3.25	2.42	16.3	71.1
IBM02	296	272.0	379	329.8	278.5	273.1	15.2	8.4	7.3	41.2
IBM03	1047	881.5	808	695.5	912.5	867.5	36.1	18.4	66.6	263.4

Table 2: Results for the fingerprinting flow on three standard bi-partitioning test cases. Tests were run using actual cell areas, and a partition area balance tolerance of 10%. Each trial consists of generating an initial solution, then generating a sequence of 20 fingerprinted solutions. All results are averages over 20 independent trials.

5 Experimental Results

5.1 Partitioning

We test our fingerprinting method on 3 standard test cases from the ISPD-98 Benchmark Suite [2] [1]. These correspond to internal IBM designs that have been recently released to the VLSI CAD community. We apply the CLIP FM partitioner with a 10% balance constraint, and actual cell areas as vertex weights. For each test case, a single experimental trial generates an initial solution, followed by a sequence of 20 fingerprinted solutions (i.e., we go through Step 2 of the method in Section 4.1 a total of 20 times). Table 2 reports the average results of 20 independent trials.[9] We report the maximum and average solution cost for the initial solutions S_0, as well as the maximum and average solution costs for the fingerprinted solutions S_i. We also report the maximum and average CPU times required to generate an initial solution S_0 or a fingerprinted solution S_i. (All CPU times that we report are for a 300MHz Sun Ultra-10 running Solaris 2.6.) Finally, we report the minimum and average Hamming distances (i.e., number of transpositions required to transform one solution into another) over all $C(21,2)$ pairs among the solutions $S_0, S_1, \ldots, S_{20}$. The data show that the fingerprinted solutions: (i) require much less CPU to generate than the original solutions (by factors ranging from 18 to 77); (ii) are reasonably distinct from each other and from the original solutions; and (iii) can even have better average quality than the original solutions (which we attribute to the similarity between our fingerprinting methodology and the problem-space iterative optimization metaheuristic [14]).

5.2 Satisfiability

Our SAT instances are generated from the problem of inferring the logic in an 8-input, 1-output "blackbox" [18]. All instances that we use are satisfiable and we use **WalkSAT** as the solver [19]. As described in Section 4.2, we first solve each instance once to obtain the "seed" solution. We then retain the assignments for k% of the variables and re-solve the smaller instance involving only the remaining variables. We find five solutions for the smaller fingerprinted instances and compare them to the original solution. The distance of two solutions $S = \{s_1, s_2, \ldots, s_n\}$ and $T = \{t_1, t_2, \ldots, t_n\}$ is defined as: $dist(S,T) = \sum_{i=1}^{n} |s_i - t_i|$.

Table 3 reports the results when we maintain 20%, 30%, and 50% of the "seed" S_0 solution. From the last two rows, we can see that on average, we are able to achieve solutions which are around 20% different from the seed with a near 40% CPU time saving. Furthermore, the more variables we preserve from the seed, the smaller the new instance is. This allows us to trade off CPU time and vulnerability to collusion, i.e., preserving more of the seed results in less CPU time consumed, with the new solution being closer to the seed solution.

[9] Thus, most entries in the table are non-integer.

Original Solution			20% Preserved			30% Preserved			50% Preserved		
File	Var.	CPU	Var.	Dist.	CPU	Var.	Dist.	CPU	Var.	Dist.	CPU
ii8a1	66	8.0	52	20	6.6	46	16	7.2	33	21	6.8
ii8a2	180	9.2	144	35	7.0	126	33	6.8	90	24	7.0
ii8a3	264	9.4	211	48	8.4	184	43	7.4	132	38	7.4
ii8a4	396	12.8	316	78	8.4	277	60	7.8	198	56	9.2
ii8b1	336	9.6	268	80	9.0	235	53	7.4	168	55	8.4
ii8b2	576	13.8	460	146	7.6	403	138	7.2	288	147	7.0
ii8b3	816	18.8	652	203	7.8	571	198	8.4	408	224	8.0
ii8b4	1068	30.8	854	272	8.2	747	246	9.0	534	277	7.4
ii8c1	510	12.4	408	92	8.0	357	103	8.4	255	72	7.8
ii8c2	950	17.4	760	218	8.6	665	238	7.8	475	246	7.4
ii8d1	530	12.2	424	68	8.6	371	60	8.8	265	71	7.8
ii8d2	930	17.2	744	246	7.8	651	251	8.0	465	212	7.2
ii8e1	520	12.0	416	82	8.4	364	56	8.6	260	68	6.8
ii8e2	870	17.2	696	165	7.4	609	223	8.0	435	98	8.2
Ave. Distance (%)			-	22%	-	-	20%	-	-	19%	-
Ave. CPU Saving (%)			-	-	38%	-	-	39%	-	-	41%

Table 3: Number of undetermined variables (*Var.*), average distance from original solution (*Distance*), and average CPU time (in $1/100^{th}$s of a second) for fingerprinting SAT benchmarks.

5.3 Graph Coloring

We have implemented our proposed GC fingerprinting technique from Section 4.3 and applied it to the DIMACS challenge graph. This graph is a random graph with 1000 vertices and an edge probability slightly larger than $\frac{1}{2}$. It is believed that coloring this graph is hard and that the optimal solution is still open [18].

We colored the graph once and achieved an 86-color solution. Based on this "seed" solution, we choose various percentages of independent sets (**IS**) and create a new graph as described in Step 3 of the description in Section 4.3. For each new graph, we color it 5 times. Parameters of the new graphs and fingerprinted solutions to the original graph, along with average runtimes, are reported in Table 4. We again see a tradeoff between the quality of the solution and the credibility of the fingerprints. When we recolor more ISs, we can provide more convincing fingerprints, but it will take more time to color the graph. In any case, the savings over the original from-scratch runtimes is still significant.

	Nodes	Edges	Edge Prob.	solution		Run-time
				Ave.	Best	(hours)
Original Graph	1000	249826	0.5002	86	86	15.45
Recolor 20% ISs	222	13630	0.5556	86	86	1.10
Recolor 30% ISs	319	26908	0.5305	86	86	1.46
Recolor 40% ISs	397	40712	0.5179	86	86	2.02
Recolor 50% ISs	495	62726	0.5130	87	87	3.07
Recolor 60% ISs	593	89006	0.5071	87	87	4.55
Recolor 70% ISs	694	121591	0.5056	87.2	87	6.42

Table 4: Results for coloring the DIMACS challenge graph with iterative fingerprinting.

5.4 Standard-Cell Placement

For standard-cell placement, we have applied our fingerprinting technique to the four industry designs listed in Table 5. For each test case, we generate an initial solution S_0 and a sequence of 20 different fingerprinted solutions $S_1, \ldots, S_{20}$; for each fingerprinted solution, the previous fingerprinted solution is used as the initial solution for QPlace Incremental Mode. Table 6 presents a detailed

Test Case:	Test1	Test2	Test3	Test4
Number of Cells	3286	12133	12857	20577
Number of Nets	2902	11828	10880	25634

Table 5: Test cases for Standard-Cell Placement experiment.

Solution	1% Nets Weighted			2% Nets Weighted			5% Nets Weighted		
	Cost	Dist.	CPU	Cost	Dist.	CPU	Cost	Dist.	CPU
Orig(S0)	1.000	0	6:24	1.000	0	6:24	1.000	0	6:24
S1	0.994	3.532	2:58	1.001	3.757	2:58	1.020	3.946	2:58
S2	0.986	4.341	2:57	0.994	3.858	2:58	1.023	4.518	2:54
S3	0.981	4.666	2:58	0.997	4.032	3:03	1.016	4.536	2:51
S4	0.982	4.545	3:02	0.997	4.203	3:03	1.018	4.990	2:52
S5	0.979	4.807	3:01	0.986	4.574	2:57	1.005	5.095	2:54
S6	0.981	4.989	2:59	0.990	5.115	3:01	1.007	5.272	2:55
S7	0.971	5.121	3:30	0.982	5.359	3:33	1.014	5.543	3:28
S8	0.972	5.257	3:32	0.989	5.510	3:30	1.005	5.657	3:28
S9	0.970	5.513	3:38	0.979	5.804	3:32	1.004	5.972	3:29
S10	0.970	5.820	3:33	0.976	6.354	3:29	0.998	6.089	3:29
S11	0.970	6.011	3:34	0.977	6.253	3:30	1.005	6.248	3:30
S12	0.974	6.030	3:34	0.983	6.794	3:34	1.006	6.264	3:29
S13	0.974	6.738	3:35	0.974	6.554	3:34	0.998	6.356	3:30
S14	0.972	6.826	3:38	0.966	6.441	3:32	1.002	6.742	3:30
S15	0.978	6.975	3:36	0.972	6.354	3:32	1.004	7.106	3:28
S16	0.976	6.762	3:39	0.976	6.107	3:33	0.992	7.034	3:29
S17	0.978	6.911	3:38	0.975	6.654	3:34	0.992	7.129	3:27
S18	0.970	7.144	3:39	0.974	6.657	3:34	0.990	7.198	3:26
S19	0.974	7.254	3:37	0.969	6.440	3:28	0.995	7.161	3:27
S20	0.969	7.331	3:40	0.977	6.414	3:34	0.986	6.989	3:26

Table 6: Standard-cell placement fingerprinting results for the Test2 instance. We report CPU time (mm:ss) needed to generate each solution, as well as total wirelength costs normalized to the cost of the initial solution S_0. Manhattan distances from S_0 are given in 10^6 microns.

analysis of the solutions obtained for the Test2 instance. We measure the structural difference between solutions as "Manhattan Distance": the sum over all cells in the design of the Manhattan distance between the two placed locations for each cell. We see that a fingerprint that perturbs just 1% of the net weights achieves reasonably large Manhattan distance from S_0, and that the incremental optimization saves a significant amount of CPU versus the from-scratch optimization. Again, there is a "problem-space metaheuristic" effect in that the fingerprinted solutions are typically of higher quality than the original solution. A summary of results for all four test cases is given in Table 7. From this table we can see that we can reduce the time to generate the next fingerprinted solution while maintaining the quality as well as producing a unique solution.

Test cases	% weighted nets	Normalized Cost		CPU (sec)		Manhattan Dist.	
		Max	Ave	Max	Ave	Min	Ave
Test1	Original	1.000	1.000	97	97	0	0
	1%	1.009	0.986	53	51	0.530	0.861
	2%	1.003	0.987	54	50	0.509	0.886
	5%	1.024	0.992	58	53	0.801	1.084
Test2	Original	1.000	1.000	384	384	0	0
	1%	0.994	0.976	220	205	3.532	5.829
	2%	1.001	0.982	214	202	3.757	5.662
	5%	1.023	1.004	210	198	3.946	5.992
Test3	Original	1.000	1.000	344	344	0	0
	1%	1.000	0.988	168	164	3.232	4.048
	2%	1.007	0.995	165	162	3.239	4.123
	5%	1.015	1.009	166	163	3.503	4.423
Test4	Original	1.000	1.000	960	960	0	0
	1%	1.006	0.983	510	499	11.39	15.53
	2%	0.997	0.987	506	485	13.06	17.44
	5%	1.031	1.018	514	496	12.35	18.02

Table 7: Summary of results for fingerprinting of all four standard-cell placement instances. "Original" lines refer to the initial solutions S_0. All other lines refer to fingerprinted solutions S_i, $i > 0$. Manhattan distance is again expressed in 10^6 microns.

6 Conclusions

The key problem in fingerprinting for intellectual property protection is the tradeoff between collusion resiliency and runtime. In this paper, we have proposed a new generic fingerprinting technique for IPP of solutions to optimization/decision problems and, therefore, of hardware and software intellectual property. By judiciously exploiting partial solution reuse and the incremental application of iterative optimizers, we have developed fingerprinting techniques for partitioning, graph coloring, satisfiability and placement that simultaneously provide high collusion resiliency and low runtimes. Our current research seeks improved methods for fingerprinting, and also assesses the resiliency of our methods to various attacks.

References

[1] C. J. Alpert, "Partitioning Benchmarks for the VLSI CAD Community", `http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html`

[2] C. J. Alpert, "The ISPD-98 Circuit Benchmark Suite", *Proc. ACM/IEEE International Symposium on Physical Design*, April 98, pp. 80-85. See errata at `http://vlsicad.cs.ucla.edu/~cheese/errata.html`

[3] I. Biehl and B. Meyer, "Protocols for Collusion-Secure Asymmetric Fingerprinting", *Proc. 14th Annual Symposium on Theoretical Aspect of Computer Science*, Springer-Verlag, 1997, pp. 399-412.

[4] D. Boneh and J. Shaw, "Collusion-Secure Fingerprinting for Digital Data", *Proc. 15th annual International Cryptology Conference*, Springer-Verlag, 1995, pp. 452-465.

[5] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Proc. IEEE International Conference on Computer-Aided Design*, 1996, pp. 194-200.

[6] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", *Proc. ACM/IEEE Design Automation Conference*, 1982, pp. 175-181.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, New York, W. H. Freeman and Company, 1979.

[8] I. Hong and M. Potkonjak, "Behavioral Synthesis Techniques for Intellectual Property Protection", unpublished manuscript, 1997.

[9] 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", *Proc. ACM/IEEE Design Automation Conference*, June 1998, pp. 776-781.

[10] A. B. Kahng, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang and G. Wolfe, "Robust IP Watermarking Methodologies for Physical Design", *Proc. ACM/IEEE Design Automation Conference*, June 1998, pp. 782-787.

[11] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell System Tech. Journal* 49 (1970), pp. 291-307.

[12] D. Kirovski, Y. Hwang, M. Potkonjak and J. Cong, "Intellectual Property Protection by Watermarking Combinational Logic Synthesis Solutions", *Proc. IEEE/ACM International Conference on Computer Aided Design*, 1998.

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

[14] I. H. Osman and J. P. Kelly, eds., *Meta-Heuristics: Theory and Applications*, Kluwer, 1996.

[15] B. Pfitzmann, and M. Schunter, "Asymmetic Fingerprinting", *Proc. International Conference on the Theory and Application of Cryptographic Techniques*, Springer-Verlag, 1996, pp. 84-95.

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

[17] R. H. Storer, S. D. Wu and R. Vaccari, "New Search Spaces for Sequencing Problems With Application to Job Shop Scheduling", *Management Science* 38 (1992), pp. 1495-1509.

[18] http://dimacs.rutgers.edu/

[19] http://aida.intellektik.informatik.th-darmstadt.de/~hoos/SATLIB/