Full characterization of a class of graphs tailored for software watermarking[☆]

Lucila M. S. Bento[a,b], Davidson R. Boccardo[b], Raphael C. S. Machado[b], Vinícius G. Pereira de Sá[a], Jayme Luiz Szwarcfiter[a,b,c]

[a] *Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil*
[b] *Instituto Nacional de Metrologia, Qualidade e Tecnologia, Rio de Janeiro, Brasil*
[c] *COPPE Sistemas, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil*

Abstract

Digital watermarks have been regarded as a promising way to fight copyright violations in the software industry. In some graph-based watermarking schemes, identification data is disguised as control-flow graph of dummy code. Recently, Chroni and Nikolopoulos proposed an ingenious such scheme whereby an integer is encoded into a particular kind of permutation graph. We give a formal characterization of the class of graphs generated by their encoding function, a simpler decoding function, and a robust polynomial-time algorithm which restores watermarks with a constant number of missing edges whenever at all possible, providing a reasonable level of protection against distortive attacks.

Keywords: digital watermarking, linear-time algorithms, permutation graphs, robust algorithms, software security

[☆] Part of this paper was presented as an extended abstract entitled "Towards a provably resilient scheme for graph-based watermarking" at the 39th International Workshop on Graph Theoretic Concepts in Computer Science, WG 2013, and appeared in *Lecture Notes in Computer Science* **8165** (2013), 50–63.

Email addresses: lucilabento@ppgi.ufrj.br (Lucila M. S. Bento), drboccardo@inmetro.gov.br (Davidson R. Boccardo), rcmachado@inmetro.gov.br (Raphael C. S. Machado), vigusmao@dcc.ufrj.br (Vinícius G. Pereira de Sá), jayme@nce.ufrj.br (Jayme Luiz Szwarcfiter)

1. Introduction

The illegal reproduction of software has become a major concern for the industry. According to the Business Software Alliance, the commercial value of unlicensed software put into the world market in 2011 totaled 63.4 billion dollars [2]. To counter such practice, many promising methods have been devised, among which the idea of software watermarking.

The use of paper watermarks to prevent counterfeiting dates back to the thirteenth century. Generally speaking, watermarks are unique identifiers embedded into proprietary objects to enforce authenticity. In a digital object, particularly in a piece of software, a watermark may act not only as a certificate of authorship, but also as a means of tracing the original owner of the object, therefore discouraging piracy.

The first software watermark was proposed in 1996 by Davidson and Myrhvold [13], while the first watermarking scheme to exploit concepts of Graph Theory was formulated by Venkatesan, Vazirani and Sinha [22] in 2001. Their technique, whereby an integer was encoded as a special digraph to be disguised into the software's control-flow graph, was later patented [23]. Other original ideas, improvements and surveys on the available methods have been contributed by many authors ever since. See, for example, [9, 10, 11, 14, 20, 25, 26].

Willing to prevent the timely retrieval of the encoded identification data, malicious agents may attempt to tamper with the watermark. A watermark solution is therefore only as secure as it is able to resist attacks of various sorts. Naturally, a lot of research has been put up lately towards developing more resilient solutions as well as strengthening existing ones. This paper pursues this latter goal.

We consider the graph-based watermarking scheme introduced by Collberg, Kobourov, Carter and Thomborson [9], and afterwards developed and improved upon by Chroni and Nikolopoulos in a series of papers [3, 4, 5, 6, 7]. These latter authors proposed a watermark graph belonging to a subclass of the reducible permutation graphs introduced by the former authors. Though the mechanics of encoding and decoding the proposed watermark is well described in [5], such special subclass of reducible permutation graphs has not been fully characterized. We provide such a characterization, based solely on the topology of the graph. Furthermore, we formulate a simpler linear-time decoding algorithm and an algorithm to recover—in polynomial time—the encoded integer even after some edges have been maliciously removed.

This paper is organized as follows. In Section 2, we present some preliminary concepts related to graph-based software watermarking, including the most common forms of attacks. In Section 3, we recall the watermark from Chroni and Nikolopoulos, and we state a number of structural properties, the proofs of which we delay until Section 7 for the sake of readability. In Section 4, we define and characterize the family of canonical reducible permutation graphs, which correspond to the watermarks produced by Chroni and Nikolopoulos's encoding function. In Section 5, we formulate a simpler linear-time decoding algorithm. In Section 6, we propose a robust polynomial-time algorithm that, given a watermark with *whatever* constant number k of missing edges, either recovers the encoded data or proves that the watermark has become irremediably damaged. Finally, Section 7 contains the postponed proofs for the properties stated in Section 3, and Section 8 concludes the paper with our final remarks.

Throughout the text, we let $V(G)$ and $E(G)$ respectively denote, as usual, the vertex set and edge set of a given graph G. Also, we let $N_G^+(v)$ and $N_G^-(v)$ be the sets of out-neighbors and in-neighbors of vertex v in G, with $d_G^+(v)$ and $d_G^-(v)$ their respective sizes. If J is a subset of either $V(G)$ or $E(G)$, then $G - J$ corresponds to the graph obtained from G by the removal of J.

2. Graph-based software watermarking

Software watermarking schemes provide the necessary means of embedding identification data—typically a copyright notice or a customer number—into a piece of software. We refer to the identification data as the *identifier*, and we may regard it as an integer, for simplicity. In short, software watermarks are appropriate embeddings of surreptitious identifiers into computer programs, and they can be broadly divided into two categories: static and dynamic [8]. The former are embedded in the code, whereas the latter are embedded into the program's execution state at runtime.

A static, *graph-based* watermarking scheme usually consists of four algorithms:

- an *encoder*, which converts the identifier into a graph—the watermark;
- an *embedder*, a function whose input parameters are the software itself (either the binary code or the source code in some programming language), the intended watermark, and possibly some secret key, and whose output is a modified software containing the watermark;

- an *extractor*, which retrieves the watermark graph from the watermarked software; and
- a *decoder*, which extracts the identifier from the watermark;

To every computer program one can associate a directed graph representing the possible sequences of instructions, where each vertex corresponds to a maximal block of contiguous instructions. Such graph, called the *control-flow graph* (CFG) of the software [1], can be obtained by means of static analysis [17, 24]. What the embedder does is basically to insert dummy code into the program so that the intended watermark graph shows up as an induced subgraph of the CFG. The position of the watermark graph within the CFG is often determined as a function of a secret key. Knowledgeable of the secret key, the extractor retrieves that subgraph, which is then passed along to the decoding algorithm. Among the existing tools for embedding/extracting graph-based watermarks, we cite Collberg's SandMark project [12]. In this paper, we focus on the encoding/decoding algorithms described by Chroni and Nikolopoulos in [5].

Attacks. Among the several different kinds of attacks against graph-based watermarks, we list the following:

- *additive* attacks, in which other watermarks are inserted into the same object, generating ambiguity;
- *subtractive* attacks, in which the watermark is removed altogether; and
- *distortive* attacks, in which the watermark graph is modified to confound the decoder.

Additive and subtractive attacks can be precluded to a great extent by techniques of cryptography and software diversity [19]. On the other hand, distortive attacks—also known as *jamming* attacks—are more difficult to deal with. In some cases, the distortive attacker may even be able to reverse engineer the entire code and apply semantics-preserving modifications which modify the CFG structurally without affecting the software's functionalities. Distortive attacks are arguably the most important attack model one should worry about [22].

3. The watermark from Chroni and Nikolopoulos

We recall the encoding algorithm described in [5]. The index of the first element in all considered sequences is 1.

Let ω be a positive integer identifier, and n the size of the binary representation B of ω. Let also n_0 and n_1 be the number of 0's and 1's, respectively, in B, and let f_0 be the index of the leftmost 0 in B. The extended binary B^* is obtained by concatenating n digits 1, followed by the one's complement of B and by a single digit 0. We let $n^* = 2n + 1$ denote the size of B^*, and we define $Z_0 = (z_i^0)$, $i = 1, \ldots, n_1 + 1$, as the ascending sequence of indexes of 0's in B^*, and $Z_1 = (z_i^1)$, $i = 1, \ldots, n + n_0$, as the ascending sequence of indexes of 1's in B^*.

Let S be a sequence of integers. We denote by S^R the sequence formed by the elements of S in backward order. If $S = (s_i)$, for $i = 1, \ldots, t$, and there is an integer $k \leq t$ such that the subsequence consisting of the elements of S with indexes less than or equal to k is ascending, and the subsequence consisting of the elements of S with indexes greater than or equal to k is descending, then we say S is *bitonic*. If all t elements of a sequence S are distinct and belong to $\{1, \ldots, t\}$, then S is a *permutation*. If S is a permutation of size t, and, for all $1 \leq i \leq t$, the equality $i = s_{s_i}$ holds, then we say S is *self-inverting*. In this case, the unordered pair (i, s_i) is called a 2-*cycle* of S, if $i \neq s_i$, and a 1-*cycle* of S, if $i = s_i$. If S_1, S_2 are sequences (respectively, paths in a graph), we denote by $S_1||S_2$ the sequence (respectively, path) formed by the elements of S_1 followed by the elements of S_2.

Back to Chroni and Nikolopoulos's algorithm, we define $P_b = (b_i)$, with $i = 1, \ldots, n^*$, as the bitonic permutation $Z_0||Z_1^R$. Finally, the self-inverting permutation $P_s = (s_i)$ is obtained from P_b as follows: for $i = 1, \ldots, n^*$, element s_{b_i} is assigned value b_{n^*-i+1}, and element $s_{b_{n^*-i+1}}$ is assigned value b_i. In other words, the 2-cycles of P_s correspond to the n unordered pairs of distinct elements of P_b that share the same minimum distance to one of the extremes of P_b, that is, the pairs $(p, q) = (b_i, b_{n^*-i+1})$, for $i = 1, \ldots, n$. Since the central index $i = n + 1$ of P_b is the solution of equation $n^* - i + 1 = i$, element b_{n+1} — and no other — will constitute a 1-cycle in P_s. We refer to such element of P_s as its *fixed element*, and we let f denote it.

The watermark generated by Chroni and Nikolopoulos's encoding algorithm [5] is a directed graph G whose vertex set is $\{0, 1, \ldots, 2n + 2\}$, and whose edge set contains $4n + 3$ edges, to wit: a *path edge* $(u, u - 1)$ for $u = 1, \ldots, 2n + 2$, constituting a Hamiltonian path that will be unique in G,

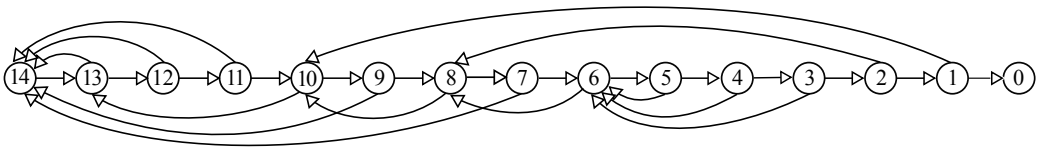

Figure 1: Watermark for identifier $\omega = 43$.

and a *tree edge* from u to $q(u)$, for $u = 1, \ldots, n^*$, where $q(u)$ is defined as the vertex $v > u$ with the greatest index in P_s to the left of u, if such v exists, or $2n + 2$ otherwise. The rationale behind the name *tree edge* is the fact that such edges induce a spanning tree of $G \setminus \{0\}$.

Let us glance at an example. For $\omega = 43$, we have $B = 101011$, $n = 6$, $n_0 = 2$, $n_1 = 4$, $f_0 = 2$, $B^* = 1111110101000$, $n^* = 13$, $Z_0 = (7, 9, 11, 12, 13)$, $Z_1 = (1, 2, 3, 4, 5, 6, 8, 10)$, $P_b = (7, 9, 11, 12, 13, 10, 8, 6, 5, 4, 3, 2, 1)$, $P_s = (7, 9, 11, 12, 13, 10, 1, 8, 2, 6, 3, 4, 5)$ and $f = 8$. The watermark graph associated to ω presents, along with the path edges in the Hamiltonian path $14, 13, \ldots, 0$, the tree edges $(1, 10), (2, 8), (3, 6), (4, 6), (5, 6), (6, 8), (7, 14), (8, 10), (9, 14), (10, 13), (11, 14), (12, 14)$ and $(13, 14)$, as illustrated in Figure 1.

3.1. Structural properties

We now state a number of properties concerning the watermark from Chroni and Nikolopoulos and the special permutations they are associated to. These properties, whose proofs are given in Section 7, set the basis for the characterization of the class of canonical reducible permutation graphs, which is given in Section 4.

For all properties stated below, let G be the watermark graph associated to an identifier ω of size n, and let P_b and P_s be, respectively, the bitonic and the self-inverting permutations dealt with during the construction of G.

Property 1. *For $1 \leq i \leq n$, the element b_{n+i+1} in P_b is equal to $n - i + 1$, that is, the n rightmost elements in P_b, from right to left, are $1, \ldots, n$.*

Property 2. *The elements with indexes $1, \ldots, n$ in P_s are all greater than n.*

Property 3. *The fixed element f satisfies $f = n + f_0$, unless the identifier ω is equal to $2^k - 1$ for some integer k, whereupon $f = n^* = 2n + 1$.*

Property 4. *In self-inverting permutation P_s, elements indexed $1, \ldots, f - n - 1$ are respectively equal to $n + 1, n + 2, \ldots, f - 1$, and elements indexed $n + 1, n + 2, \ldots, f - 1$ are respectively equal to $1, \ldots, f - n - 1$.*

Property 5. *The first element in P_s is $s_1 = n + 1$, and the central element in P_s is $s_{n+1} = 1$.*

Property 6. *If $f \neq n^*$, then the index of element n^* in P_s is equal to $n_1 + 1$, and vice-versa. If $f = n^*$, then the index of element n^* in P_s is also n^*.*

Property 7. *The subsequence of P_s consisting of elements indexed $1, \ldots, n + 1$ is bitonic.*

Property 8. *For $u \leq 2n$, $(u, 2n + 2)$ is a tree edge of watermark G if, and only if, $u - n$ is the index of a digit 1 in the binary representation B of the identifier ω represented by G.*

Property 9. *If (u, k) is a tree edge of watermark G, with $k \neq 2n + 2$, then*

(i) element k precedes u in P_s; and

(ii) if v is located somewhere between k and u in P_s, then $v < u$.

4. Canonical reducible permutation graphs

This section is devoted to the characterization of the class of canonical reducible permutation graphs. After describing some terminology and proving some preliminary results, we define the class using purely graph-theoretical predicates. Then, we show it corresponds exactly to the set of watermarks produced by Chroni and Nikolopoulos's encoding algorithm [5]. Finally, we characterize it in a way that allows for efficient algorithms to recover from distortive attacks.

A *reducible flow graph* [15, 16, 21] is a directed graph G with a source $s \in V(G)$, such that, for each cycle C of G, every directed path from s to C reaches C at the same vertex. It is well known that a reducible flow graph has at most one Hamiltonian cycle.

Definition 10. *A* self-labeling reducible flow graph *is a directed graph G such that*

(i) G presents exactly one directed Hamiltonian path H, hence there is a unique labeling function $\sigma : V(G) \to \{0, 1, \ldots, |V(G)| - 1\}$ of the vertices of G such that the order of the labels along H is precisely $|V(G)|, |V(G)| - 1, \ldots, 0$; and,

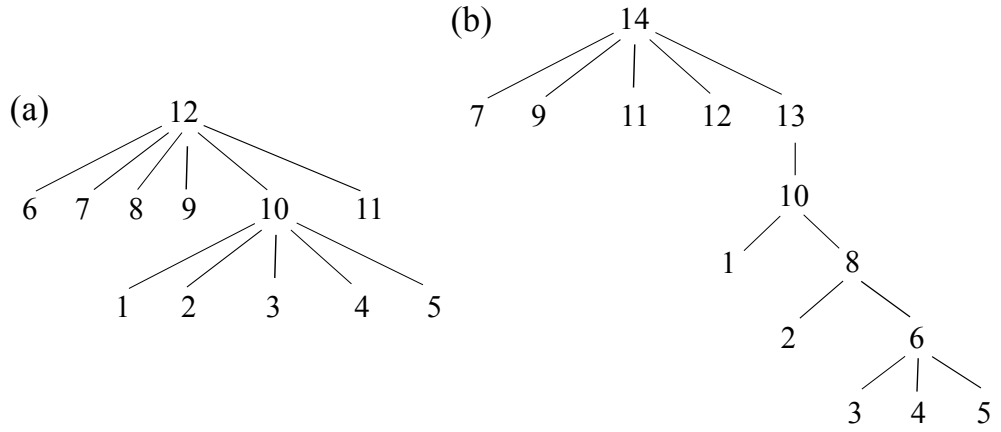

Figure 2: Representative trees of the watermark graphs produced by Chroni and Nikolopoulos's encoding algorithm for identifiers (a) $\omega = 31$ and (b) $\omega = 43$ (the full watermark for $\omega = 43$ is shown in Figure 1). It is easy to check that such graphs are self-labeling reducible flow graphs.

(ii) considering the labeling σ as described, $N_G^+(0) = \emptyset$, $N_G^-(0) = \{1\}$, $N_G^+(|V(G)| - 1) = \{|V(G)| - 2\}$, $|N_G^-(|V(G)| - 1)| \geq 2$, and, for all $v \in V(G) \setminus \{0, |V(G)| - 1\}$, $N_G^+(v) = \{v - 1, w\}$, for some $w > v$.

From now on, without loss of generality, we shall take σ for granted and assume the vertex set of any self-labeling reducible flow graph G *is* the very set $V(G) = \{0, 1, \ldots, |V(G)| - 1\}$. By doing so, we may simply compare two vertices, e.g. $v > u$ (or v greater than u, in full writing), whereas we would otherwise need to compare their images under σ, e.g. $\sigma(v) > \sigma(u)$.

Definition 11. *The* representative tree T *of a self-labeling reducible flow graph G with Hamiltonian path H has vertex set $V(T) = V(G) \setminus \{0\}$ and edge set $E(T) = E(G) \setminus E(H)$, where all edges are deprived of their orientation.*

A representative tree T is always regarded as a *rooted* tree whose root is $|V(G)| - 1$, Moreover, it is regarded as an *ordered* tree, that is, for each $v \in V(T)$, the children of v are always considered according to an ascending order of their labels. For $v \in T$, we denote by $N_T^*(v)$ the set of descendants of v in T. Figure 2 depicts two representative trees.

Observation 12. *The representative tree T of a self-labeling reducible flow graph G satisfies the max-heap property, that is, if vertex u is a child of vertex v in T, then $v > u$.*

Proof: Direct from the way T is rooted and from property (ii) in the definition of self-labeling reducible flow graphs, whereby the in-neighbors of w in $G \setminus E(H)$ comprise only vertices $v < w$. We convey the idea that a representative tree T satisfies the max-heap property by saying that T is a *descending*, ordered, rooted tree. □

Definition 13. *Let $S = (s_i), i = 1, \ldots, 2n+1$, be a self-inverting permutation. We say S is* canonical *if:*

(i) there is exactly one 1-cycle in S;

(ii) each 2-cycle (s_i, s_j) of S satisfies $1 \leq i \leq n$, for $s_i > s_j$;

(iii) $s_1, \ldots, s_{n+1}$ is a bitonic subsequence of S starting at $s_1 = n+1$ and ending at $s_{n+1} = 1$.

Lemma 14. *In any canonical self-inverting permutation, the fixed element f satisfies $f \in [n+2, 2n+1]$.*

Proof: By property (ii) of canonical self-inverting permutations, each 2-cycle of S must contain at least one element whose index i satisfies $1 \leq i \leq n$. From property (i), and given the size of S, it follows that the number of 2-cycles in S is n, hence, by the pigeonhole principle, each and every 2-cycle in S contains *exactly* one such element s_i with $1 \leq i \leq n$. But this means the other element in each 2-cycle, namely s_j, satisfies $s_j \in [n+1, 2n+1]$. Since there are $n+1$ values in that range and only n such elements s_j, there must be exactly one element $s_k \in [n+1, 2n+1]$ which is not part of a 2-cycle, and therefore $s_k = f$. Now, by property (iii), $n+1 = s_1$, hence $f \neq n+1$, and the lemma follows. □

Let T be a representative tree. The *preorder traversal* P of T is a sequence of its vertices that is recursively defined as follows. If T is empty, P is also empty. Otherwise, P starts at the root r of T, followed by the preorder traversal of the subtree whose root is the smallest child of r, followed by the preorder traversal of the subtree whose root is the second smallest child of r, and so on. The last (rightmost) element of P is also referred to as the rightmost element of T.

Lemma 15. *The preorder traversal of a representative tree T is unique. Conversely, a representative tree T is uniquely determined by its preorder traversal.*

Proof: We use induction on $|V(T)|$. If $|V(T)| \leq 1$, the lemma holds trivially. Let $|V(T)| > 1$, and let v_k be the uniquely defined leaf of T for which the path $v_1, \ldots, v_k$ from the root v_1 of T to v_k has the property that each v_i, for $1 < i \leq k$, is the greatest vertex among the children of $v - 1$. By the induction hypothesis, the preorder traversal P' of $T \setminus \{v_k\}$ is unique. Because v_k is necessarily the rightmost vertex of T, the preorder traversal P of T is uniquely determined as $P'||v_k$.

Conversely, let P be a preorder traversal of some representative tree T. If $|P| \leq 1$, there is nothing to prove. Otherwise, suppose the lemma holds for preorder traversals of size $\leq k$, and consider $|P| = k$. Let v_k be the rightmost element of P. Clearly, v_k must be a leaf of T, and also the rightmost (i.e., greatest) vertex among the children of its parent. Now define $P' = P - \{v_k\}$. By the induction hypothesis, there is a unique tree T' whose preorder traversal is P'. Let v_{k-1} be the rightmost element of P'. We obtain T from T', by making v_k the rightmost child of the smallest ancestor v_j of v_{k-1} satisfying $v_j > v_k$, so P is clearly the preorder traversal of T. Since no other parent for v_k would be possible without breaking the ascending order of siblings in a representative tree, T is uniquely defined by P. □

The first element of the preorder traversal P of a tree T is always its root. If we remove the first element of P, the remaining sequence is said to be the *root-free* preorder traversal of T.

We can now define the class of canonical reducible permutation graphs.

Definition 16. *A canonical reducible permutation graph G is a self-labeling reducible flow graph on $2n + 3$ vertices, for some integer ≥ 1, such that the root-free preorder traversal of the representative tree of G is a canonical self-inverting permutation.*

Lemma 17. *If G is a watermark instance produced by Chroni and Nikolopoulos's encoding algorithm [5], then G is a canonical reducible permutation graph.*

Proof: Recall, from Section 3, that the watermark graph G associated to identifier ω, whose binary representation B has size n, is constructed with

vertex set $V(G) = \{0, \ldots, 2n+2\}$ and an edge set $E(G)$ which can be partitioned into path edges and tree edges in such a way that all conditions in the definition of self-labeling reducible flow graphs are satisfied, as can be easily checked. Now, by Property 9 of Chroni and Nikolopoulos's watermarks (see Section 3.1), the tree edges of G constitute a representative tree T of G whose root-free preorder traversal is precisely the self-inverting permutation P_s determined by the encoding algorithm from [5] as a function of B. Consequently, what is left to prove is that P_s is canonical. The first condition to P_s being canonical is asserted by Property 3 in Section 3.1 (the fixed element f corresponds to the unique 1-cycle in P_s); the second condition is given by Property 2; and, finally, Properties 5 and 7 fulfill the third condition, therefore P_s is canonical. □

Lemma 18. *If G is a canonical reducible permutation graph, then G is the watermark produced by Chroni and Nikolopoulos's encoding algorithm [5] for some integer identifier ω.*

Proof: Let G be a canonical reducible permutation graph, and T its representative tree. By Lemma 15, T is uniquely defined by its preorder traversal P. We show that P corresponds to the self-inverting permutation P_s generated by the encoding algorithm of [5] (please refer to Section 3 for details) when computing the watermark for some integer identifier ω. By definition, $P = (s_i), i = 1, \ldots, 2n+1$, is a canonical self-inverting permutation presenting a single 1-cycle f and a number n of 2-cycles (p, q). Those 2-cycles (p, q) define exactly one bitonic permutation $P_b = (b_j), j = 1, \ldots, 2n+1$ satisfying Property 1 of Chroni and Nikolopoulos's watermarks with

(i) $b_{n+1} = f$, and,

(ii) for all $j \in \{1, \ldots, n\}$, $b_j = p$ if and only if $b_{2n+1-j} = q$.

Such bitonic permutation P_b can be regarded as $Z_0||Z_1^R$ by assigning to Z_0 the prefix of P_b comprising its maximal ascending subsequence, and now the indexes of 0's and 1's in the extended binary B^* are totally determined. We proceed by extracting the binary B that is the one's complement of the subsequence of B^* with digits from the $(n+1)$th to the $(2n)$th position in B^*. Regarding B as the binary representation of a positive integer ω, the image of such ω under the encoding function of Chroni and Nikolopoulos is isomorphic to G. □

We proceed to the last definitions before we can give an appropriate, algorithmic-flavored characterization of canonical reducible permutation graphs. Let T be the representative tree of some canonical reducible permutation graph G, and P a canonical self-inverting permutation corresponding to the root-free preorder traversal of T. We refer to the fixed element f of P also as the fixed element (or vertex) of both G and T. Similarly, the 2-cyclic elements of P correspond to *cyclic* elements (or vertices) of both G and T. A vertex $v \in V(T) \setminus \{2n+2\}$ is considered *large* when $n < v \leq 2n+1$; otherwise, $v \leq n$ and v is dubbed as *small*. Denote by X, Y, respectively, the subsets of large and small vertices in T, so $|X| = n+1$ and $|Y| = n$. By Lemma 14, $f \in X$. We then define $X_c = X \setminus \{f\} = \{x_1, \ldots, x_n\}$ as the set of large cyclic vertices in T.

Definition 19. *A representative tree T is a* Type-1 *tree — see Figure 3(a) — if*

(i) $n+1, n+2, \ldots, 2n+1$ *are children of the root* $2n+2$ *in* T*; and*

(ii) $1, 2, \ldots, n$ *are children of* $2n$*.*

Definition 20. *A representative tree T is a* Type-2 *tree relative to f — see Figure 3(b) — if*

(i) $n+1 = x_1 < x_2 < \ldots < x_\ell = 2n+1$ *are the children of* $2n+2$*, for some* $\ell \in [2, n-1]$*;*

(ii) $x_i > x_{i+1}$ *and* x_i *is the parent of* x_{i+1}*, for all* $i \in [\ell, n-1]$*;*

(iii) $1, 2, \ldots, f-n-1$ *are children of* x_n*;*

(iv) $x_i = n+i$*, for* $1 \leq i \leq f-n-1$*;*

(v) f *is a child of* x_q*, for some* $q \in [\ell, n]$ *satisfying* $x_{q+1} < f$ *whenever* $q < n$*; and*

(vi) $N_T^*(f) = \{f-n, f-n+1, \ldots, n\}$ *and* $y_i \in N_T^*(f)$ *has index* $x_{y_i} - f + 1$ *in the preorder traversal of* $N_T^*[f]$*.*

Lemma 21. *If y_r is the rightmost vertex of a Type-2 representative tree T relative to some $f \neq 2n+1$, then y_r is equal to the number ℓ of children of the root $2n+2$ in T.*

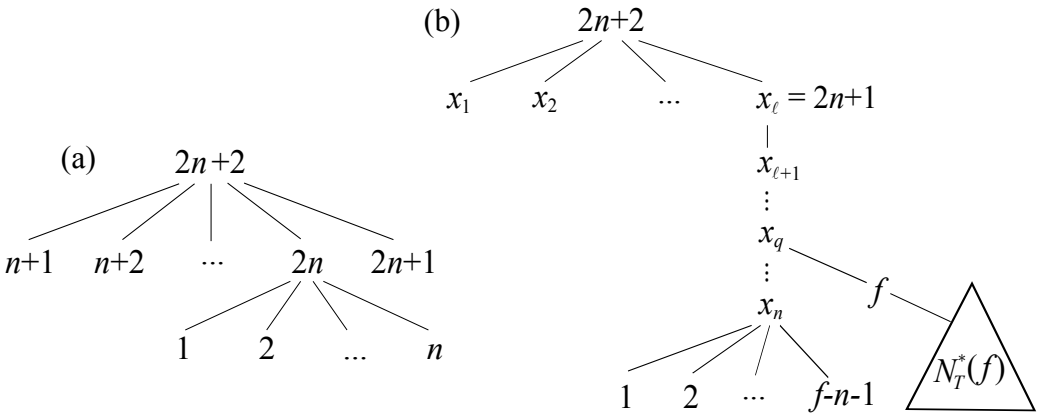

Figure 3: (a) A Type-1 representative tree. (b) A Type-2 representative tree.

Proof: By the definition of a Type-2 representative tree, the only non-leaf child of the root $2n+2$ of T is its rightmost child x_ℓ, therefore each child x_i of $2n+2$, for $1 \leq i \leq \ell$, appears precisely at the ith position in the root-free preorder traversal P of T. Since, by definition, P is self-inverting, and y_r is the last, $(2n+1)$th element of P, it follows that y_r must be equal to the index of $2n+1 = x_\ell$ in P, that is, $y_r = \ell$. □

The following theorem characterizes canonical reducible permutation graphs in terms of the above defined trees. Such characterization is crucial for the remainder of the paper, since its straightforward conditions can be checked in linear time and give rise to the algorithms that will come next.

Theorem 22. *A digraph G is a canonical reducible permutation graph if, and only if, G is a self-labeling reducible flow graph and*

(i) the fixed element of G is $2n+1$ and G has a Type-1 representative tree; or

(ii) the fixed element of G belongs to $[n+2, 2n]$ and G has a Type-2 representative tree.

Proof: Let G be a canonical reducible permutation graph and T its representative tree. By definition, G is a self-labeling reducible flow graph. Let $P = s_1, \ldots, s_{2n+1}$ be the root-free preorder traversal of T, hence a canonical self-inverting permutation, also by definition. This means, among other

things, that P has a unique fixed element f, and that $P' = s_1, \ldots, s_{n+1}$ is a bitonic subsequence of P. Since T is descending, it follows that the prefix A of P' constituting its maximal ascending subsequence must comprise solely vertices that are children of the root $2n+2$, the rightmost of which certainly being $2n+1$.

First, let $f = 2n+1$. Since f constitutes a 1-cycle of P, f must occupy the rightmost, $(2n+1)$th position in P, hence f is a leaf of T. Furthermore, by Property 4 of canonical reducible permutation graphs, it follows that P' consists of elements $n+1, n+2, \ldots, 2n, 1$, hence $A = n+1, n+2, \ldots, 2n$, and these vertices are therefore children of $2n+2$ in T. Now, again by Property 4, elements $1, \ldots, n$ appear, in this order, to the right of A in P. Considering that P is a preorder traversal and a representative tree satisfies the max-heap property, we conclude that vertices $1, \ldots, n$ can only be children of $2n$, hence T is a Type-1 tree, as required.

Next, suppose $f < 2n+1$. By Lemma 14, it follows that $f \in [n+2, 2n]$. We already know that the children of $2n+2$ are the vertices of A. Let D be the subset formed by the remaining vertices of P'. Clearly, the vertices of D must appear in descending order. Since T satisfies the max-heap property and P is a preorder traversal of T, it follows that the largest vertex of D is a child of $2n+1$, and subsequently each vertex in D is the parent in T of the vertex placed to its left along the sequence D. Again, because T satisfies the max-heap property, $f \in [2n+2, 2n]$ must be the child of the smallest vertex $x_q \in D \cup \{2n+1\}$ satisfying $x_q > f$. Let us again examine the ascending subsequence A. We know that the first vertex of A is $n+1$. Suppose the leading k vertices of A are $n+1, n+2, \ldots, n_k$, for some k. Because P is self-inverting, it follows that the vertices $1, 2, \ldots, k$ must be the children of the last (i.e., smallest) vertex of D, and $k = f - n - 1$ by Property 4. It remains solely to describe how the remaining small vertices, namely $f-n, f-n+1, \ldots, n$, are placed in T. Since they appear after f in P, it can only be that this subset comprises exactly the descendants $N_T^*(f)$ of f in T. Each of the vertices $y \in N_T^*(f)$ constitute a 2-cycle with some vertex x belonging to the bitonic subsequence P', hence the index of y in P is exactly x, and all the conditions for a Type-2 tree have thus been verified.

Conversely, let G be a self-labeling reducible flow graph. First, suppose that (i) applies and let T be the corresponding Type-1 representative tree. Then the root-free preorder traversal P of T is

$$n+1, n+2, \ldots, 2n, 1, 2, \ldots, n, 2n+1.$$

Regarding P as a permutation of {1, ..., 2n+1}, we observe that $2n+1$ is the only fixed vertex on it; for $1 \leq i \leq n$, each element $n+i$ of P has index i, while i has index $n+i$, and $n+1, n+2, \ldots, 2n, 1$ form a bitonic subsequence of P. Consequently, P is a canonical self-inverting permutation, and G is a canonical reducible permutation graph.

Finally, suppose (ii) applies. Let T be the corresponding Type-2 representative tree relative to some $f \in [n+2, 2n]$. The root-free preorder traversal P of T consists of

$$x_1, \ldots, x_\ell, x_{\ell+1}, \ldots, x_q, x_{q+1}, \ldots, x_n, 1, 2, \ldots, f-n-1, f, P(N_T^*(f)),$$

where $x_1 = n+1$; $x_\ell = 2n+1$; $x_i = n+i$ for $1 \leq i \leq f-n-1$; $x_1, x_2, \ldots, x_n, 1$ is a bitonic subsequence of P; and $P(N_T^*(f))$ denotes the preorder traversal of the vertices of $N_T^*(f)$, in which each $y_i \in N_T^*(f)$ has index $x_{y_i} - f + 1$. Observe that, for $1 \leq i \leq f-n-1$, $(n+i, i)$ constitutes a 2-cycle in P. Moreover, for $f-n \leq i \leq n$, vertex x_i forms a 2-cycle with an element $y_j \in N_T^*(f)$. All conditions have been met, thus P is a canonical self-inverting permutation and G is a canonical reducible permutation graph. □

Corollary 23. *The recognition of canonical reducible permutation graphs can be achieved in linear time.*

Proof: Direct from Theorem 22 and from the definitions of self-labeling reducible flow graphs, Type-1 and Type-2 representative trees, all of whose conditions can be verified in linear time easily. □

5. A new linear-time decoding algorithm

We can now formulate a new decoding algorithm, whose steps are described in Algorithm 1, and which is rather simpler than the one proposed in [5].

Theorem 24. *Algorithm 1 correctly retrieves the identifier encoded by a canonical reducible permutation graph G in linear time.*

Proof: Let G be a canonical reducible permutation graph on $2n+3$ vertices encoding the identifier ω. The set A, obtained in step 3 of Algorithm 1,

Algorithm 1 *decode*(G)

input: a watermark G with $2n+3$ vertices and $0 \leq k \leq 2$ missing edges
output: the identifier ω encoded by G

1. Let H be the unique Hamiltonian path in G.

2. Label the vertices of G so that H reads $2n+2, 2n+1, \ldots, 0$.

3. Let $A = N_G^-(2n+2)$, with $|A| = \ell$, and let $x_1, \ldots, x_{\ell-1} \in A$ be the ascending sequence of vertices of $A \setminus \{2n+1\}$.

4. Return $\omega = \sum_{i=1}^{\ell-1} 2^{2n-x_i}$.

corresponds to the children of $2n+2$ in the representative tree T of G, that is, the vertices x_i of G, for $i = 1, \ldots, \ell$, which are the tail of some tree edge of G pointing to $2n+2$. From Property 8 of canonical reducible permutation graphs, such vertices x_i which are not $2n+1$ are precisely those satisfying $x_i = n + z_i$, where z_i is the index of a digit 1 in the binary representation B of ω. The summation yielding ω can now be easily checked, since the relative value of a digit 1 placed at position z_i is $2^{n-z_i} = 2^{n-(x_i-n)} = 2^{2n-x_i}$.

The first step can be accomplished in straightforward fashion. Let H be initially the single vertex $v_0 \in V(G)$ whose out-degree is zero. Update H by concatenating to the left of H the unique vertex v_1 which is an in-neighbor of v_0 in G. Now, while $V(H) \neq V(G)$, proceed by concatenating to the left of H the unique vertex v_h, with $h = |H|$, which is an in-neighbor of v_{h-1} in $G - \{v_{h-2}, v_{h-3}, \ldots, v_0\}$. The whole process can be easily achieved in linear $O(|V(G)| + |E(G)|) = O(n)$ time by using adjacency lists for both in-neighbors and out-neighbors of each vertex.

The ascending order, in the third step of the algorithm, can be achieved in linear time by checking in an adjacency matrix whether v is an in-neighbor of $2n+2$, for each $v \in \{n+1, n+2, \ldots, 2n\}$. Note that it is not necessary to actually obtain the representative tree of G. Steps 2 and 4 are computed in linear time trivially. □

6. Polynomial-time decoding with k missing edges

The linear-time recognition of the class of canonical reducible permutation graphs, wrapped up in the form of Corollary 4, allows the construction of a polynomial-time algorithm to recover watermarks which have been deprived of k edges, for any fixed integer k. The proposed algorithm is formally robust [18], since it manages to repair a damaged watermark G' whenever such a thing is possible; otherwise, rather than producing a misled result, it shows that G' does not belong to the family of damaged watermarks that can possibly be recovered. As a certificate for this latter case, it outputs the set of watermarks that may become isomorphic to G' through the removal of exactly k of their edges, thus proving that the intended restore is not at all possible.

Let G be a watermark and G' the graph obtained from G when a certain subset of k edges are removed. The idea is simple. The algorithm attempts the addition to $E(G')$ of each and every k-subset of non-edges of G', one subset at a time. After each attempt, it checks whether a valid watermark (i.e., a canonical reducible permutation graph) was produced. If, after trying all subsets, only one graph was recognized as such, then the decoding was successful. Otherwise, it displays a set containing all watermark candidates.

Since $|V(G')| = 2n+3$ and $|E(G')| = 4n+3-k$, the number of k-subsets of non-edges of G' is

$$\binom{\binom{2n+3}{2} - (4n+3-k)}{k} = \mathcal{O}(n^{2k}).$$

Thus, considering the effort of running the recognition algorithm for each one of these watermark candidates, the algorithm runs in overall $\mathcal{O}(n^{2k}) \cdot \mathcal{O}(n) = \mathcal{O}(n^{2k+1})$ time.

The aforementioned formulation considers that all non-edges of G' could be an edge of the original watermark. However, owing to the particular structure of canonical reducible permutation graphs, relatively few among those non-edges do really stand a chance of belonging to G. More precisely, every vertex v of G has out-degree at most 2, hence v must be the tail endpoint of a most $2 - |N^+_{G'}(v)|$ edges. The multiset M^* of all candidates to

being the tail of a missing edge has therefore

$$\begin{aligned} |M*| &= \sum_{v \in V(G')} \left(2 - |N^+_{G'}(v)|\right) \\ &= 2 \cdot |V(G')| - \sum_{v \in V(G')} |N^+_{G'}(v)| \\ &= 2 \cdot |V(G')| - |E(G')| \\ &= 2 \cdot (2n+3) - (4n+3-k) = k+3 \end{aligned}$$

elements (not necessarily distinct), and therefore the k missing edge tails may be chosen in $\binom{|M^*|}{k} = \mathcal{O}(k^3)$ different ways. For each k-subset of M^*, the algorithm must choose the head corresponding to each tail, which can be done in $\mathcal{O}(n)$ ways per edge, for an overall $\mathcal{O}(k^3 n^k)$ number of k-subsets of non-edges that shall be tentatively added to G'. With the $\mathcal{O}(n)$ running time of the recognition algorithm for each such attempt, we complexity of the whole decoding algorithm is an overall $\mathcal{O}(k^3 n^{k+1})$.

As a matter of fact, it is still possible to eliminate a whole $\mathcal{O}(k^3)$ factor from that asymptotical complexity, if the labels of the vertices are known. To assume that the labels are known is reasonable in many situations, since each vertex corresponds to a block in the CFG of the software, and, by construction, the watermark graph possesses a Hamiltonian path $2n+2, 2n+1, \ldots, 0$ that corresponds, in the CFG, to a chunk of subsequent blocks. If that is the case, then we know the out-degree, in G, of all watermark vertices (the tail and the head of the Hamiltonian path have out-degrees 1 and 0, respectively; all other vertices have out-degree 2) and, consequently, the tails of all missing edges. By running the linear-time recognition algorithm on each possible choice of heads, the decoding algorithm has an overall $\mathcal{O}(n^{k+1})$ time complexity.

7. Proofs of the properties from Section 3

We had postponed the proofs of the properties stated in Section 3 to avoid an overhead of technical pages too early in the paper. We now present the full proofs.

Proof of Property 1. When read from right to left, the n rightmost elements in P_b correspond to the n first elements in Z_1, i.e the n first indexes, in B^*,

where a digit 1 is located. Since B^* starts with a sequence of n contiguous 1's, the property ensues. □

Proof of Property 2. In B^*, digits with indexes $1, 2, \ldots, n$ are all 1, by construction. Since the n rightmost elements in P_b (i.e., elements indexed $n+2 \leq i \leq n^*$ in P_b) correspond to the first n elements in Y, and therefore to the first n indexes of 1's in B^*, those will always be precisely the elements of set $S = \{1, 2, \ldots, n\}$. In other words, if $s \in S$, then s will have index $n^* - s + 1 > n + 1$ in P_b. By the time the elements of P_b are gathered together in pairs with views to defining their placement in P_s, element s will be paired with element q whose index is $n^* - (n^* - s + 1) + 1 = s$. Because $s \leq n$, such q clearly does not belong to S, hence $q > n$. Now, because s will be assigned index q in P_s, the element with index s in P_s will be its pair $q > n$, concluding the proof. □

Proof of Property 3. The bitonic permutation P_b is assembled in such a way that its $(n+1)$th element $f = b_{n+1}$ is either:

(i) the $(n+1)$th element of Z_0, in case B^* has at least $n+1$ digits 0; or

(ii) the $(n+1)$th element of Z_1, otherwise.

By construction, the number of 0's in B^* is one unit greater than the number of 1's in B.

If (i) holds, then B corresponds to an identifier w that is the predecessor of a power of 2, implying all n digits of B are 1's. If that is the case, then the desired property follows immediately, once the $(n+1)$th element of Z_0 will be the index of the $(n+1)$th — i.e., the last — digit 0 in B^*. Such index is, by construction, n^*.

If (ii) holds, then f is the index of the $(n+1)$th digit 1 in B^*. By construction, the n first digits 1 in B^* occupy positions with indexes $1, \ldots, n$, and the $(n+1)$th digit 1 in B^* corresponds to the first digit 1 in the one's complement of B. Since that digit has index f_0 in the one's complement of B, and there are in B^* exactly n digits to the left of the one's complement of B, the property follows. □

Proof of Property 4. From the construction of P_s and Property 1, it follows that the elements that occupy positions with indexes $1, 2, \ldots, n$ in P_s are the first n elements in P_b. It just occurs that the first $n_1 + 1$ numbers in P_b are the elements of Z_0, i.e., the indexes of 0's in B^*. Now, the last digit in B^* — the one indexed n^* — is always a 0. Besides that 0, the other digits 0 in B^* have indexes $z = n + d$, where each d is the index of a digit 1 in B (the original binary representation of the identifier ω). While the first digit in B is always 1, it is also true that:

(i) the $f_0 - 1$ first digits in B constitute a seamless sequence of 1's, in case there is at least one 0 in B; or

(ii) *all* n digits of B are 1's, in which case ω is the predecessor of a power of 2.

Whichever the case, Property 3 allows us to state that there is a sequence of $f - n - 1$ digits 1 in B starting at the first digit of B. Such sequence will show up, in B^*, starting at index $n + 1$, in such a way that the $f - n - 1$ first elements of Z_0 will be $n + 1, n + 2, \ldots, n + (f - n - 1) = f - 1$. Those elements, as we have seen, will be precisely the first numbers in P_b. Because there are no more than n such elements, they will be paired against elements $1, 2, \ldots, f - n - 1 \leq n$ (located from the right end of P_b leftwards) in order to determine their placement in P_s, and the property follows. □

Proof of Property 5. If the identifier ω is not the predecessor of a power of 2, then its binary representation B, whose first digit is always a 1, contains some digit 0. In light of this, Property 3 implies $f \geq n + 2$ for all integers ω, and the first equality now follows from Property 4. The second equality is granted by the self-invertibility of P_s, whereby $s_j = u \iff s_u = j$. □

Proof of Property 6. First, note that $f \neq n^*$ corresponds to the case where the identifier ω is not the predecessor of a power of 2, i.e., $n_1 < n$. Because the sequence Z_0 has exactly $n_1 + 1$ elements, the last of which being the index n^* of the rightmost digit in B^*, element n^* will always be assigned index $n_1 + 1$ in P_b. As we have seen in the proof of Property 4, for $i \leq n$, the ith element in P_b will also be the ith element in P_s, for it will be paired against element i, indexed $n^* - i + 1$ in P_b (due to the starting sequence of n digits 1 in B^*). That being said, element n^*, indexed $n_1 + 1 \leq n$ in P_b, will

have index $n_1 + 1$ in P_s as well. If $f = n^*$, then the definition of f verifies the property trivially. □

Proof of Property 7. We employ again the fact, noted for the first time in the proof of Property 4, that the subsequence consisting of the first n elements in P_s and the subsequence consisting of the first n elements in P_b are one and the same. Since P_b is bitonic, whatever subsequence of P_b is bitonic too, particularly the one containing its first n elements. By Property 5, the central element s_{n+1} of P_s is always equal to 1, therefore the bitonic property of the subsequence consisting of the leftmost elements of P_s will not be broken after its length has grown from n to $n+1$, that is, after element $s_{n+1} = 1$ has been appended to it. □

Proof of Property 8. The first $n_1 + 1$ elements of the bitonic permutation P_b are the elements of Z_0, corresponding to the indexes of 0's in the extended binary B^* (which consists, we recall, of n digits 1, followed by the one's complement of the binary representation B of the identifier ω encoded by G, followed by a single digit 0). Those elements constitute the ascending prefix $A = n + z_1, n + z_2, \ldots, n + z_{n_1}, 2n + 1$, where, for $i \in \{1, \ldots, n_1\}$, z_i is the index of a digit 1 in B. From the proof of Property 4, we know that, for $i \leq n$, the ith element in P_b will also be the ith element in the self-inverting permutation P_s. Since $n_1 \leq n$, we have that the n_1 first elements of P_s are precisely the n_1 first elements of A, hence the tree edge tailed at each of those elements must point, by construction, to $2n + 2$. It remains to show that no element $u \notin A \cup \{2n + 1\}$ is the tail of a tree edge pointing to $2n + 2$. But this comes easily from the fact that, by Property 6, the $(n_1 + 1)$th element in P_s is $2n + 1$. Since all vertices u with indexes $i > n_1 + 1$ in P_s are certainly smaller than $2n+1$, they can only be the tail of tree edges pointing to vertices $q(u) \leq 2n + 1$, and the proof is complete. □

Proof of Property 9. Both items are trivially verified, since, by construction, every tree edge $(u, k) \in G$ is such that either $k > u$ is the element that is closest to u and to the left of u in P_s, or $k = 2n + 2$. □

8. Final considerations

We characterized the class of canonical reducible permutation graphs devised in [3]. Such graphs allow for the encoding of binary identifiers and can be embedded into control-flow graphs of software through the insertion of dummy code. Our charaterization led to a simpler linear-time decoding algorithm and a polynomial-time algorithm to retrieve the encoded word from a damaged graph whenever it is possible to do so deterministically.

The proposed characterization actually gives rise to an $O(n)$-time algorithm which can *always* retrieve the encoded binary (with $n > 2$ bits) even when up to two edges of the canonical reducible permutation graph are missing. We shall present such algorithm in a sequel of this paper.

References

[1] F.E. Allen, Frances E., Control flow analysis, *SIGPLAN Not.* **5** (1970), 1–19.

[2] Business Software Alliance, *Shadow Market: 2011 BSA Global Software Piracy Study* (2012), available at `http://globalstudy.bsa.org`.

[3] M. Chroni, S.D. Nikolopoulos, Efficient encoding of watermark numbers as reducible permutation graphs, arXiv:1110.1194v1 [cs.DS], 2011.

[4] M. Chroni, S.D. Nikolopoulos, Encoding watermark numbers as cographs using self-inverting permutations, *Proc. 12th Int'l Conference on Computer Systems and Technologies, CompSysTech'11*, ACM ICPS **578** (2011), 142–148 (Best Paper Award).

[5] M. Chroni, S.D. Nikolopoulos, An efficient graph codec system for software watermarking, *Proc. 36th IEEE Conference on Computers, Software and Applications, COMPSAC'12*, IEEE Proceedings (2012), 595–600.

[6] M. Chroni, S.D. Nikolopoulos, Multiple encoding of a watermark number into reducible permutation graphs using cotrees, *Proc. 13th Int'l Conference on Computer Systems and Technologies, CompSysTech'12*, ACM ICPS Proceedings (2012), 118–125.

[7] M. Chroni, S.D. Nikolopoulos, An embedding graph-based model for software watermarking, *Proc. 8th Int'l Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP' 12*, IEEE Proceedings (2012), 261–264.

[8] C. Collberg, C. Thomborson, Software watermarking: models and dynamic embeddings, *Proc. 26th ACM SIGPLAN-SIGACT on Principles of Programming Languages, POPL'99* (1999), 311–324.

[9] C. Collberg, S. Kobourov, E. Carter, C. Thomborson, Error-correcting graphs for software watermarking, *Proc. 29th Workshop on Graph-Theoretic Concepts in Computer Science, WG'03*, LNCS **2880** (2003), 156–167.

[10] C. Collberg, C. Thomborson, G. Townsend, Dynamic graph-based software fingerprinting, *ACM Transactions on Programming Languages and Systems* **29** (2007), 1–67.

[11] C. Collberg, A. Huntwork, E. Carter, G. Townsend, M. Stepp, More on graph theoretic software watermarks: implementation, analysis and attacks, *Information and Software Technology* **51** (2009), 56–67.

[12] C. Collberg, *SandMark: A Tool for the Study of Software Protection Algorithms*, `http://sandmark.cs.arizona.edu/index.html`, last visited 25 Nov 2013.

[13] R.L. Davidson, N. Myhrvold, Method and system for generating and auditing a signature for a computer program, US Patent 5.559.884, Microsoft Corporation (1996).

[14] J. Hamilton, S. Danicic, A Survey of Static Software Watermarking, *Proc. World Congress on Internet Security, WorldCIS'11* (2011) 100–107.

[15] M.S. Hecht, J.D. Ullman, Flow graph reducibility, *SIAM J. Computing* **1** (1972), 188–202.

[16] M.S. Hecht, J.D. Ullman, Characterizations of reducible flow graphs, *Journal of the ACM* **21** (1974), 367–375.

[17] F. Nielson, H.R. Nielson, C. Hankin, *Principles of Program Analysis* (2004), Springer-Verlag.

[18] V. Raghavan, J. Spinrad, Robust Algorithms for Restricted Domains, *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'01* (2001), 460–467.

[19] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, K. Villela, Software diversity: state of the art and perspectives, *International Journal on Software Tools for Technology Transfer* **14** (2012), 477–495.

[20] Y. Su, J. Liu, D. Li, Hiding Signatures in Variable Names, *Communications in Computer and Information Science* **320** (2013), 333–340.

[21] R.E. Tarjan, Testing flow graph reducibiliy, *Journal of Computer and System Sciences* **9** (1974), 355–365.

[22] R. Venkatesan, V. Vazirani, S. Sinha, A graph theoretic approach to software watermarking, *Proc. 4th International Information Hiding Workshop* (2001), 157–168.

[23] R. Venkatesan, V. Vazirani, Technique for producing through watermarking highly tamper-resistant executable code and resulting watermarked code so formed (2006), Microsoft Corporation, US Patent: 7051208.

[24] B.A. Wichmann, A.A. Canning, D.L. Clutterbuck, L.A. Winsborrow, N.J. Ward, D.W.R. Marsh, Industrial perspective on static analysis, *Software Engineering Journal* **10** (1995), 69–75.

[25] J. Zhu, Y. Liu, K. Yin, A novel dynamic graph software watermark scheme, *Proc. 1st Int'l Workshop on Education Technology and Computer Science* **3** (2009), 775–780.

[26] W. Zhu, C. Thomborson, F-Y. Wang, A survey of software watermarking, *Proc. IEEE Int'l Conference on Intelligence and Security Informatics, ISI'05* (2005), 454–458.