Recognition in Software Watermarking

William Zhu*
the Department of Computer Sciences,
the University of Auckland,
Auckland, New Zealand
fzhu009@ec.auckland.ac.nz

Clark Thomborson
the Department of Computer Sciences,
the University of Auckland,
Auckland, New Zealand
cthombor@cs.auckland.ac.nz

ABSTRACT

The piracy of software has long been a concern for owners and developers of software. In order to prevent software from piracy and unauthorized modification, many techniques to protect software have been developed. Software watermarking is such a technique for protecting software by embedding secret information into the software to identify its copyright owner. As a relatively new scientific area, the key concepts in software watermarking are informal; some are even confusing. There is a need to formalize these fundamental terms to facilitate the research in this field. In this paper, we formally define some concepts for software watermark recognition and use them to develop some software watermarking recognition algorithms for the QP software watermarking algorithm. We also design a prototype model of software watermark embedding and recognition system based on the concepts and algorithms established in this paper.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.0 [**Software Engineering**]: General—*Protection mechanisms*; K.4.1 [**Computers and Society**]: Public Policy Issues—*Intellectual property rights*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Intellectual property; Security*; K.5.1 [**Legal Aspects Of Computing**]: Hardware/ Software Protection—*Copyrights; Proprietary rights*

General Terms

Security

Keywords

Software Watermark, Embedder, Extraction, Recognition, Graph, Interference Graph

*Corresponding author. Phone: +64-9-3737-599 ext. 82289; fax: +64-9-3737-453;

MCPS'06, October 28, 2006, Santa Barbara, California, USA.

1. INTRODUCTION

As the software industries develop rapidly, the protection of intellectual property of software from piracy and unauthorized modification becomes more and more important to computer business and academia. Software watermarking is an approach to embed a message into software to claim the ownership of it [5, 6, 9, 46]. It is an effective mechanism to protect the intellectual property of the software developers.

Since software watermarking is a relatively new research field, some of the core concepts are informal, even confusing. The formalization of them will benefit researchers in this area and facilitate the progress in it. In this paper, we focus on one of the important concepts in software watermarking – recognition. Through the discussion of these terms, we can see more clearly the problems in a good software watermarking system.

This paper is organized as follows. Section 2 is a brief review of the concepts in software watermarking. Section 3 gives the concepts of embedding and extracting and some examples. These form the basis for the definition of the recognition system and for the examples in the following sections. In Section 4, we define recognition, partial recognition, blind recognition, and blind partial recognition of software watermarks. In this section, we also develop several recognition and partial recognition algorithms for the QP algorithm [21, 30, 31, 39]. Section 5 has a description of a model of a software watermark embedding and recognition system based on the concepts and algorithms developed in this paper. Section 6 concludes our paper.

2. OVERVIEW

The basic definitions of software watermarking concepts appeared in the early papers by Collberg et al. [5, 9]. They detailed concepts such as types of attacks on software watermarking systems, static software watermarks and dynamic software watermarks, and evaluation criteria for a software watermarking system. In these papers, dynamic watermarking techniques are divided into three classes: Easter egg watermarks, data structure watermarks, and execution trace watermarks. Attacks are classified into three categories: subtractive attack, distortive attack, and additive attack. Evaluation criteria are stealth, resilience, and data rate. They also defined the extraction and recognition of software watermarks, but these definitions are not very formal and detailed.

Nagra et al. [24, 25, 26] and Thomborson et al. [36] gave definitions of four types of software watermarks from a functional view: authorship mark, fingerprinting mark, valida-

tion mark, and licensing mark. The authors also discussed several other concepts such as visible and invisible watermarks, robust and fragile watermarks, and tamperproofing.

Embedding and extraction are defined more formally by Collberg et al. in their paper [4] as the encoding function and exposition function, respectively. In that paper, the authors considered multiple watermarks in software. In our papers we regard them as parts of a whole watermark. The paper [4] addressed some new ideas on attacks on software watermarking and presented new evaluation metrics of software watermarking system. Zhu and Thomborson formally defined embedding and extraction in [46]. They also addressed several concepts involved in embedding and extraction of software watermarking.

Pastuszak et al. [28] discussed the functional watermarking system and its components and the security evaluation of such a system. A recent survey of software watermarking is in [42]. Concepts and techniques of software watermarking also abound in [5, 10, 11, 15, 16, 20, 21, 22, 23, 27, 29, 35, 38].

A similar but more active research area is digital watermarking [13]. Moulin et al. have already done an elegant work to formalize basic concepts in digital watermarking from the point of information view [18, 17, 19], but their consideration is mainly on digital setting such as audio and video. They regard digital watermarking as a communication problem. In software watermarking, semantics is an important consideration.

Hopper et al. also defined similar concepts in steganography [12], but their definitions are based on a complexity-theoretic point of view. They did not consider the special situation in software watermarking. Similar concepts also appeared in [3, 37] for natural language watermarking and in [1, 2, 32, 33, 34] for database watermarking.

The main contributions of our paper are as follows: Firstly, we give the definitions of the positive-partial recognition, negative-partial recognition, and recognition corresponding to an embedding algorithm. Secondly, we prove the existence of one and only one complete recognizer corresponding to an embedding algorithm. Then, this paper has definitions of the strength of positive-partial recognitions and negative-partial recognitions corresponding to an embedding algorithm. In addition, we show the existence of the weakest and the strongest positive-partial recognitions and negative-partial recognitions corresponding to an embedding algorithm. Furthermore, our paper defines the blind and informed positive-partial recognition, negative-partial recognition, and recognition corresponding to an embedding algorithm. Lastly, we use the above definitions to develop partial recognition algorithms and recognition algorithms for software watermarking and to design a model for a software watermarking embedding and recognition system.

3. EMBEDDING AND EXTRACTING

A software watermarking system must do two basic things. Firstly, it must embed a watermark into a software object. Secondly, it must either extract all bits of the watermark inserted by itself, or recognize whether or not there exists a watermark embedded by itself. In this section, we introduce some concepts of embedding a watermark into and extracting a watermark from a software program, an issue which was already addressed in our previous paper [46]. The extraction problem is not the focus of this paper although we need its concepts for our discussion of the recognition problem, the main topic on which we will concentrate in section 4 of this paper.

3.1 Embedding

Informally speaking, to embed a software watermark into a program is to insert a secret message into this code. We formally define this concept as follows.

DEFINITION 1. *(Watermark) A watermark is a message of bits of 0 and 1 with a finite length ≥ 0. We denote the set of all watermarks as $\mathbf{W}$.*

DEFINITION 2. *(Embedding) Let $\mathbf{P}$ denote the set of programs and $\mathbf{W}$ the set of watermarks. We call a function $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$ a watermark embedding algorithm, or, for the sake of simplicity, an embedding algorithm, or even more simply, an embedder.*

If $P' = A(P, W)$ for some $P \in \mathbf{P}$ and some $W \in \mathbf{W}$, then the P' is called a watermarked program with respect to the embedder A, or, when there is no confusion, simply a watermarked program. We also call the program $P \in \mathbf{P}$ the original program corresponding to the watermarked program P'.

In order to illustrate the concepts defined in this paper, we take a concrete software watermarking algorithm, the QP algorithm, as an example.

3.2 The QP Watermarking Algorithm

Qu and Potkonjak proposed a watermarking algorithm for watermarking solutions to Graph Coloring (GC) problems [30, 31], which is called the QP algorithm in [21, 39, 40]. It requires the vertices of the graph to be indexed, that is, each vertex must be labeled with a unique integer in the range 1 to $|V(G)|$. The QP algorithm relies heavily on the ordering of node indices. The following are some concepts used in the QP algorithm.

DEFINITION 3. *The Cyclic mod n ordering [30, 31]: We use "$<_i$" to denote the cyclic mod n ordering relation for a fixed i, such that $i <_i (i+1) <_i \ldots <_i n <_i 1 <_i \cdots <_i i-1$. Where there is no confusion over the value of i, we omit the subscript in $<_i$.*

DEFINITION 4. *The two nearest vertices that are not connected to a vertex v_i [30, 31]: For a vertex v_i of a graph G with $|V| = n$, we say $v_{i_1} \in V$ and $v_{i_2} \in V$ are the two nearest vertices that are not connected to a vertex v_i if $i <_i i_1 <_i i_2$; $(v_i, v_{i_1}) \notin E$; $(v_i, v_{i_2}) \notin E$; $\forall j : i <_i j <_i i_1$, $(v_i, v_j) \in E$; and $\forall j : i_1 <_i j <_i i_2$, $(v_i, v_j) \in E$.*

In this paper, if the above two vertices exist for a vertice v_i, we also say the vertice v_i has two candidate vertices, v_{i_1} and v_{i_2}.

The essence of the QP algorithm is to add an extra edge between every vertex v_i and one of its two candidate vertices. The watermark bits to be embedded determine the choice between these two nearest unconnected vertices. It is important to notice that this concept is a dynamic one, since the two candidate vertices of v_i may change whenever an edge is added to the neighborhood of v_i.

The original QP algorithm in Fig. 1 was proposed by Qu and Potkonjak [30, 31]. It inserts a watermark into a solution to a GC problem.

Input: An unwatermarked graph G with $n = |V|$ and
An unbounded series of message bits: $W = w_1 w_2 \dots w_m$
Output: A watermarked graph G'.
Algorithm:
$G' := G$;
$j := 1$;
if m > n **then** // not all bits of W can be inserted in G
return G
for each i from 1 to n **do**
if j > m **then** // all bits of W already inserted in G
return G'
if find the nearest two vertices v_{i_1}, v_{i_2}
not connected to v_i in G **then**
if $w_j = 0$ **then**
connect v_i to v_{i_1} in G'
else
connect v_i to v_{i_2} in G'
j++
if m $\geq$ j **then** // not all bits of W inserted in G
return G
return G'

Figure 1: A clarified version of the QP algorithm

3.3 Extraction

After a watermark is inserted in a cover message using an embedder, an important consideration is the potential for an algorithm to extract this watermark. The following definition specifies all potential watermarks an embedder can insert into a program. This set excludes messages which do not change a cover program.

DEFINITION 5. *(Set of candidate watermarks) A $W \in \mathbf{W}$ is called a candidate watermark with respect to a program P and an embedder A if $A(P, W) \neq P$.*

All candidate watermarks constitute the set of candidate watermarks of the program P and the embedder A. This set is denoted as candidate(P, A).

The set of candidate watermarks of a program P and an embedder are all the watermarks that can actually be inserted into P by A. Embedding other watermarks into P will not change the original program. This concept is important for defining extracting.

EXAMPLE 1. *(Set of candidate watermarks) Let A be the QP algorithm in Fig. 1 and P be a program with the interference graph having 4 vertices v_1, v_2, v_3, v_4 and two edges $(v_1, v_3), (v_2, v_4)$. Then, the set of candidate watermarks of A and P is {0, 1, 00, 01, 10, 11, 010, 011, 110, 111}. The interference graphs of original program and the watermarked programs are in Fig. 2. We can see from Fig. 2 that the interference graph for the watermarked program after watermark 010 is inserted is the same as that after watermark 111 is inserted. For this reason, a watermark embedded by the QP algorithm can not be extracted reliably.*

After we embed a watermark into a program, we may ask how to extract this watermark. We attempt to extract only candidate watermarks.

DEFINITION 6. *(Extractor) Let A be an embedder; a function $X : \mathbf{P} \times \mathbf{P} \rightarrow \mathbf{W}$ is called an extractor corresponding to the embedder A if X has the following property: $\forall P, P' \in \mathbf{P}$, $w \in \mathbf{W}$, if $W \in$ candidate(P, A) and $P' = A(P, W)$, then $X(P', P) = W$. Otherwise, $X(P', P) = \epsilon$.*

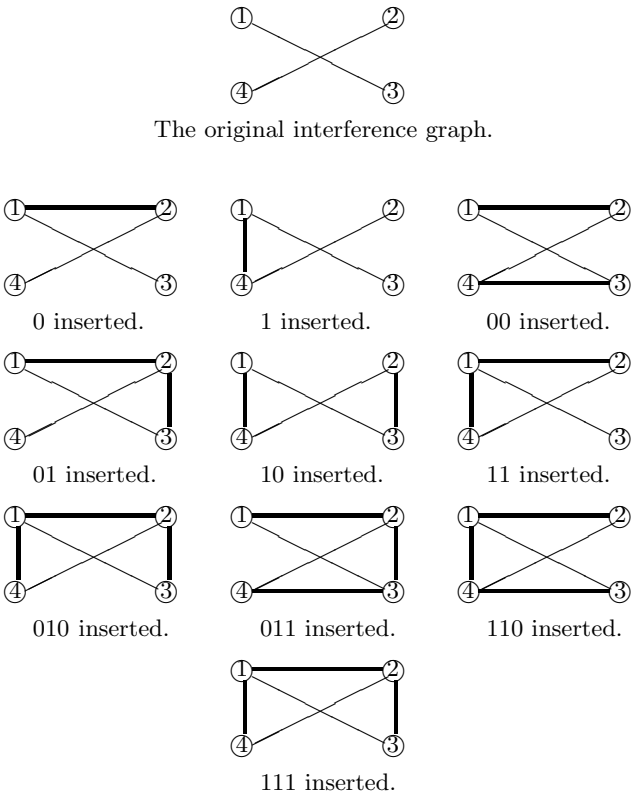


Figure 2: The interference graphs of the original and the watermarked programs

DEFINITION 7. *(Extractable) We say watermarks embedded by an embedder A are* extractable *if there exists an extractor X corresponding to the embedder A. We also say X demonstrates that the embedder A is extractable, or more simply, that A is extractable.*

EXAMPLE 2. *(The QP embedding algorithm in Fig. 1 is not extractable) From [46], we see that the QP algorithm in Fig. 1 is not extractable.*

4. RECOGNITION

In some situations, we may not have to extract all bits of a watermark inserted in the software – we just want to see whether software has a watermark embedded by an embedder. In some other situations, the software watermark embedder may be in essence unextractable, or, even when the embedder is extractable, the extractor may be inefficient for a specific application. In this section, we discuss the problem of judging the existence of a watermark embedded in a software. In the following subsection we treat the case of informed recognition, where the recognition function is supplied with two programs. The first argument of the recognition function is the program to be tested for the existence of a watermark, and the second argument is the putative unwatermarked original program.

4.1 Recognizers and Partial Recognizers

For a positive-partial recognition, if a program really has a watermark, the recognition will detect it. But, such a recognition might say a program has a watermark while this program actually has no watermarks.

DEFINITION 8. *(Positive-partial recognition) For an embedder* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, if a function* $R : \mathbf{P} \times \mathbf{P} \rightarrow \{TRUE, FALSE\}$*, satisfies that* $\forall P, P' \in \mathbf{P}$*, if there is a* $W \in candidate(A, P)$ *such that* $P' = A(P, W)$ *then* $R(P', P) = TRUE$*, we call* R *a* positive-partial *recognition function for the embedder* A*, or simply a* positive-partial *recognizer.*

For a negative-partial recognizer, if it says a program has a watermark, this program really has a watermark. But, such a recognizer might say a program has no watermarks while this program actually has a watermark.

DEFINITION 9. *(Negative-partial recognition) For an embedder* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, if a function* $R : \mathbf{P} \times \mathbf{P} \rightarrow \{TRUE, FALSE\}$ *satisfies* $\forall P, P' \in \mathbf{P}$*,* $R(P', P) = TRUE \implies P' = A(P, W)$ *for some* $W \in \mathbf{W}$*, we call* R *a* negative-partial *recognition function for the embedder* A*, or simply a* negative-partial *recognizer.*

For a complete accurate recognizer, if a program has a watermark, the recognizer will say that this program has a watermark; if a program has no watermarks, the recognizer will say that this program has no watermarks.

DEFINITION 10. *(Recognizer) For an embedder* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, if a function* $R : \mathbf{P} \times \mathbf{P} \rightarrow \{TRUE, FALSE\}$ *satisfies* $\forall P, P' \in \mathbf{P}$*,* $R(P', P) = TRUE \iff P' = A(P, W)$ *for some* $W \in candidate(A, P)$*, we call* R *a* complete recognition function *for the embedder* A*, or simply a recognizer.*

We say that A *is recognizable if there exists a recognizer for* A*.*

If A is extractable, it is also recognizable. Let X be the extraction algorithm with respect to A. We can construct a recognizer $R(P', P)$ by testing whether an extracted watermark $X(P', P)$ is equal to some $W \in candidate(A, P)$.

THEOREM 1. *For every embedder* A*, there exists one and only one recognizer corresponding to* A*. We denote the unique recognizer corresponding to* A *as* $Reg(A)$*.*

Proof. $\forall P, P' \in \mathbf{P}$, define $R(P', P)$ as follows.

$R(P', P)$ = TRUE, if there is some $W \in candidate(A, P)$ such that $P' = A(P, W)$

$R(P', P)$ = FALSE, otherwise

It is easy to see R is a recognizer corresponding to A. □

From Theorem 1 and Example 2, not all embedders are extractable, but every embedder is recognizable.

Theorem 1 shows there is one and only abstract recognizer, but there might be several concrete recognition algorithms to realize such a recognizer.

PROPERTY 1. *For every embedder* A*,* $Reg(A)$ *is both the positive-partial and the negative-partial recognizers corresponding to* A*.*

We present four examples to illustrate the above concepts.

EXAMPLE 3. *(Trivial partial recognizers) The partial recognition concepts are very flexible. The following are some trivial partial recognitions. For an embedder* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, define a function* $S : \mathbf{P} \times \mathbf{P} \rightarrow \{TRUE, FALSE\}$*, as* $P', P \in \mathbf{P}$*,* $S(P', P) = TRUE$*. This is a positive-partial recognition corresponding to* A*. We call such a function a trivial positive-partial recognizer corresponding to* A *and denote it as* $TrivPP(A)$*.*

For an embedder $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, define a function* $S : \mathbf{P} \times \mathbf{P} \rightarrow \{TRUE, FALSE\}$ *as* $P', P \in \mathbf{P}$*,* $S(P', P) = FALSE$*. This is a negative-partial recognizer corresponding to* A*. We call such a function a trivial negative-partial recognizer and denote it as* $TrivNP(A)$*.*

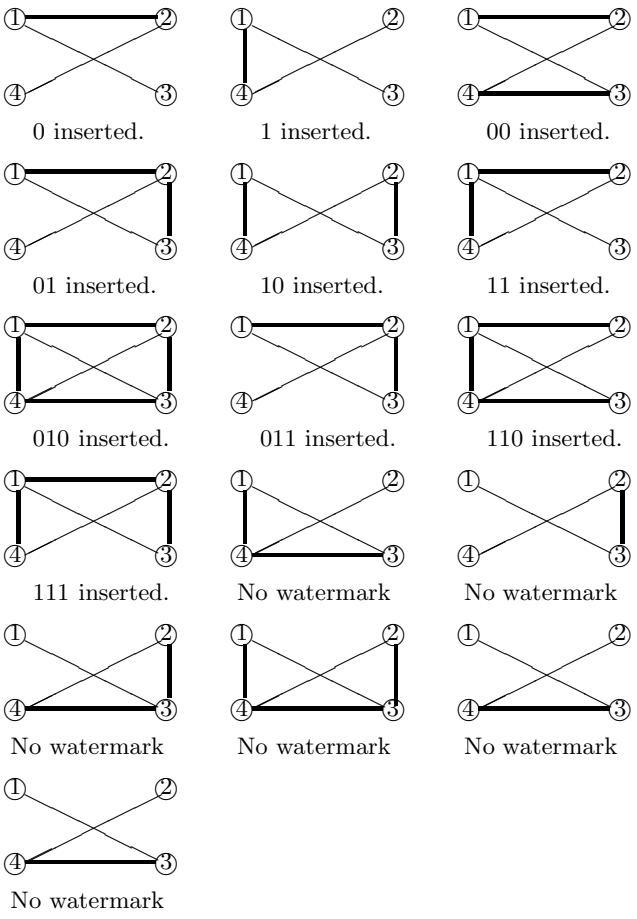


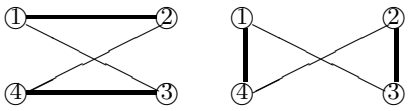
Figure 3: The interference graphs for the positive-partial recognizer of Figure 4.

EXAMPLE 4. *(A positive-partial recognizer for the QP algorithm) A positive-partial recognizer for the QP algorithm is in Fig. 4. For the program with its interference graph as in Example 1, the programs recognized by this recognizer are the ones with interference graphs as in Fig. 3.*

EXAMPLE 5. *(A negative-partial recognizer for the QP algorithm) A negative-partial recognizer for the QP algorithm is in Fig. 6. For the program with its interference graph as in Example 1, the programs recognized by this recognizer are the ones with interference graphs as in Fig. 5.*

Input: an unwatermarked graph $G(V, E)$ with $n = |V|$
a watermarked graph G'
Output: is a message W embedded in G' ?
Algorithm:
if G is not a subgraph of G' **then**
  return FALSE
j:=0
for each i from 1 to n **do**
  if find the nearest two vertices v_{i_1}, v_{i_2}
    not connected to v_i in G **then**
    j++
    if $(v_i, v_{i_2}) \in G'$ **then**
      connect v_i to v_{i_2} in G
    else if $(v_i, v_{i_1}) \in G'$ **then**
      connect v_i to v_{i_1} in G
    else // all bits extracted
      exit
if j=0 **then**
  return FALSE
return TRUE

Figure 4: A positive-partial recognizer for the QP algorithm



00 inserted. 10 inserted.

Figure 5: The interference graphs for the negative-partial recognizer of Figure 6.

Input: an unwatermarked graph $G(V, E)$ with $n = |V|$
a watermarked graph G'
Output: is a message W embedded in G' ?
Algorithm:
if G is not a subgraph of G' **then**
  return FALSE
j:=0
for each i from 1 to n **do**
  if find the nearest two vertices v_{i_1}, v_{i_2} not connected
    to v_i in G **then**
    j++
    if $(v_i, v_{i_1}) \in G'$ **then**
      connect v_i to v_{i_1} in G
    else if $(v_i, v_{i_2}) \in G'$ **then**
      connect v_i to v_{i_2} in G
    else
      return FALSE
if j=0 **then**
  return FALSE
return TRUE

Figure 6: A negative-partial recognizer for the QP algorithm

Input: an unwatermarked graph $G(V, E)$ with $n = |V|$
a watermarked graph G'
Output: is a message W embedded in G' ?
Algorithm:
if G is not a subgraph of G' **then**
  return FALSE
j:=0
for each i from 1 to n **do**
  if find the nearest two vertices v_{i_1}, v_{i_2}
    not connected to v_i in G **then**
    j++
    if $(v_i, v_{i_2}) \in G'$ **then**
      connect v_i to v_{i_2} in G
    else if $(v_i, v_{i_1}) \in G'$ **then**
      connect v_i to v_{i_1} in G
    else // all bits extracted
      exit
if j=0 **then**
  return FALSE
if $|E'| \neq |E| + j$ **then**
  return FALSE
return TRUE

Figure 7: A recognizer for the QP algorithm

EXAMPLE 6. *(A recognizer for the QP algorithm) A recognition for the QP algorithm is in Fig. 7.*

An extreme positive partial recognizer will always say a program has a watermark while an extreme negative partial recognizer will always say a program has no watermarks. These two recognizers are not useful in practice. Now we consider the relative strength of two recognizers.

DEFINITION 11. *(Strength of partial recognizers) Let $PP1$ and $PP2$ be two positive-partial recognizers corresponding to an embedder A. If $\forall P, P' \in \mathbf{P}$, $PP2(P', P) = TRUE \Longrightarrow PP1(P', P) = TRUE$, we say $PP2$ is at least as strong as $PP1$.*

Let $NP1$ and $NP2$ be two negative-partial recognizers corresponding to an embedder A. If $\forall P, P' \in \mathbf{P}$, $NP1(P', P) = TRUE \Longrightarrow NP2(P', P) = TRUE$, we say $NP2$ is at least as strong as $NP1$.

PROPERTY 2. *For an embedder A, $TrivPP(A)$ is the weakest positive-partial recognizer and $Reg(A))$ is the strongest positive-partial recognizer for A; $TrivNP(A)$ is the weakest negative-partial recognizer and $Reg(A)$ is the strongest negative-partial recognizer for A.*

4.2 Blind recognizers

In some situations, in attempts to detect the presence of a watermark in a potential watermarked program, the original program is unavailable. Recognition of a watermark without the original program is called blind recognition.

DEFINITION 12. *(Blind and informed recognizer) For an embedder $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$ and a function $S : \mathbf{P} \rightarrow \{TRUE, FALSE\}$ we make the following definitions.*

If S has the property that $\forall P' \in \mathbf{P}$, if there is a $P \in \mathbf{P}$ and a $W \in candidate(A, P)$ such that $P' = A(P, W)$, then

$S(P') = TRUE$, *we call such an* S *a* blind positive-partial *recognizer for the embedder* A.

If S *has the property that '* $\forall P' \in \mathbf{P}$, $S(P') = TRUE \Longrightarrow$ *there is a* $P \in \mathbf{P}$ *and a* $W \in candidate(A, P)$ *such that* $P' = A(P, W)$, *we call such an* S *a* blind negative-partial *recognizer for the embedder* A.

If S *has the property that* $\forall P' \in \mathbf{P}$, $S(P') = TRUE \Longleftrightarrow$ *there is a* $P \in \mathbf{P}$ *and a* $W \in candidate(A, P)$, *such that* $P' = A(P, W)$, *we call such an* S *a blind recognizer for the embedder* A.

We say that A *is blind recognizable if a blind recognizer algorithm exists for* A.

The combination (A, S) *is called a* blind watermark recognition system, *when* S *is a blind recognizer for* A.

The blind recognizer is of more practical interest than the informed recognizer, and, for the same reasons, blind extraction is more useful than informed extraction. In the following, we present two examples to illustrate the blind recognizer.

Example 7 present an embedder and the blind recognizer corresponding to this embedder.

EXAMPLE 7. *(A blind recognizer) Define an embedder* A *as follows: For any program* P, *if* $W = 101$ *or if* $W = 110$, $A(P, W)$ *is* P *plus an extra constant declaration. Otherwise,* $A(P, W) = P$.

A blind recognizer S *corresponding to* A *is defined as follows. For any* $P' \in \mathbf{P}$, *if* P' *has at least one constant declaration,* $S(P') = TRUE$. *Otherwise,* $S(P') = FALSE$.

THEOREM 2. *For every embedder* A, *there exists one and only one blind recognizer corresponding to* A. *We denote the unique blind recognizer algorithm corresponding to* A *as* $BReg(A)$.

Proof. $\forall P, P' \in \mathbf{P}$, define $R(P', P)$ as follows:
If there is a $P \in \mathbf{P}$ and a $W \in candidate(A, P)$ such that $P' = A(P, W)$, $R(P', P) =$ TRUE. Otherwise, $R(P', P) =$ FALSE.

It is easy to see R is a blind recognizer corresponding to A. □

PROPERTY 3. *For every embedder* A, $BReg(A)$ *is both the blind positive-partial and the negative-partial recognizers corresponding to* A.

EXAMPLE 8. *(Trivial blind partial recognizers) The blind partial recognizer concepts are also very flexible. The following are some trivial blind partial recognizers. For an embedder* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$, *define* $S(P) = TRUE$. *This constant-valued function is a blind positive-partial recognizer for* A. *We call such an* S *a trivial blind positive-partial recognizer and denote it as* $TrivBPP(A)$.

For an embedder $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$, *define* $S(P) = FALSE$. *This constant-valued function is a blind negative-partial recognizer for* A. *We call such a function a trivial blind negative-partial recognizer and denote it as* $TrivBNP(A)$.

For an embedder A, there are several blind positive-partial recognizers and several blind negative-partial recognizers with respect to A. We need to determine their relative strength.

DEFINITION 13. *(Strength of blind partial recognizers) Let* $BPP1$ *and* $BPP2$ *be two blind positive-partial recognizers corresponding to an embedder* A. *If* $\forall W \in \mathbf{W}$ *and* $\forall P' \in \mathbf{P}$, $BPP2(P') = TRUE \Longrightarrow BPP1(P') = TRUE$, *we say* $BPP2$ *is stronger than* $BPP1$.

Let $BNP1$ *and* $BNP2$ *be two blind negative-partial recognizers corresponding to an embedder* A. *If* $\forall W \in \mathbf{W}$ *and* $\forall P' \in \mathbf{P}$, $BNP1(P') = TRUE \Longrightarrow BNP2(P') = TRUE$, *we say* $BNP2$ *is stronger than* $BNP1$.

PROPERTY 4. *For every embedder* A, $TrivBPP(A)$ *is the weakest blind positive-partial recognizer and* $BReg(A)$ *is the strongest blind positive-partial recognizer;* $TrivBNP(A)$ *is the weakest blind negative-partial recognizer and* $BReg(A)$ *is the strongest blind negative-partial recognizer.*

5. A SOFTWARE WATERMARK EMBEDDING AND RECOGNITION SYSTEM

We design a system model using the above concepts we established to embed watermarks into and recognize watermarks from programs as follows.

Embedding subsystem:
For $\forall P \in \mathbf{P}$ and $\forall W \in \mathbf{W}$,
Step 1: Construct the interference graph G of P.
Step 2: Embed the watermark W into the graph G by the QP embedder and we have the watermarked graph G'.
Step 3: Establish interference relationships of some variable pairs in P so that G' is the interference graph of the new program.

Recognition subsystem:
For $\forall P, P' \in \mathbf{P}$,
Step 1: Construct the interference graphs G, G' of P, P', respectively.
Step 2: Recognize the watermark W from the graphs G and G' by one of the QP recognition algorithms.

6. CONCLUSIONS

As we can see from the above definitions, recognition is a very complicated concept in software watermarking. How to construct a good recognition algorithm for a specific situation and purpose still deserves future research.

Algorithmic design, even with an adequate formal statement of the problem to be solved is both an art and a science. Without a precise statement of the problem, we cannot hope to prove the correctness of any algorithm, and indeed we may have difficulty even explaining the purposes of any algorithm.

When we started this research project, we thought that it would be a simple matter to prove the QP algorithm either correct or incorrect. However we could not do this until we devised appropriate definitions for two basic problems in watermarking – recognition and extraction. None of our initial, intuitively-formed, problem definitions were sufficient to support a careful analysis of the QP algorithm; and we found little support for a careful analysis in the published literature. We were, however, successful in devising a serviceable set of definitions, allowing us to complete a careful analysis of the QP algorithm (and its variants). In the process we discovered some subtle bugs and algorithmic issues. Our major findings are summarized very briefly below.

For any software watermark embedder, there is one and only one recognizer corresponding to it. This recognizer is

also the strongest positive-partial and the strongest negative-partial recognizer corresponding to that embedder. There are also a weakest positive-partial recognizer and a negative-partial recognizer corresponding to that embedder, but they are all trivial partial recognizers. The similar results hold for the blind recognizer and the partial recognizers.

Recognition is more flexible than extraction, but it is also hard to develop a good recognition algorithm for a specific situation. We design a model for a software watermark system through register allocation based on the concepts and algorithms from this paper. We will implement and analyse the model in our future work. We will also continue to define these concepts in a random setting.

We did not consider the attack issue in this paper, but how to recognize watermarks from attacked programs is also an important problem in software watermarking. It is still a challenging research topic for our future work. We will also study how to combine software obfuscation [7, 8, 41, 43, 44] and artificial intelligence [45] with software watermarking to develop more secure software watermarks. Another interesting research topic is to apply techniques of software watermarking to intelligence and security informatics [14].

7. ACKNOWLEDGMENTS

This work is in part supported by the New Economy Research Fund of New Zealand. The authors thank Jasvir Nagra for his insightful comments on an early draft of this paper. The helpful comments and suggestions for this paper from the three anonymous referees are appreciated. The authors are also deeply indebted to Barbara Thomborson for her professional proofreading and editing this paper.

8. REFERENCES

[1] R. Agrawal, P. Haas, and J. Kiernan. Potpourri: A system for watermarking relational databases. In *the 2003 ACM SIGMOD international conference on Management of data*, June 2003.

[2] R. Agrawal, P. Haas, and J. Kiernan. Watermarking relational data: framework, algorithms and analysis. *The International Journal on Very Large Data Bases*, 12(2), Aug 2003.

[3] M. Atallah, V. Raskin, C. Hempelmann, M. Karahan, R. Sion, K. Triezenberg, and U. Topkara. Natural language watermarking and tamperproofing. In *Fifth Information Hiding Workshop*, volume 2578 of *LNCS*, pages 196–212, 2003.

[4] C. Collberg, S. Jha, D. Tomko, and H. Wang. Uwstego: A general architecture for software watermarking. *Technical Report TR04–11*, Aug. 31 2001.

[5] C. Collberg and C. Thomborson. Software watermarking: Models and dynamic embeddings. In *Proceedings of Symposium on Principles of Programming Languages, POPL'99*, pages 311–324, 1999.

[6] C. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. *IEEE Transactions on Software Engineering*, 28:735–746, Aug. 2002.

[7] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. In *Tech. Report, No.148, Dept. of Computer Sciences, The University of Auckland*, 1997.

[8] C. Collberg, C. Thomborson, and D. Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *POPL'98*, pages 184–196, 1998.

[9] C. Collberg, C. Thomborson, and D. Low. On the limits of software watermarking. In *Technical Report #164, Department of Computer Science, The University of Auckland*, 1998.

[10] D. Grover. *The Protection of Computer Software - Its Technology and Applications*. Cambridge University Press, 2 edition, 1997.

[11] G. Hachez. *A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards*. PhD thesis, Universite Catholique de Louvain, Mar 2003.

[12] N. J. Hopper, J. Langford, and L. von Ahn. Provably secure steganography. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 77–92, 2002.

[13] H. C. Kim, E. T. Lin, O. Guitart, and E. J. Delp. Further progress in watermark evaluation testbed (wet). In *Security, Steganography, and Watermarking of Multimedia Contents*, volume 5681 of *Proceedings of SPIE*, pages 241–251. SPIE, 2005.

[14] S. Mehrotra, D. D. Zeng, H. Chen, B. M. Thuraisingham, and F.-Y. Wang. Intelligence and security informatics. In *ISI 2006*, volume 3975 of *LNCS*, May 2006.

[15] A. Monden, H. Iida, et al. A watermarking method for computer programs (in japanese). In *Proceedings of the 1998 Symposium on Cryptography and Information Security, SCIS'98*. Institute of Electronics, Information and Communication Engineers, Jan. 1998.

[16] A. Monden, H. Iida, K. ichi Matsumoto, K. Inoue, and K. Torii. Watermarking java programs. In *International Symposium on Future Software Technology '99*, pages 119–124, October 1999.

[17] P. Moulin and A. Ivanovic. Game-theoretic analysis of watermark detection. In *ICIP (3)*, pages 975–978, 2001.

[18] P. Moulin and J. O'Sullivan. Information–theoretic analysis of watermarking. In *Proc. Int. Conf. on Ac., Sp. and Sig. Proc. (ICASSP)*, 2000.

[19] P. Moulin and J. O'Sullivan. Information–theoretic analysis of information hiding. *IEEE Transactions on Information Theory*, 49(3):563–593, 2003.

[20] G. Myles and C. Collberg. Detecting software theft via whole program path birthmarks. In *Information Security Conference*, 2004.

[21] G. Myles and C. Collberg. Software watermarking through register allocation: Implementation, analysis, and attacks. In *LNCS 2971*, pages 274–293, 2004.

[22] G. Myles and C. Collberg. Software watermarking via opaque predicates: Implementation, analysis, and attacks. In *ICECR-7*, 2004.

[23] G. Myles and H. Jin. Self-validating branch-based software watermarking. In *IH 2005*, volume 3727 of *LNCS*, pages 342–356. Springer Verlag, 2005.

[24] J. Nagra and C. Thomborson. Threading software watermarks. In *IH'04*, 2004.

[25] J. Nagra, C. Thomborson, and C. Collberg. A functional taxonomy for software watermarking. In M. J. Oudshoorn, editor, *Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, Melbourne, Australia, 2002. ACS.

[26] J. Nagra, C. Thomborson, and C. Collberg. Software watermarking: Protective terminology. In *Proceedings of the ACSC 2002*, 2002.

[27] J. Palsberg, S. Krishnaswamy, K. Minseok, D. Ma, Q. Shao, and Y. Zhang. Experience with software watermarking. In *ACSAC '00*, pages 308–316. IEEE, 2000.

[28] J. Pastuszak, D. Michalek, and J. Pieprzyk. Copyright protection of object-oriented softwares. In *ICICS 2001*, volume 2288 of *LNCS*, pages 186–199, 2002.

[29] J. Pieprzyk. Fingerprints for copyright software protection. In M. Mambo and Y. Zheng, editors, *Proceedings of the Second International Workshop on Information Security, ISW'99 (LNCS 1729)*, pages 178–190, Germany, 1999. Springer.

[30] G. Qu and M. Potkonjak. Analysis of watermarking techniques for graph coloring problem. In *IEEE/ACM International Conference on Computer Aided Design, '98*, pages 190–193, 1998.

[31] G. Qu and M. Potkonjak. Hiding signatures in graph coloring solutions. In *Information Hiding Workshop '99*, pages 348–367, 1999.

[32] R. Sion, M. J. Atallah, and S. Prabhakar. wmdb.: Rights protection for numeric relational data. In *ICDE*, page 863. IEEE Computer Society, 2004.

[33] R. Sion, M. J. Atallah, and S. Prabhakar. Rights protection for categorical data. *IEEE Trans. Knowl. Data Eng.*, 17(7):912–926, 2005.

[34] R. Sion, M. J. Atallah, and S. Prabhakar. Rights protection for relational data. *IEEE Trans. Knowl. Data Eng.*, 16(12):1509–1525, 2005.

[35] J. Stern, G. Hachez, F. Koeune, and J.-J. Quisquater. Robust object watermarking: Application to code. In *Information Hiding Workshop'99*, pages 368–378, 1999.

[36] C. Thomborson, J. Nagra, Somaraju, and Y. He. Tamper-proofing software watermarks. In *Proc. Second Australasian Information Security Workshop(AISW2004)*, pages 27–36, 2004.

[37] M. Topkara, C. M. Taskiran, and E. J. Delp. Natural language watermarking. In *Security, Steganography, and Watermarking of Multimedia Contents*, volume 5681 of *Proceedings of SPIE*, pages 441–452. SPIE, 2005.

[38] R. Venkatesan, V. Vazirani, and S. Sinha. A graph theoretic approach to software watermarking. In *Information Hiding Workshop '00*, March 2000.

[39] W. Zhu and C. Thomborson. Algorithms to watermark software through register allocation. In *DRMTICS 2005*, volume 3919 of *LNCS*, pages 180–191, October 2005.

[40] W. Zhu and C. Thomborson. On the QP algorithm in software watermarking. In *IEEE ISI 2005*, volume 3495 of *LNCS*, pages 646–647, May 2005.

[41] W. Zhu and C. Thomborson. A provable scheme for homomorphic obfuscationin in software security. In *The IASTED International Conference on Communication, Network and Information Security, CNIS'05*, pages 208–212, Phoenix, USA, Nov 2005.

[42] W. Zhu, C. Thomborson, and F.-Y. Wang. A survey of software watermarking. In *IEEE ISI 2005*, volume 3495 of *LNCS*, pages 454–458, May 2005.

[43] W. Zhu, C. Thomborson, and F.-Y. Wang. Application of homomorphic function to software obfuscation. In *WISI 2006*, volume 3917 of *LNCS*, pages 152–153, April 2006.

[44] W. Zhu, C. Thomborson, and F.-Y. Wang. Obfuscate arrays by homomorphic functions. In *Special Session on Data Security and Privacy in IEEE GrC 2006, to appear*, pages 770–773, May 2006.

[45] W. Zhu and F.-Y. Wang. Covering based granular computing for conflict analysis. In *IEEE ISI 2006*, volume 3975 of *LNCS*, pages 566–571, 2006.

[46] W. Zhu and C. Thomborson. Extraction in software watermarking. In *to appear in ACM Multimedia and Security Workshop, 26-27, September, 2006, Geneva, Switzerland*, 2006.