WATERMARKING IMAGES USING 2D REPRESENTATIONS OF SELF-INVERTING PERMUTATIONS

Maria Chroni, Angelos Fylakis, and Stavros D. Nikolopoulos
Department of Computer Science, University of Ioannina, GR-45110 Ioannina, Greece
{*mchroni, afylakis, stavros*}*@cs.uoi.gr*

Keywords: Watermarking Techniques; Image Watermarking Systems; Intellectual Property Rights; Color Images; Self-inverting Permutations; 2D-representations of Permutations; Encoding; Decoding; Algorithms.

Abstract: In this work we propose an efficient and easily implemented codec system, which we named WaterIMAGE, for watermarking images that are intended for uploading on the web and making them public online. An important fact of our system is that it suggests a way in which an integer number can be represented in a two dimensional grid and, thus, since images are two dimensional objects that representation can be efficiently marked on them. In particular, our system uses an efficient technique which is based on a 2D representation of self-inverting permutations and mainly consists of two components: the first component contains an encoding algorithm which encodes an integer w into a self-inverting permutation π^* and a decoding algorithm which extracts the integer w from π^*, while the second component contains codec algorithms which are responsible for embedding a watermark into an image I, resulting the image I_w, and extract it from I_w. Our system incorporates important properties which allow us to successfully extract the watermark w from the image I_w even if the input image has been compressed with a lossy method and/or rotated. All the system's algorithms have been developed and tested in JAVA programming environment.

1 INTRODUCTION

Internet technology, in modern communities, becomes day by day an indispensable tool for everyday life since most people use it on a regular basis and do many daily activities online (Garfinkel, 2001). As a consequence, transferring digital information via the Internet, such as audio, pictures, video, or software, has also become very popular during the last years.

It is without any doubt that nowdays images, apart from text, are the most frequent type of data that can be found on the internet. Of course this frequent use of the internet means that measures taken for internet security are indispensable since the web is not risk-free (Chun-Shien et al., 2000; Davis, 1997). One of those risks is the fact that the web is an environment where intellectual property is under threat.

Concerning the images, a characteristic type of intellectual material, people hesitate to upload and transfer them via the internet because of the ease of intercepting, copying and redistributing digital images in their exact original form (O'Ruanaidh et al., 1996). Encryption is not the problem's solution in most cases, as most people that upload images in a website want them to be visible by everyone, but safe and theft protected as well. And that's where watermarks come to place. Thanks to watermarks someone can claim the property of an image if he previously inserted one in it. Image watermarks can be visible or not, but if we don't want any cosmetic changes in an image then a not visible watermark should be used and that's what our work suggests, a technique according to which invisible watermarks are embedded into images using features of the image's space domain and graph theory as well.

We next briefly describe the main idea behind the watermarking technique, some issues about intellectual property rights (IP), the motivation of our work, and our contribution which is a technique which enables us to embed a watermark that initially has the form of an integer into a two dimensional structure which in our case is an image.

Watermarking. Watermarks are symbols which are placed into physical objects such as documents, photos and bank notes and their purpose is to carry information about an object's authenticity. In our case the watermarks have digital form and they are embedded into digital objects, this technique is called digital watermarking. Digital watermarking (or, simply,

watermarking) is a technique for protecting the intellectual property of a digital object; the idea is simple: a unique identifier, which is called *watermark*, is embedded into a digital object which may be used to verify its authenticity or the identity of its owners (Grover, 1997; Collberg and Nagra, 2010). A digital object may be audio, picture, video, or software, and the watermark is embedded into object's data through the introduction of errors not detectable by human perception (Cox et al., 1996); note that, if the object is copied then the watermark also is carried in the copy.

The watermarking problem can be described as the problem of embedding a watermark w into an object I and, thus, producing a new object I_w, such that w can be reliably located and extracted from I_w even after I_w has been subjected to transformations (Collberg and Nagra, 2010); for example, compression in case the object is an image.

There are two general types of watermarking, namely, *visible* and *invisible* watermarking. In visible watermarking, information (i.e., the watermark) is visible in the object, i.e., audio, image, or video. For example, when a television broadcaster adds its logo to the corner of transmitted video, this is a visible watermark. Moreover, there are many watermarking tools that allow us to quickly and easily protect our images with a visible watermark; with the many watermarking options available, we are able to personalize our images in a variety of ways. In invisible watermarking, information is added as digital data to object, but it cannot be perceived as such (although it may be possible to detect that some amount of information is hidden in the object).

It is worth noting that although digital watermarking has made considerable progress and become a popular technique for copyright protection of multimedia information (Cox et al., 1996; Tamada et al., 2004), our work proposes something new. And that is the fact that we can transform a watermark from numerical form into a representation in a two dimensional grid. Also we point out certain properties of this representation with very interesting characteristics.

Intellectual Property. The term intellectual property (IP) refers to a creation of a mind for which a set of exclusive rights are recognized (Raysman et al., 1999). That creation may have any form possible; for example, it may be a work of art, an invention, literary or artistic work, a discovery or even a phrase. More precisely, IP can be divided into two categories: *industrial property*, which includes inventions (patents), trademarks, industrial designs, and geographic indications of source; and *copyright*, which includes literary and artistic works such as novels, poems, plays, films, musical works, drawings, paintings, photographs, sculptures, and architectural designs.

The objective of recognizing intellectual property is to encourage innovation. That is because people won't have the incentive to create if they are not legally protected in order to get the social value that they deserve from their creations (Lemley, 2005). Of course the world's evolution and economic growth depends on creations and inventions and that makes intellectual property such an important and vital aspect (Jain et al., 2009).

Images are a very characteristic example of intellectual material and our work suggests a solution for protecting this kind of intellectual property.

Motivation. We believe that protecting intellectual material on the web is one of the major issues concerning the proper use of the internet. Digital images are a very characteristic part of this material found online and our target is to make people feel free to upload their images without hesitating because of the fear of their work being unauthorized used.

Watermarking is the ideal solution for protecting your property of the images and keeping them visible to the public as well, so research towards imperceptible secure and robust image watermarking techniques is vital. As mentioned, there are already various methods that can achieve that, but every single method requires attention and that's because we can not discriminate a specific method as the best. Every case has its ideal solution and the same idea applies for image watermarking.

Concerning watermarking, a system should efficiently watermark images that are about to be uploaded on the web, where users copy and use images all the time and sometimes make certain modifications to them. We considered important to take into account this fact and provide a method which uses watermarks, robust under these modifications. Such a modification might be scaling or even rotation especially if the image is an indeterminate depiction.

Contribution. In this work we present an efficient and easily implemented codec system, which we named *WaterIMAGE*, for watermarking images that we are interested in uploading in the web and making them public online; this way web users are now enabled to consider how to protect their own images.

What is important for our system is the fact that it suggests a way in which an integer number can be represented in a two dimensional grid, and thus, since images are two dimensional objects that representation can be efficiently marked on them resulting the watermarked images.

In particular, our system uses an efficient tech-

nique for watermarking images, which is based on a 2D representation of self-inverting permutations (or, for short, SIP), and mainly consists of two main components:

- The first component consists of an encoding algorithm which encodes an integer w into a self-inverting permutation π^* and a decoding algorithm which extracts the integer w from the self-inverting permutation π^*.
- The second component consists of codec algorithms which are responsible for embedding a watermark into an image I, resulting the image I_w, and extract it from the watermarked image I_w. Our codec algorithms use as watermark a self-inverting permutation and embed it into images using a 2D representation of it.

More precisely, we first present an efficient algorithm which encodes a number (integer) w as self-inverting permutation π^*. Our algorithm, which we call Encode_W-to-SIP, takes as input an integer w, computes first its binary representation $b_1 b_2 \cdots b_n$, then constructs a bitonic permutation on $2n+1$ numbers, and finally produces a self-inverting permutation π^* of length $n^* = 2n+1$ in $O(n)$ time and space. We also present a decoding algorithm which extracts the integer w from the self-inverting permutation π^* within the same time and space complexity; we call the decoding algorithm Decode_SIP-to-W.

Having designed an efficient method for encoding integers as self-inverting permutations, we next describe an algorithm for encoding a self-inverting permutation into an image I using a 2D representation. In particular, we propose the algorithm Encode_SIP-to-IMAGE which embeds the self-inverting permutation π^* into an image I by first mapping the elements of π^* into an $n^* \times n^*$ matrix A^* and then using the information stored in A^* to change specific pixels of image I resulting the watermarked image I_w. We also propose an efficient and easily implemented algorithm, the algorithm Decode_IMAGE-to-SIP, which extract the self-inverting permutation π^* from the watermarked image I_w first by locating the positions of certain pixels in I_w which enable us to contract the 2D representation of the self-inverting permutation π^*.

It is worth noting that our system incorporates such properties which allow us to successfully extract the watermark w from the image I_w even if the input image of algorithm Decode_IMAGE-to-SIP has been compressed with a lossy method and/or rotated.

We have evaluated the embedding and extracting algorithms by testing them on various and different in characteristics images that were initially in JPEG format and we had positive results as the watermark was successfully extracted at every case even if the image was converted back into JPEG format. What is more, the method is open to extensions as the same method might be used with a different marking procedure part of the Encode_SIP-to-IMAGE algorithm.

All the system's algorithms have been initially developed and tested in MATLAB (Gonzalez et al., 2003) and then redeveloped and also tested in JAVA.

Our system has optimal time and space performance. Let $N \times M$ be the size of the input image, that is, the number of pixels of both the original image I and the watermarked image I_w. The total time performance of our codec system, neglecting the conversion of the input image I into raw raster format, is $N \times n^*$ for embedding the watermark w into I and $N + M + (n^* \times n^*) \times \log(n^* \times n^*)$ for extracting w from the watermarked image I_w. Moreover, the extra space needed by our codec system is linear in the size of the input image since it uses only some extra auxiliary variables and an auxiliary matrix for the 2DM representation of the self-inverting permutation.

Road Map. The paper is organized as follows. In Section 2 we present efficient representations of the two main objects of our watermarking system: the self-inverting permutations and the digital color images. We also describe the algorithms for encoding/decoding integers w into/from self-inverting permutations. In Section 3 we describe the main codec algorithms of the proposed image watermarking system. In Section 4 we show important properties of our system and its time and space performance, while in Section 5 we conclude the paper and discuss possible future extensions.

2 BASIC TOOLS

In this section we present basic tools which are used by our image watermarking system. In particular, we first describe discrete structures, namely, permutations, self-inverting permutations, and bitonic permutations, and then briefly outline a codec system (encoding/decoding algorithms) which encodes an integer w into a self-inverting permutation π^* and extracts it from π^*. Finally, we propose a 2D representation of permutations and give a 3D representation of color images.

2.1 Self-inverting Permutations

Informally, a permutation of a set of objects S is an arrangement of those objects into a particular order, while in a formal (mathematical) way a permutation

of a set of objects S is defined as a bijection from S to itself (i.e., a map $S \to S$ for which every element of S occurs exactly once as image value).

Permutations may be represented in many ways. The most straightforward is simply a rearrangement of the elements of the set $N_n = \{1,2,\ldots,n\}$; in this way we think of the permutation $\pi^* = (5,6,9,8,1,2,7,4,3)$ as a rearrangement of the elements of the set N_9 such that "1 goes to 5", "2 goes to 6", "3 goes to 9", "4 goes to 8", and so on (Sedgewick and Flajolet, 1996; Golumbic, 1980). Hereafter, we shall say that π^* is a permutation over the set N_9.

Definition 2.1.1. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a permutation over the set N_n, where $n > 1$. The inverse of the permutation π is the permutation $\tau = (q_1, q_2, \ldots, q_n)$ with $q_{\pi_i} = \pi_{q_i} = i$. A *self-inverting permutation* (or, for short, SIP) is a permutation that is its own inverse: $\pi_{\pi_i} = i$.

By definition, every permutation has a unique inverse, and the inverse of the inverse is the original permutation. Clearly, a permutation is a SIP (self-inverting permutation) if and only if all its cycles are of length 1 or 2; hereafter, we shall denote a 2-cycle as $c = (x,y)$ and an 1-cycle as $c = (x)$, or, equivalently, $c = (x,x)$.

The permutation $\pi^* = (5,6,9,8,1,2,7,4,3)$ is a SIP with cycles: $(1,5)$, $(2,6)$, $(3,9)$, $(4,8)$, and $(7,7)$.

2.2 Encoding Numbers as SIPs

Next, we present an algorithm for encoding an integer w into a self-inverting permutation π^* and an algorithm for extracting w from π^*; both algorithms run in $O(n)$ time, where n is the length of the binary representation of the integer w (author's algorithms). The encoding process uses the notion of *Bitonic Permutations* which we briefly describe below.

Bitonic Permutations. The key-object in our algorithm for encoding integers as self-inverting permutations is the bitonic permutation: a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ over the set N_n is called bitonic if either monotonically increases and then monotonically decreases, or else monotonically decreases and then monotonically increases. For example, the permutations $\pi_1 = (1,4,6,7,5,3,2)$ and $\pi_2 = (6,4,3,1,2,5,7)$ are both bitonic.

Our encoding algorithm uses only bitonic permutations that monotonically increase and then monotonically decrease. Let π be such a bitonic permutation over the set N_n and let π_i, π_{i+1} be the two consecutive elements of π such that $\pi_i > \pi_{i+1}$. Then, the sequence $X = (\pi_1, \pi_2, \ldots, \pi_i)$ is called first increasing subsequence of π and the sequence $Y = (\pi_{i+1}, \pi_{i+2}, \ldots, \pi_n)$ is called first decreasing subsequence of π.

We next give some notations and terminology we shall use throughout the encoding and decoding process. Let w be an integer number. We denote by $B = b_1 b_2 \cdots b_n$ the binary representation of w. If $B_1 = b_1 b_2 \cdots b_n$ and $B_2 = d_1 d_2 \cdots d_m$ be two binary numbers, then the number $B_1 || B_2$ is the binary number $b_1 b_2 \cdots b_n d_1 d_2 \cdots d_m$. The binary sequence of the number $B = b_1 b_2 \cdots b_n$ is the sequence $B^* = (b_1, b_2, \ldots, b_n)$ of length n.

Let $B = b_1 b_2 \cdots bn$ be a binary number. Then, $flip(B) = b'_1 b'_2 \cdots b'_n$ is the binary number such that $b'_i = 0$ (1 resp.) if and only if $b_i = 1$ (0 resp.), $1 \le i \le n$.

Algorithm W-to-SIP: We present, with the help of an example, an algorithm for encoding an integer as self-inverting permutation. Our algorithm, which we call Encode_W-to-SIP, takes as input an integer w, computes the binary representation $b_1 b_2 \cdots b_n$ of w, and then produces a self-inverting permutation π^* in $O(n)$ time. We next describe the encoding process for the number 12:

Example W-to-SIP: Let $w = 12$ be the input watermark integer in the algorithm Encode_W-to-SIP. We first compute the binary representation $B = 1100$ of the number 12; then we construct the binary number $B' = 000011001$ and the binary sequence $B^* = (1,1,1,1,0,0,1,1,0)$ of $flip(B')$; we compute the sequences $X = (5,6,9)$ and $Y = (1,2,3,4,7,8)$, and then construct the bitonic permutation $\pi = (5,6,9,8,7,4,3,2,1)$ on $n' = 9$ numbers; since $n' = 9$ odd, we select 4 pairs $(5,1)$, $(6,2)$, $(9,3)$, $(8,4)$ and the number 7 and then construct the self-inverting permutation $\pi^* = (5,6,9,8,1,2,7,4,3)$.

Algorithm SIP-to-W: Having presented the encoding algorithm Encode_W-to-SIP, let us now present an extraction algorithm, that is, an algorithm for decoding a self-inverting permutation. More precisely, our extraction algorithm, which we call Decode_SIP-to-W, takes as input a self-inverting permutation π^* produced by the algorithm Encode_W-to-SIP and returns its corresponding integer w. The time complexity of the decode algorithm is also $O(n)$, where n is the length of the permutation π^*. We next describe, through an example, the decoding process:

Example SIP-to-W: Let $\pi^* = (5,6,9,8,1,2,7,4,3)$ be a self-inverting permutation produced by the algorithm Encode_W-to-SIP. The cycle representation of π^* is the following: $(1,5)$, $(2,6)$, $(3,9)$, $(4,8)$, (7); from the cycles we construct the permutation $\pi = (5,6,9,8,7,4,3,2,1)$; then, we compute first

increasing subsequence $X = (5,6,9)$ and the first decreasing subsequence $Y = (8,7,4,3,2,1)$; we then construct the binary sequence $B^* = (1,1,1,1,0,0,1,1,0)$ of length 9; we flip the elements of B^* and construct the sequence $B' = (0,0,0,0,1,1,0,0,1)$; the binary number 1100 is the integer $w = 12$.

2.3 2DM Representations

Given a permutation π over the set $N_n = \{1,2,\ldots,n\}$, we first define a two-dimensional representation (2D-representation) of the permutation π that is useful for studying properties which help us to define, later, a more suitable representation of π for efficient use in our watermarking system.

In this representation, the elements of the permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ are mapped in specific cells of an $n \times n$ matrix A as follows:

- integer $i \longrightarrow$ entry $A(\pi_i^{-1}, i)$

or, equivalently,

- the cell at row i and column π_i is labeled by the number π_i, for each $i = 1,2,\ldots,n$.

Figure 1 shows the 2D representation of the self-inverting permutation $\pi = (5,6,9,8,1,2,7,4,3)$.

	1	2	3	4	5	6	7	8	9
1					**5**				
2						**6**			
3									**9**
4								**8**	
5	**1**								
6		**2**							
7							**7**		
8				**4**					
9			**3**						

Figure 1: A 2D representation of the self-inverting permutation $\pi = (5,6,9,8,1,2,7,4,3)$.

Note that, there is one label in each row and in each column, so each cell in the matrix A corresponds to a unique pair of labels; see, (Sedgewick and Flajolet, 1996) for a long bibliography on permutation representations and also in (author's paper) for a DAG representation.

Based on the previous 2D representation of a permutation, we next propose a two-dimensional marked

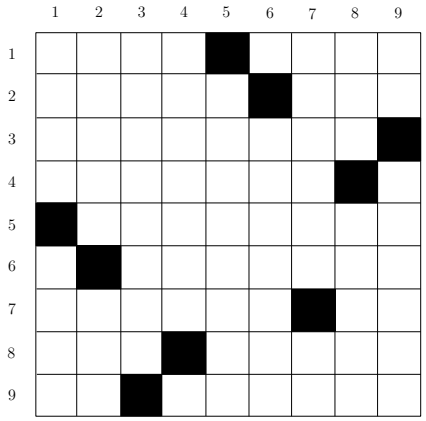

Figure 2: A 2DM representation of the self-inverting permutation $\pi = (5,6,9,8,1,2,7,4,3)$.

representation (2DM representation) of a permutation which is an efficient tool for watermarking images.

In our 2DM representation, a permutation π over the set $N_n = \{1,2,\ldots,n\}$ is represented by an $n \times n$ matrix A^* as follows:

- the cell at row i and column π_i is marked by a specific symbol, for each $i = 1,2,\ldots,n$.

Figure 2 shows the 2DM representation of the permutation π. Note that, as in the 2D representation, there is also one symbol in each row and in each column of the matrix A^*.

We next present an algorithm which extracts the permutation π from its 2DM representation matrix. More precisely, let π be a permutation over N_n and let A^* be the 2DM representation matrix of π (see, Figure 2); given the matrix A^*, we can easily extract π from A^* in linear time (in the size of matrix A^*) by the following algorithm:

Algorithm Extract_π_from_2DM
Input: the 2DM representation matrix A^* of π;
Output: the permutation π;

1. For each row i of matrix A^*, $1 \le i \le n$, do:
   find the marked cell and let j be its column;
   set $\pi_i \leftarrow j$;
2. Return the permutation π;

Remark 2.3.1. It is easy to see that the resulting permutation π, after the execution of Step 1, can be taken by reading the matrix A^* from top row to bottom row and write down the positions of its marked cells. Since the permutation π is a self-inverting permutation, its 2D matrix A has the following property:

- $A(i,j) = j$ if $\pi_i = j$, and
- $A(i,j) = 0$ otherwise, $1 \le i,j \le n$.

Thus, the corresponding matrix A^* is symmetric:

- $A^*(i,j) = A^*(j,i) =$ "mark" if $\pi_i = j$, and
- $A^*(i,j) = A^*(j,i) = 0$ otherwise, $1 \leq i,j \leq n$.

Based on this property, it is also easy to see that the resulting permutation π can be also taken by reading the matrix A^* from left column to right column and write down the positions of its marked cells.

2.4 Color Images

A digital image is a numeric representation of a 2-dimensional image; it has a finite set of values, called *picture elements* or *pixels*, that represent the brightness of a given color at any specific point in the image (Gonzalez and Woods, 2007).

A digital image contains a fixed number of rows and columns of pixels which are usually stored in computer memory as a two-dimensional matrix I of numeric values; in our system the numeric values are integers from 0 to 255. When we say that an image has a *resolution* of $N \times M$ we mean that its two-dimensional matrix I contains N rows and M columns and the value of each entry $I(i,j)$, i.e., the value of each pixel, is an integer k_0 (grayscale image), or a triple of integers (k_1,k_2,k_3) (color image), $0 \leq k_0,k_1,k_2,k_3 \leq 255$.

There are several models used for representing color. In our system, we use the *RGB* model; it is an additive color model in which *red*, *green*, and *blue* light is added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, Red, Green, and Blue (Gonzalez and Woods, 2007; Pascale, 2003).

The range of colors can be represented on the Cartesian 3-dimensional system as a cube with the following characteristics:

- on the x-axis (R-axis) we have the brightness of the **red** color,
- on the y-axis (G-axis) we have the brightness of the **green** color, and
- on the z-axis (B-axis) we have the brightness of the **blue** color.

Figure 3 shows the 3D topology of the colors. For example, the white color (255, 255, 255) is located in the front upper right point of the color cube.

In our system, since a color is a triple of integers (x,y,z), a digital image I of resolution $N \times M$ (i.e., it contains N rows and M columns) is stored in a three-dimensional matrix Img of size $N \times M \times 3$ as follows:

- if the pixel $I(i,j)$ of the image I has (x,y,z) color, then $Img(i,j,1) = x$, $Img(i,j,2) = y$, and $Img(i,j,3) = z$.

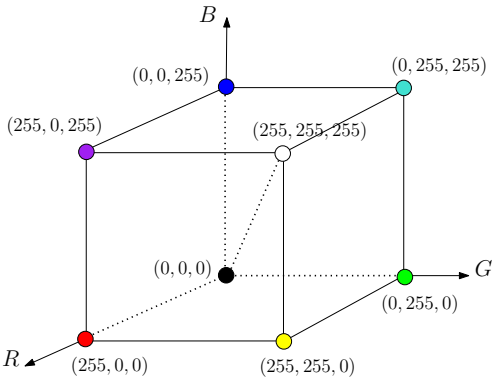

Figure 3: The range of colors represented on the Cartesian 3-dimensional system.

For example, let (240, 29, 35) be the color of the upper left pixel of an image I, i.e., $I(1,1) = (240,29,35)$. Then, in our system $Img(1,1,1) = 240$, $Img(1,1,2) = 29$, and $Img(1,1,3) = 35$.

3 OUR IMAGE WATERMARKING SYSTEM

Having proposed an efficient method for encoding integers as self-inverting permutations using the bitonic property of a permutation, and the 2DM representation of self-inverting permutations, we next describe the two main algorithms of our image watermarking system; the encoding algorithm Encode_SIP-to-IMAGE which encodes a self-inverting permutation π^*, corresponding to watermark w, into an image I resulting the watermarked image I_w and the decoding algorithm Decode_IMAGE-to-SIP which extracts the permutation π^* from the image I_w.

3.1 Embed Watermark into Image

We next describe the algorithm Encode_SIP-to-IMAGE of our codec system which embeds a self-inverting permutation (SIP) π^* into an image I; recall that, in our system we use a SIP π^* over the set N_{n^*} for encoding the watermark w, where $n^* = 2n+1$ and n is the length of the binary representation of the integer w (author's technique); see, Subsection 2.2.

The algorithm takes as input a SIP π^* and an image I, in which the user wants to embed the watermark $w = \pi^*$, and produces the watermarked image I_w; it works as follows:

Step 1: The algorithm first computes the 2DM representation of the permutation π^*, that is, it computes the $n^* \times n^*$ array A (see, Subsection 2.3); the

entry (i,π_i^*) of the array A contains the symbol "*", $1 \le i \le n^*$.

Step 2: Next, the algorithm computes the size $N \times M$ of the input image I and do the following: if N is an even number it removes the pixels from the bottom row of I and reduces N by 1, while if M is an even number it removes the pixels from the right column of I and reduces M by 1. The resulting image has size $N^* \times M^*$, where N^* and M^* are both odd numbers.

Step 3: Let n^* be the size of the SIP π^* and let $N^* \le M^*$. Now the algorithm takes the input image I and places on it an imaginary grid $\mathcal{G}$, which covers almost the whole image I, having

$$n^* \times n^* \text{ grid-cells } C_{ij}(I)$$

each $C_{ij}(I)$ of size

$$\lfloor N^*/n^* \rfloor \times \lfloor N^*/n^* \rfloor$$

where, $1 \le i,j \le n^*$.

It places the imaginary grid $\mathcal{G}$ on I as follows: it first locates the central pixel P^0_{cent} of the image I, which is at position $(\lfloor N^*/2 \rfloor + 1, \lfloor M^*/2 \rfloor + 1)$, then locates the central pixel p^0_{ii} of the central grid-cell $C_{ii}(I)$, where $i = \lfloor n^*/2 \rfloor + 1$, and places the grid $\mathcal{G}$ on image I such that both P^0_{cent} and p^0_{ii} have the same position in I.

Step 4: Then it scans the image and goes to each grid-cell $C_{ij}(I)$ (there are always $n^* \times n^*$ grid-cells in any image) and locates the central pixel p^0_{ij} of the grid-cell $C_{ij}(I)$ and also the four pixels p^1_{ij}, p^2_{ij}, p^3_{ij}, and p^4_{ij} around it, $1 \le i,j \le n^*$; hereafter, we shall call these four pixels *cross* pixels.

Then, it computes the difference between the brightness of the central pixel p^0_{ij} and the average brightness of the twelve pixels around it, that is, the pixels $p^{\ell 1}_{ij}$, $p^{\ell 2}_{ij}$, and $p^{\ell 3}_{ij}$ ($\ell = 1,2,3,4$), and stores this value in the variable $\text{dif}(p^0_{ij})$ (see, Figure 4).

Finally, it computes the maximum absolute value of all $n^* \times n^*$ differences $\text{dif}(p^0_{ij})$, $1 \le i,j \le n^*$, and stores it in the variable $\text{Maxdif}(I)$.

Step 5: The algorithm goes again to each central pixel p^0_{ij} of each grid-cell C_{ij} and if the corresponding entry $A(i,j)$ contains the symbol "*", then it increases

- the brightness k^0_{ij} of the central pixel p^0_{ij}, and
- the brightness k^1_{ij}, k^2_{ij}, k^3_{ij}, and k^4_{ij} of its cross pixels.

Actually, it first increases the central pixel p^0_{ij} by the value e^0_{ij} so that it surpasses the image's maximum difference $\text{Maxdif}(I)$ by a constant c; that is,

- $k^0_{ij} + e^0_{ij} = \text{Maxdif}(I) + c$

and, then, it sets the brightness of the four cross pixels p^1_{ij}, p^2_{ij}, p^3_{ij}, and p^4_{ij} equal to k^0_{ij}.

In our system we use $c = 5$, and thus the brightness k^0_{ij} of the central pixel of each grid-cell C_{ij} is increased by e^0_{ij}, where

$$e^0_{ij} = \text{Maxdif}(I) - k^0_{ij} + 5 \tag{1}$$

where, $1 \le i,j \le n^*$.

Let I_w be the resulting image after increasing the brightness of the n^* central and the corresponding cross pixels, with respect to π, of the image I. Hereafter, we call the n^* central pixels of I as *2DM-pixels*; recall that, p^0_{ij} is a 2DM-pixel if $A(i,\pi_i)$ = "*", or, equivalently, the cell (i,π_i) of the matrix A is marked.

Step 6: The algorithm returns the watermarked image I_w.

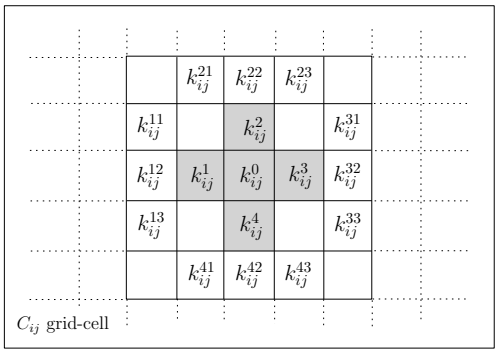

Figure 4: The brightness k^ℓ_{ij} of the central and cross pixels p^ℓ_{ij} of the grid-cell $C_{ij}(I)$, $0 \le \ell \le 4$, and the brightness $k^{\ell m}_{ij}$ of the cycle-cross pixels $p^{\ell m}_{ij}$, $1 \le \ell \le 4$ and $m = 1,2,3$.

Having described our encoding algorithm which embeds a permutation π into an image I, let us now show the efficiency of our algorithm by computing its time and space complexity.

Complexity. We shall compute the complexity of each step of the algorithm; suppose that the input image I has $N \times M$ size (i.e., pixels).

It is easy to see that Step 1 requires $n^* \times n^*$ (asymptotic) time and space, since the length of the permutation π and the size of the array A are n^* and $n^* \times n^*$, respectively.

In Step 2 the algorithm computes the values of the two dimensions of the image I, and thus this computations takes $N + M$ time.

In Step 3 the algorithm places on I the imaginary grid $\mathcal{G}$ having $n^* \times n^*$ grid-cells $C_{ij}(I)$ each of size $\lfloor N^*/n^* \rfloor \times \lfloor N^*/n^* \rfloor$, where $1 \le i,j \le n^*$, and thus it covers $(\lfloor N^*/n^* \rfloor \times \lfloor N^*/n^* \rfloor) \times (n^* \times n^*)$ pixels of the image I. The location of the $n^* \times n^*$ central pixels p^0_{ij}

can be done in $n^* \times n^*$ time, where $1 \leq i, j \leq n^*$ and $n^* < N^*$. Thus, the step takes $(\lfloor N^*/n^* \rfloor n^*)^2$ time.

In Step 4 it computes the difference between the brightness of the central pixel p^0_{ij} and the average brightness of the twelve cycle-cross pixels $p^{\ell m}_{ij}$ around it, $\ell = 1,2,3,4$ and $m = 1,2,3$. This difference, denoted by $\text{dif}(p^0_{ij})$, is computed as follows:

$$\text{dif}(p^0_{ij}) = \left| k^0_{ij} - \frac{\sum_{\ell=1}^{4} \sum_{m=1}^{3} k^{\ell m}_{ij}}{12} \right| \qquad (2)$$

where,

$$k^{\ell m}_{ij} = \frac{x^{\ell m}_{ij} + y^{\ell m}_{ij} + z^{\ell m}_{ij}}{3}. \qquad (3)$$

Recall that, the values $x^{\ell m}_{ij}$, $y^{\ell m}_{ij}$, and $z^{\ell m}_{ij}$ compose the brightness $k^{\ell m}_{ij}$ of the pixel $p^{\ell m}_{ij}$ in the *RGB* model (see, Subsection 2.4). Thus, the $n^* \times n^*$ differences $\text{dif}(p^0_{ij})$ can be computed in $n^* \times n^*$ time and require $n^* \times n^*$ space (i.e., an array of $n^* \times n^*$ size).

Finally, in this step the algorithm computes the maximum absolute value $\text{Maxdif}(I)$ of all $n^* \times n^*$ differences $\text{dif}(p^0_{ij})$, that is,

$$\text{Maxdif}(I) = max\{\text{dif}(p^0_{ij}) | 1 \leq i, j \leq n^*\} \qquad (4)$$

which obviously takes also $n^* \times n^*$ time.

The only operation performed in Step 5 is the increment of the brightness k^0_{ij} of each central pixel and the brightness k^1_{ij}, k^2_{ij}, k^3_{ij}, and k^4_{ij} of its four cross pixels by the value e^0_{ij} (see, Equation 1); it obviously takes $n^* \times n^*$ time since there are $n^* \times n^*$ such central pixels.

Based on the above step-by-step analysis of our encoding algorithm Encode_SIP-to-IMAGE we conclude that it runs (asymptotically) in order $N \times n^*$ time and requires $n^* \times n^*$ space, where N is the smallest dimension of the input image I and n^* is the size of the SIP.

Remark 3.1.1. The values x^{ℓ}_{ij}, y^{ℓ}_{ij}, and z^{ℓ}_{ij} which compose the brightness k^{ℓ}_{ij} of the pixel p^{ℓ}_{ij} are stored in the array *Img* at the entries $(i', j', 1)$, $(i', j', 2)$, and $(i', j', 3)$, respectively (see, Subsection 2.4). Note that, (i', j') is the position of pixel p^{ℓ}_{ij} in image I, while (i, j) is the position of pixel p^{ℓ}_{ij} in the $n^* \times n^*$ grid.

3.2 Extract Watermark from Image

Next we describe our decoding algorithm which is responsible for extracting the watermark $w = \pi^*$ form

Figure 5: The original image I.

Figure 6: The watermarked image I_w.

image I_w. In particular, the algorithm, which we call Decode_IMAGE-to-SIP, takes as input a watermarked image I_w and returns the SIP π^* which corresponds to integer watermark w; the steps of the algorithm are the following:

Step 1: The algorithm places again the same imaginary $n^* \times n^*$ grid on image I_w and locates the central pixel p^0_{ij} of each grid-cell $C_{ij}(I)$, $1 \leq i, j \leq n^*$; there are $n^* \times n^*$ central pixels in total. Then, it finds the n^* central pixels $p^0_1, p^0_2, \ldots, p^0_{n^*}$, among the $n^* \times n^*$, with the maximum brightness using a known sorting algorithm (Cormen et al., 2001).

Step 2: In this step, the algorithm takes the n^* grid-cell $C_1, C_2, \ldots, C_{n^*}$ of the image I_w which corre-

spond to n^* central pixels $p_1^0, p_2^0, \ldots, p_{n^*}^0$, and compute an $n^* \times n^*$ matrix A^* as follows:

- Initially, set $A^*(i,j) \leftarrow 0$, $1 \leq i,j \leq n^*$;
- For each grid-cell C_m, $1 \leq m \leq n^*$, do:

  if (i,j) is the position of the grid-cell C_m in the grid $\mathcal{G}$ then set $A^*(i,j) \leftarrow$ "$*$";

It is easy to see that, the $n^* \times n^*$ matrix A^* is exactly the 2DM representation of the self-inverting permutation π^* embedded in image I_w by the algorithm Encode_SIP-to-IMAGE.

Then, the permutation π^* can be extracted from the matrix A^* using the algorithm Extract_π_from_2DM; see, Subsection 2.3.

Step 3: Finally, the algorithm returns the self-inverting permutation π^*.

Let us next compute the time and space efficiency of the proposed decoding algorithm Decode_IMAGE-to-SIP by computing the complexity of each step separately.

Complexity. Again, we suppose, as we did with the encoding algorithm Encode_SIP-to-IMAGE, that the input image I_w has $N \times M$ size (i.e., it consists of $N \times M$ pixels) and $N \leq M$.

In Step 1 the algorithm places on I_w an imaginary $n^* \times n^*$ grid, as the embedding algorithm do on image I, and thus the values of the two dimensions of the image I_w must be known; this computations takes $N+M$ time. Then, the location of the $n^* \times n^*$ central pixels p_{ij}^0 can be done in $n^* \times n^*$ time, $1 \leq i,j \leq n^*$, while the finding of the n^* central pixels, among the $n^* \times n^*$, with the maximum brightness can be done in $(n^* \times n^*) \times \log(n^* \times n^*)$ time; note that, it is well known that fastest sorting algorithm on an input of size n takes $n \log n$ time (Cormen et al., 2001).

In Step 2 the algorithm takes the n^* grid-cell $C_1, C_2, \ldots, C_{n^*}$ of the image I_w which correspond to the n^* central pixels $p_1^0, p_2^0, \ldots, p_{n^*}^0$, and compute an $n^* \times n^*$ matrix A^*. It is easy to see that this computation can be done in $n^* \times n^*$ time. It is also easy to see that the permutation π^* can be extracted from A^*, using the algorithm Extract_π_from_2DM, within the same time. Thus, Step 2 requires $n^* \times n^*$ time. Obviously, Step 3 takes constant time.

Based on the above complexity analysis we conclude that the proposed decoding algorithm Decode_IMAGE-to-SIP extracts the watermark SIP π^* from the image I_w in $N+M+(n^* \times n^*) \times \log(n^* \times n^*)$ time; it requires $n^* \times n^*$ space.

4 Performance

Our image watermarking system mainly consists of four algorithms, each of which is responsible for a particular codec operation:

- Encode_W-to-SIP: algorithm for encoding an integer w into a self-inverting permutation π^*;
- Decode_SIP-to-W: algorithm for extracting w from π^*;
- Encode_SIP-to-IMAGE: algorithm for encoding a self-inverting permutation π^* into an integer I;
- Decode_IMAGE-to-SIP: algorithm for extracting π^* form the watermarked image;

The two algorithms that are considered the basic ones for our system are those responsible for embedding a SIP into an image and extracting the SIP from it.

We next discuss some issues concerning the performance of our image watermarking system. In particular, we mainly focus on the embedding algorithm Encode_SIP-to-IMAGE and the efficiency of watermarking image I_w produced by this algorithm, and also on important properties of the $n^* \times n^*$ matrix A^* which stores the 2DM representation of a SIP. Finally we show the time and space performance of our system by computing the complexity of their algorithms.

It is worth noting that our system incorporates such properties which allow us to successfully extract the watermark w from the image I_w even if the input image I_w of algorithm Decode_IMAGE-to-SIP has been compressed with a lossy method and/or rotated.

We have evaluated the embedding and extracting algorithms by testing them on various and different in characteristics images that were initially in JPEG format and we had positive results as the watermark was successfully extracted at every case even if the image was converted back into JPEG format. What is more, the method is open to extensions as the same method might be used with a different marking procedure part of the Encode_SIP-to-IMAGE algorithm.

All the system's algorithms have been initially developed and tested in MATLAB (Gonzalez et al., 2003) and then redeveloped and also tested in JAVA.

Compression. The experimental results show that the watermark w is "well hidden" in the image I_w. We believe that it is because we mark the image by changing the difference between the brightness of the 2DM-pixels p_{ij}^0 of the $n^* \times n^*$ imaginary grid and its 12 neighborhood pixels around it, that is, the pixels $p_{ij}^{\ell 1}$, $p_{ij}^{\ell 2}$, and $p_{ij}^{\ell 3}$, for $\ell = 1,2,3,4$ (see, Figure 4 and also Step 2 of the embedding algorithm Encode_SIP-to-IMAGE); recall that, we also set the brightness of the

four cross pixels of each 2DM-pixel p^0_{ij}, that is, the pixels p^1_{ij}, p^2_{ij}, p^3_{ij}, and p^4_{ij}, to be equal to the brightness of the 2DM-pixel p^0_{ij}.

Note that, we change the brightness of the 2DM-pixels by increasing them so that they surpass the image's maximum difference Maxdif(I) by a constant c, where in our implementation $c = 5$. We add five because if we compress the image the values of the pixels may slightly change, and we want our watermark to be robust. We also believe that this technique despite being simple, it is efficient because the brightness of each of the n^* marked central pixels does not have a great difference from the brightness of the 12 neighborhood pixels and thus the modified central pixel, along with the cross pixels, does not change something significantly in the image.

Rotation. The watermarked images produced by our embedding method have a property worth to be referenced. And this is certain characteristics noticed at the 2DM representation of the image's watermarks which in our system are self-inverting permutations. Sometimes an image might show an indeterminate depiction, such as a night sky or an aerial view. These types of images might be rotated changing the coordinates of the watermark's marks making invalid the watermark that we are about to extract. Also it is about an indeterminate depiction which does not allow someone to tell which is the right angle of the image.

Thanks to our embedding method's properties this problem can be overcome. It has to do with the coordinates of the marks of a 2DM representation of a self-inverting permutation found on image I_w. Those coordinates allow us to turn the image into the initial angle and then extract the watermark successfully.

The 2DM representation of a self-inverting permutation has the following properties:

(i) The main diagonal of the $n^* \times n^*$ symmetric matrix A^* have always one and only one marked cell, and

(ii) this marked cell are always in the entries (i,i) of A^*, where $i = \lceil \frac{n^*}{2} \rceil + 1, \lceil \frac{n^*}{2} \rceil + 2, \ldots, n^*$.

If the main diagonal of matrix A^* has no marked cell then we rotate the image by 90 degrees. Additionally, if the marked cell of the main diagonal is in entry (i,i) with $i \leq \lceil \frac{n^*}{2} \rceil$, then we turn the image by 180 degrees and thus we end up at the initial image from which we are able to extract the right watermark.

Time and Space Complexity. As far as the time and space complexity of our codec system is concerned, we should mention that it is asymptotically linear in the size (i.e., number $N \times M$ of pixels) of the input images.

More precisely, the embedding algorithm takes $(N \times n^*) + (n^* \times n^*)$ time which is less than the size $N \times M$ of the input image I. Recall that, in our implementation: (i) $N \leq M$, and (ii) the length of the watermark is n^* and thus we always have $n^* \times n^*$ grid-cells.

The extracting algorithm is also very fast since it also operates mainly on the $n^* \times n^*$ grid-cells of the input image I_w. The most time consuming step of the algorithm is that of sorting the $n^* \times n^*$ cental pixels of the image in order to find the n^* pixels with the max brightness; it takes $n^* \times n^* \times \log(n^* \times n^*)$ time.

Finally, it is fair for the time performance of our system to take into consideration the time needed for converting the image I that the system takes as input from the initial format to raw raster format; note that, the system usually uses compressed images as input. It is obvious that the time needed for converting the image I into a *raw raster format* depends on the type of the image selected. The most common types of images would be the JPEG as digital cameras store images of this type and also nearly every image on the WWW (world wide web) is in JPEG format. The compression to a JPEG requires the usage of the DCT (discrete cosine transform); the DCT is similar to a Fourier transform and it is of order n^2, but it is also possible to do the same thing by doing something similar to the FFT (fast fourier transform) which is of order $n \log n$. Note that the same techniques applies for the JIF images which are also popular in the web (Ahmed et al., 1974; Cooley and Tukey, 1965).

Summarizing, the total time performance of our codec system, neglecting the conversion of the input image I into raw raster format, is $N \times n^*$ for embedding the watermark w into I and $N + M + (n^* \times n^*) \times \log(n^* \times n^*)$ for extracting w from the watermarked image I_w. Moreover, the extra space needed by our codec system is linear in the size of the input image, i.e., it uses only some extra auxiliary variables and an auxiliary matrix for the 2DM representation of the self-inverting permutation.

5 CONCLUDING REMARKS

In this paper, we proposed a codec system, which we named WaterIMAGE, for watermarking images that are intended for online publication.

Our system embeds an integer w into an image I using the following main steps: it computes first a self-inverting π^* which corresponds to w, then computes a 2D representation of π^*, and finally selects and marks specific pixels of the image I, according

to 2D representation of π^*, resulting the watermarked image I_w. Moreover, our system efficiently finds the marked pixels of a watermarked image I_w, using the locations of the marked pixels in I_w, then produces the self-inverting permutation π^* and finally returns the corresponding integer w.

The proposed WaterIMAGE system has the following design and functional advantages:

- it is an efficient image watermarking system; the experimental results showed that the watermark w is "well hidden" in the image I_w,
- its embedding method incorporates properties that allow us to successfully extract the watermark w for the image I_w even if the image I_w has been compressed with a lossy method and/or rotated,
- it is a simple and easily implemented system, and
- finally, as far as the time and space needed for the encoding/decoding process, it performs very well.

We should point out that the main feature of our WaterIMAGE system is the fact that it uses a combinatorial object to watermark an image; we show that an integer can be efficiently represented by a self-inverting permutation which, in turn, can be represented in the 2-dimensional space and, thus, this representation forms a suitable watermarking object for images. In our system we propose a marking method but apart from that the investigation of alternative and more efficient methods for marking an image using a 2D representation of a permutation are an open problem for further research.

It is fair to mention that, although our system is now in a fully operational stage, we believe that a friendly graphical user interface should support our system making it accessible to users from various specialization levels; we have to design and integrate such a feature in our system (Sommerville, 2010; Gamma et al., 1995; Taylor et al., 2009).

Furthermore, since the WaterIMAGE system is time and space efficient, it might be extended so that it will be able to take a watermark w as an input and search between targeted websites and find if they use watermarked images I_w. So, that system would be something like a search engine which searches in web for images I_w from a specific owner; we leave such a system's extension as a topic for further investigation.

REFERENCES

Ahmed, N., Natarajan, T., and Rao, K. R. (1974). Discrete cosine transform. *IEEE Transactions on Computers*, C-23:90 – 93.

Chun-Shien, L., Shih-Kun, H., Chwen-Jye, S., and Hong-Yuan, M. L. (2000). Cocktail watermarking for digital image protection. *IEEE Transactions on Multimedia*, 2:209 – 224.

Collberg, C. and Nagra, J. (2010). *Surreptitious Software*. Addison-Wesley.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, C-23:297 – 301.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press, 2nd edition.

Cox, I., Kilian, J., Leighton, T., and Shamoon, T. (1996). A secure, robust watermark for multimedia. In *Proc. 1st Int'l Workshop on Information Hiding*, volume LNCS 1174, pages 317 – 333.

Davis, J. C. (1997). Intellectual property in cyberspace - what technological / legislative tools are necessary for building a sturdy global information infrastructure? In *IEEE Int'l Symposium on Technology and Society*, pages 66 – 74.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Garfinkel, S. (2001). *Web Security, Privacy and Commerce*. O'Reilly, 2nd edition.

Golumbic, M. (1980). *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, Inc., New York.

Gonzalez, R. C. and Woods, R. E. (2007). *Digital Image Processing*. Prentice-Hall, 3rd edition.

Gonzalez, R. C., Woods, R. E., and Eddins, S. L. (2003). *Digital Image Processing using Matlab*. Prentice-Hall.

Grover, D. (1997). *The Protection of Computer Software - Its Technology and Applications*. Cambridge University Press, New York.

Jain, K., Raghavan, M., and Jha, S. K. (2009). Study of the linkages between innovation and intellectual property. In *Proc. of PICMET 2009*, pages 1945 – 1953.

Lemley, M. A. (2005). Intellectual property, and free riding. *Texas Law Review*, 83: 1031:1033.

O'Ruanaidh, J., Dowling, W., and Boland, F. (1996). Watermarking digital images for copyright protection. *Vision, Image and Signal Processing, IEE Proceedings*, 143(4):250 – 256.

Pascale, D. (2003). *A Review of RGB Color Spaces ...from xyY to R'G'B'*. The BabelColor Company.

Raysman, R., Pisacreta, E. A., and Adler, K. A. (1999). *Intellectual Property Licensing: Forms and Analysis*. Law Journal Press.

Sedgewick, R. and Flajolet, P. (1996). *An Introduction to the Analysis of Algorithms*. Addison-Wesley.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley, 9th edition.

Tamada, H., Nakamura, M., and Monden, A. (2004). Design and evaluation of birthmarks for detecting theft of java programs. In *Proc. Int'l Conference on Software Engineering (SE'04)*, pages 569 – 575.

Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley.