國立交通大學

資訊科學與工程研究所

碩 士 論 文

利用多數同位檢查降低像素修改量之資訊隱藏

Reduction of Pixel Modification in Data Hiding With Majority Parity Check

研 究 生：侯政利

指導教授：曾文貴　教授

中 華 民 國　九 十 八 年 六 月

利用「多數同位檢查」降低像塑修改量之資訊隱藏

學生：侯政利　　　　　　　　　　　　　　指導教授：曾文貴　博士

國立交通大學資訊工程學系

摘要

資訊隱藏是資訊安全領域裡，一個大家所熟知的領域，而如何在不被發覺的情形下將一段資料隱藏到一個檔案中，一直是資訊隱藏裡最重要的議題。不幸的是，在隱藏資訊的過程中，勢必需要對檔案做一些更改，這些更改會造成圖形上的失真，使得被隱藏的資訊容易被發現。所以我們提出了一個方法，能有效的降低在隱藏過程中所需要做的修改量，利用樹狀結構與多數同位檢查來有效的降低資訊修改量，以隱藏一段資訊。因此本研究方法是一個可以用來幫助現有的大多數資訊隱藏方法的線性嵌入程序。

關鍵字：　資訊隱藏、多數同位檢查

Reduction of Pixel Modification in Data Hiding with Majority Parity Check

Student: Chung-Li Hou Advisor: Dr. Wen-Guey Tzeng

Department of Computer Science

National Chiao Tung University

Abstract

Steganography is a well-known topic in Information Security. In Steganography, how to embed a data into host file without being detected is the most important issue. Unfortunately, embedding process will make some modification in host file. For a digital image, pixel modification makes distortion and more distortion makes the embedded data be detected easily. So, we propose a scheme that can reduce pixel modification efficiently to decrease distortion caused by embedding process. By a tree structure and Majority Parity Check(MPC), to reduce pixel modification when hiding a data. Our scheme is a plug-in linear process to help most existing data hiding algorithms.

Keywords: Data Hiding, Steganography, Majority Parity Check

誌　　　謝

今天能順利的完成論文，最需要感謝的人就是曾文貴教授，感謝教授對於我的指導以及包容，認真的教學令我在密碼學這個領域中，能在短短的一年內學到了許多，並且順利的完成這篇論文。感謝三位口試委員，清大孫宏民教授及交大蔡錫鈞教授、黃育綸教授對於我的論文所給予的建議以及指導，啓發了我對於資訊隱藏這個領域裡更多的想法。除此之外，便是要感謝實驗室裡的學長姐以及同儕所給予我的幫助以及指導。

最後我要感謝的是我的家人，是他們在我的背後給我與精神的鼓勵，提供了我無後顧之憂的生活，讓我能專心的完成我的論文，順利的畢業，感謝家人的包容以及對我的鞭策。文字無法描述我心中對於所有給予我幫助的人的感謝，僅此，以短短的字句獻給他們，我心中的感謝。

Contents

List of Figures

Chapter 1

Introduction

In this chapter, we introduce the motivation of our research, and we give an overview about data-hiding and previous works.

1.1 Motivation

In this paper, we describe an algorithm that helps most data-hiding algorithms to reduce pixel modification. In recent years, there were many papers about data hiding. Most of them discussed about how to hide some information into some kind of objects. Especially, when we try to hide data into a picture file, there are two kinds of argument, reversible and un-reversible data-hiding algorithm.

TBPC[1] is the first paper that discussed about reduction of pixel modification, but there is still some problems in TBPC. The algorithm in TBPC, it can not efficiently reduce pixel modification when we hide too much information into a digital image. The rate of reduction for TBPC, it is not a stable one. So our purpose in this paper is to proposed an efficient and stable algorithm to reduce the rate of pixel modification.

1.2 Overview

Steganography is an issue that discuss how to hide information into a host multimedia. A digital image, an mp3 music file, or a movie file can be as a host file for data hiding algorithms. In steganography, the most issue is how to hide information without being detected. Somehow, digital image files are the most convenient file for hiding information.

Watermark is the most well-known topic. In a watermarking algorithm, a digital logo is embedded into a digital image file to prove the ownership.[2-4] are watermarking algorithms for embedding a logo image into a vector-quantized image, and [5-7] are algorithms for JEPG files. Some kinds of algorithm are lossy. During embedding information into host image by lossy watermarking algorithms, distortion is introduced and result in Peak Signal-to-Noise Ratio(PNSR) loss, but permanent loss of signal is not allowed in military or medical images. In lossy data hiding algorithms, we discussed how to reduce distortion during embedding information in the past. Besides, lossless/Reversible data hiding algorithms [4][23] can recover the original host image perfectly after the watermark is extracted, but the disadvantage in lossless/reversible data hiding algorithms are that we just can hide much lesser information into host image than lossy ones.

TBPC[1] is the first paper which presented this idea that we can reduce distortion by decreasing pixel modification. In data hiding algorithm, each pixel can be treated as an information-bit, "0" or "1". If information-bit was not the same as logo-bit, pixel modification would come up, and then distortion was occurred. TBPC[1] presents a tree based data-coding method with parity check to reduce pixel modification with existing data hiding algorithms.

1.3 Synopsis

We organized the rest of this thesis as following.

In Chapter 2, we first talk about the relative background: Lossy/Lossless data hiding algorithms, Stego-codes, and Tree Based Parity Check (TBPC). We discuss some kinds of lossy data-hiding algorithms in several image format and a data coding theorem named Stego-codes. Finally, TBPC algorithm will be discussed in this chapter. In Chapter 3, we propose our new reduction of pixel modification scheme, majority parity check(MPC). In Chapter 4, we provide the analysis for our scheme that is a linear Stego-code model, and analyze the efficiency of our scheme. In Chapter 5, simulation of MPC is performed, and we compare the result of MPC with TBPC. Finally, in Chapter 6, we make a conclusion and discuss the future work.

Chapter 2

Background

In this chapter, we introduce lossy/lossless data-hiding algorithms. We discuss several exiting schemes in watermarking domain. Then, we introduce Stego-codes, a coding problem in steganography[12]. At last, we introduce However TBPC algorithm work and help data-hiding algorithms to reduce pixel modification.

2.1 Steganography

Steganography is the art and science of hiding information by embedding messages within others, seemingly harmless messages. Steganography means "covert writing" in Greek. A famous illustration of steganography is Simmons' "Prisoners' Problem" [24], so as the goal of steganography is to create a covert channel in common object, and to hide information without being detected. The major difference between steganography and cryptography is that cryptography obscures hidden messages by encryption and steganography aims at concealing the existence of hidden information.

Nowadays, there are many steganographic tools to hide information into many kinds of objects. We focus on digital images in our paper. In the past, many schemes use uncompressed images as cover objects. Due to the convenience of transmission and storage, compressed images are popularly discussed in recently years. For Both uncompressed and compressed images, a digital image can be seen as an array of pixels. To hide information into a digital image, pixel modification must be handled. But in most steganography algorithms, the original pixel value can not be recovered after modification. This kinds of algorithms, we call them, "Lossy Data Hiding". On the other hand, if the algorithms that can recover the original pixel value after extracting information, we call them "Lossless Data Hiding".

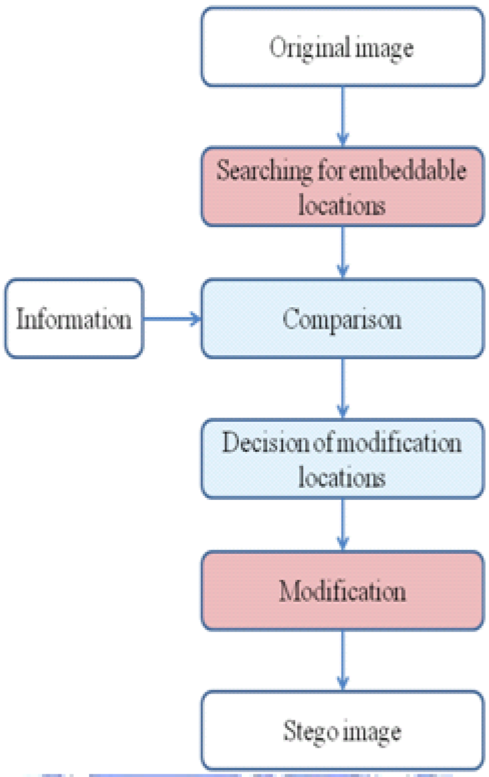

Original image
Searching for embeddable locations
Information
Comparison
Decision of modification locations
Modification
Stego image

Figure 2.1: The Common Flow Diagram of Steganography

2.1.1 Common Framework

In most data hiding algorithms, the common flows showed in figure 2.1 involve the following four steps, when we try to hide information into a host-image. The first step is to search embeddable locations in host-image. For different data hiding algorithm, different amount of embeddable locations will be decided and distinguished the value for each embeddable location as ”0” or ”1”. The second step is to compare the value of embeddable location with information that needs to be embedded and the third step is to make decision on which embeddable location needs modification to hold information bit. The last step is to modify locations which are decided in the third step and obtain the final stego-image. There are many kinds of data hiding algorithms, but these four steps are the common flows in steganography. Most papers discuss how to search embeddable locations and how to modify to hold information bit with few visual artifacts. The common flow diagram could be used in un-reversible or reversible data hiding algorithms.

2.1.2 Relative Works

With developing in steganograpgy, several different kinds of domain have been discussed extensively to hide data into different image file format. Last significant bit is one of the simplest method and used for un-compressed image. Those algorithms with LSB have been detected [16-18] in recent years. As the technique of image compressing being used widely, like vector quantization and JPEG, [5-7] are proposed for VQ based images and [3][19-22] are proposed for JPEG images. For these existing algorithms, all of them are lossy. To avoid distortion in embedding process, reversible data hiding algorithms [4][23] are proposed which can perfectly recover the original host image after extraction. But reversible data hiding algorithm can not embed a long data into a host image.

2.2 Stego-codes

Steganography is the scheme to communicate hidden messages between the sender and the receiver such that no other people can detect the existence of the message. A common strategy for steganography is to embed the message by slightly distorting the cover object to the stego object. If the distortion is small enough, the stego object will be undistinguishable from the noised cover object.

2.2.1 Matrix Embedding

Matrix Embedding[9,10], also called Syndrome Coding[11], or Coset Encoding [8], is a steganography method using (n, k) linear codes. In advance the sender and the receiver agree an $n \times (n-k)$ parity check matrix H of an (n, k) linear code. The cover object is represented as a vector $n \in F_2^n$ (e.g. for an image, take the least significant bits of all pixels) and the message is a binary vector M with length $(n-k)$. When embedding, the sender identifies a vector $x' \in F_2^n$ such that $Hx' = M$. When extracting, the receiver extracts the hidden message M from the stego object x' by computing $Hx' = M$. Let $\delta = x' - x$ be the distortion between the cover object x and the stego object x'. The Hamming weight of δ, denoted by $wt(\delta)$, is generally the measurement of quality of x'. The sender will always choose the lowest weight of δ of to ensure the smallest distortion on the cover object. Therefore, the sender should resolve that $H\delta = M - Hx$ such that $wt(\delta)$ is minimum. The set of

δ satisfying $H\delta = M - Hx$ is the coset of the linear code with respect to the parity check matrix H. Finding the lowest weight of δ is the well known coset leader problem.

2.2.2 Steganography Codes

Matrix embedding embeds and extracts messages by using a parity check matrix H. W. Zhang and S. Li[12] generalized the idea of matrix embedding and define the codes with the matrix H as steganography codes(abbreviated stego-codes). An $(n-k) \times n$ matrix H over $GF(n)$ is called an $(n, n-k, t)$ stego-coding matrix if for any given $y \in GF^{(}n-k)(q)$, there exists an $v \in GF^n(q)$ such that $wt(v) \leq t$ and $Hv = y$. In comparison with matrix embedding, v is the distortion and y is equivalent to $M - Hx$ where M is the message and x is the cover object. Let $S_y = v : Hv = y$. An $(n, n-k, t)$ linear stego-code is defined by $S = S_y : y \in GF^{(}n-k)(q)andS_y \neq$.
The steganography problem is that for any given message $M \in F^{(}n-k)_2$ and any given cover object $x \in F_2^n$, find the vector $v \in F_2^n$ such that $wt(v)$ is minimum and $H(x+v) = y$. Applying an $(n, n-k, t)$ linear stego-code, the steganography method can guarantee that the distortion for any given message and cover object is at most t bits. The coset leader problem is equivalent to the nearest codeword problem(NCP) for binary linear codes. NCP is the problem that given a $k \times n$ matrix A over $GF(2)$ and a vector $y \in GF^n(2)$, find a codeword c such that $wt(y-c)$ is minimum. For the

nearest-codeword decoding in coding theory, finding the nearest codeword for a given vector y is to identify the lowest weight vector(i.e. the coset leader) in the coset containing y as the error pattern of y[15]. NCP has been proved that Approximating NCP within any constant factor is NP-Hard[13]. In some special cases (e.g. F5[14]), a constructive method can efficiently obtain a proper solution.

2.3 Tree Based Parity Check

[1] is the first paper presents a data-coding mapping to reduce pixel modification. It is a plug-in process for exiting data hiding algorithms. TBPC algorithm increases three steps into common flow that we discussed in 2.1.1. they are namely "Tree Formation", "Parity Calculation", and "Fountain Investigation". The flow diagram of data hiding process with TBPC is shown in Figure 2.2.

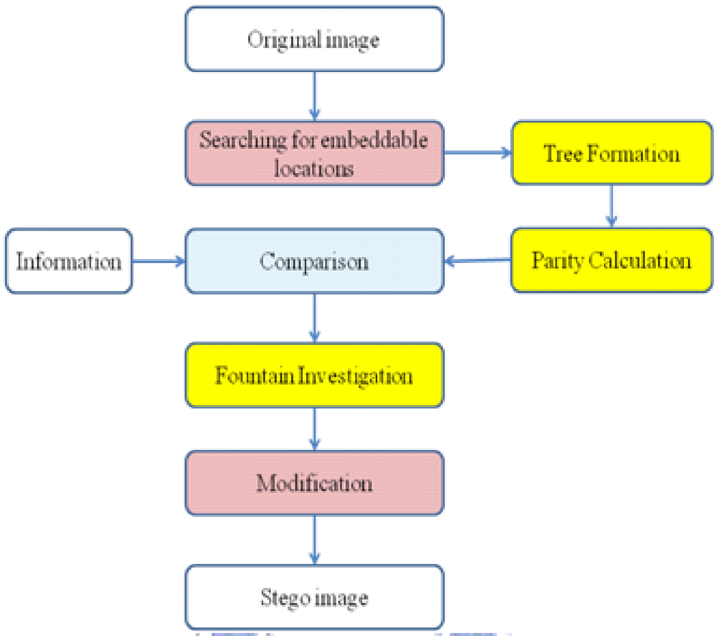


Figure 2.2: Flow Diagram of TBPC

In Tree Based Parity Check(TBPC) algorithm, the main idea is to reduce pixel modification by a tree structure. When we try to hide information into a host image, we construct a "Master Tree" which is a N-ary tree according to length of information L and the number of embeddable locations M in host image. Each node in Master Tree is set as an information bit in host image that decided by some data hiding algorithm such as LSB. The relationship between L, M and N is as bellow.

We assume that we have to construct an x-level N-ary Master Tree. So the number of nodes we need in Master tree is as following equation.

$$nNodes = 1 + N + N^1 + N^2 + \ldots + N^x, x = log_N L$$

$$nNodes = \sum_{i=0}^{x} N^i = \frac{NL - 1}{N - 1}, L = N^x$$

The number of embeddable locations M must be larger than nNodes, so we can obtain the following relationship.

$$
\begin{aligned}
M &\geq \frac{NL-1}{N-1} \\
\Rightarrow\ MN - NL &\geq M - 1 \\
\Rightarrow N &\geq \lceil \frac{M-1}{M-L} \rceil
\end{aligned}
\tag{2.1}
$$

From equation (2.1), we can choose a suitable N to construct the Master Tree to embedding a L-bits data into a host image with M embeddable locations. In Parity Calculation, we can get an "Info' array by calculating parity for each leaf in Master Tree. In this step, we calculate the number of "1" from root to each leaf. If the number of "1" is odd, the information bit of leaf node is set as "1". Otherwise, the information bit of leaf node is set as "0". After Parity Calculation, we can get "Toggle Array" by performing bitwise logical exclusive-OR(XOR) operation between information and Info array that we got in Parity Calculation step. Every "1" in "Toggle Array" means to need one time of pixel modification. In Fountain Investigation, we construct a Toggle that the structure is the same as Master Tree and leafs is filled up with "Toggle Array". From bottom to top, if all the children are "1", we make a change from "1" to "0" for all children and set the parent as "1". After Fountain Investigation, we can obtain the result Toggle Tree. Each node in Toggle Tree that contains information bit "1" means to need a pixel modification on interrelated location in Master Tree. By data hiding algorithm, like LSB, we modify "0" to "1" or "1" to "0" according to Toggle Tree, and we can obtain the final stego image. When receiver gets the stego image from sender, because of keeping the same key for both sides, receiver can reconstruct the same structure of Master Tree. By parity calculating the Master Tree constructed from stego image, receiver can get the information sent by sender.

TBPC is the first paper that presents this idea, but it can not work efficiently when there is a long information need to be embedded. The Master tree is a N-ary tree and the value of N is according to L and M.Thus the larger L makes larger N, as a result, when L is larger, TBPC is less efficiency. This is the most important problem that we try to solve in our scheme.

Chapter 3

The Proposed Scheme

In this chapter, we detail to describe our scheme. Our scheme has the same flow diagram as TBPC. We will describe how the MPC model works in each step. There are three steps in out model that is named "Tree Formulation", "Parity Check", and "Fountain Investigation". Afterword we will use LSB data hiding algorithm as a example to plug in.

3.1 Tree Formulation

When we try to hide information into a host image, we can generate a location sequence by a key that is held on sender and receiver. By data hiding algorithms, each location could be seen as information bit, "0" or "1". For example, information bit is "1" when the pixel value is odd in LSB algorithm, and on the other hand, the information bit is "0". If we try t o hide L-bits information into a M-pixel image, we have M embeddable locations by LSB algorithm. By equation (2.1), a suitable N could be chosen and we construct a N-ary tree named "Master Tree". We fill up each node from top to bottom by the bit information of embeddable locations sequence. Here is a simple example.

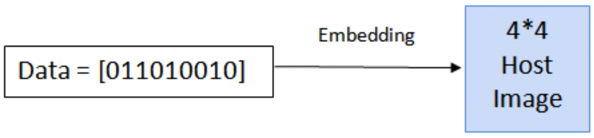
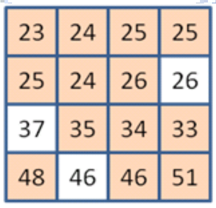


Figure 3.1: Embedding Diagram

By equation (1), we choose N is equal to 3 and the tree-level x is 2. So, we need 13 embeddable locations to construct Master Tree. Supposedly, an embeddable sequence is chosen and the information bit is [1011100101001]. The 2-level 3-ary Master Tree is constructed as bellow.



23	24	25	25
25	24	26	26
37	35	34	33
48	46	46	51

Figure 3.2: The Chosen Embeddable Locations

As we obtain the embeddable sequence, we fill the Master Tree up with embeddable sequence in order. It is shown as following.

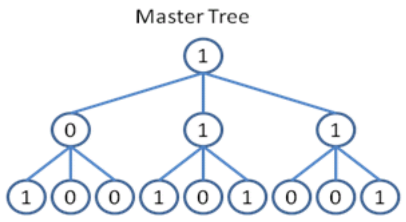


Figure 3.3: Example of Master Tree

3.2 Parity Check

After we construct Master tree, we calculate information bit for each leaf in Master Tree. We visit each leaf up to root, and count the number of "1" for each node in this path. If the number of "1" is odd, the information bit is "1", and on the other hand, the information bit is "0". After this step, we get a "Info Array" which length is L for Master Tree. A simple example is as bellow.

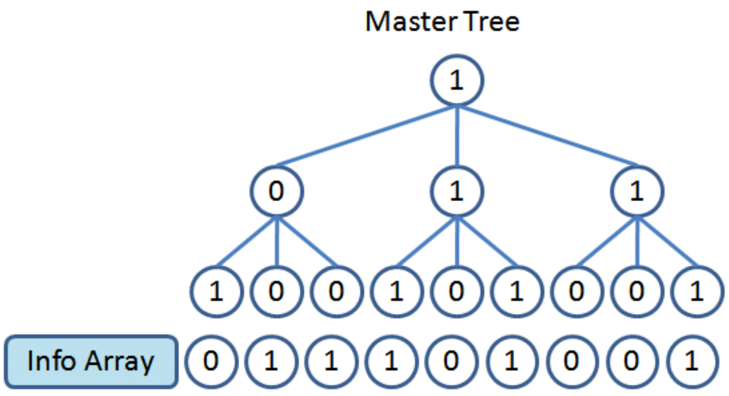


Figure 3.4: Result of Parity Check

3.3 Fountain Investigation

As we obtain Info Array and Data, we execute bitwise Exclusive-OR logical operation between Info Array carried by Master and Data, and we obtain a Toggle Array. As an example showing in (), the Info Array carried by Master Tree is [011101001], and Data is [110110111]. We obtain a Toggle Array which is [101011110] by performing bitwise Exclusive-OR operation.

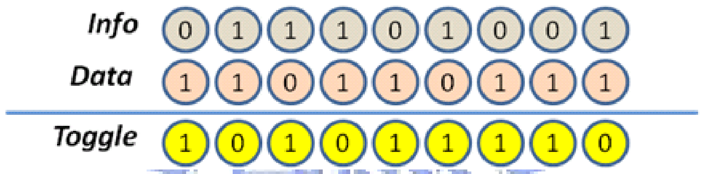


Figure 3.5: Toggle Array

In Toggle Array, each "1" represents as a toggle that is needed one pixel modification in original image. However, more modification makes more distortion. Visual artifacts are introduced by any single modification in original image. To achieve better visual quality of stego-image, the number of "1" in Toggle array must be minimized.

Take an observation for Master Tree. We can find that a simple change in any node either from "0" to "1" or "1" to "0", will result in a change in the parity of its descendant leaf nodes. In other word, changing N sibling nodes is not as good as changing the parent node.

In TBPC, a Toggle Tree is built with the same structure as the Master Tree, and the leaf nodes are filled up by Toggle Array with the same order. The other nodes in Toggle Tree are set the value as "0". The Fountain Investigation process is started from bottom of the Toggle Tree. For any node with all N of its child nodes are with the value of "1", all the child node are updated with the value of "0" and the examined node is set to "1". Otherwise, the examined node has no change. After investigating all nodes in Toggle Tree from bottom leafs to root, we obtain a final Toggle Tree that represents the Toggle Array.

In our scheme, we lead majority into the original scheme of TBPC. We assume that N is $2k$ or $2k+1$ for N-ary Tree. For any node in Toggle Tree with t of its child nodes are with the value of "1", if t k, the examined node is set as "1" and all the child nodes are changed from "0" to "1" or from "1" to "0". By our scheme, we can obtain a better Toggle Tree that represents the same Toggle Array, and it means that we make less pixel modification in original image to hide the same data. The example being discussed is shown in Figure. 3.6 and 3.7.

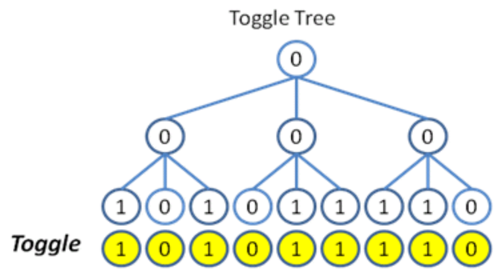


Figure 3.6: Initial of Toggle Tree

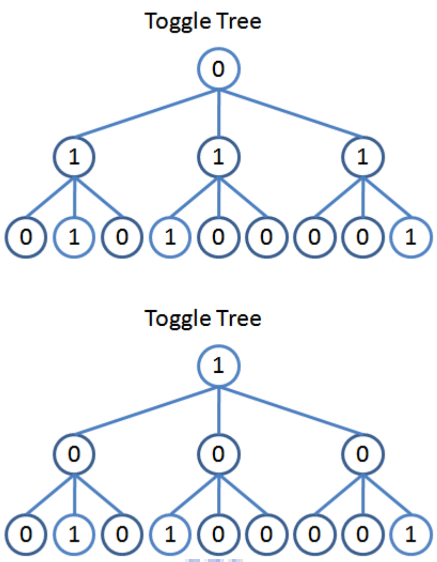


Figure 3.7: Steps of Fountain Investigation

Because this check is from bottom to top and process investigates the Toggle Tree like a fountain, we call this process "Fountain Investigation". After Fountain Investigation process, we can obtain a final Toggle tree and each node in Toggle Tree that contains information bit "1" means a pixel modification at this corresponding pixel on original image. After the pixel modification on host image, we obtain a stego-image that is embedded a secret message.

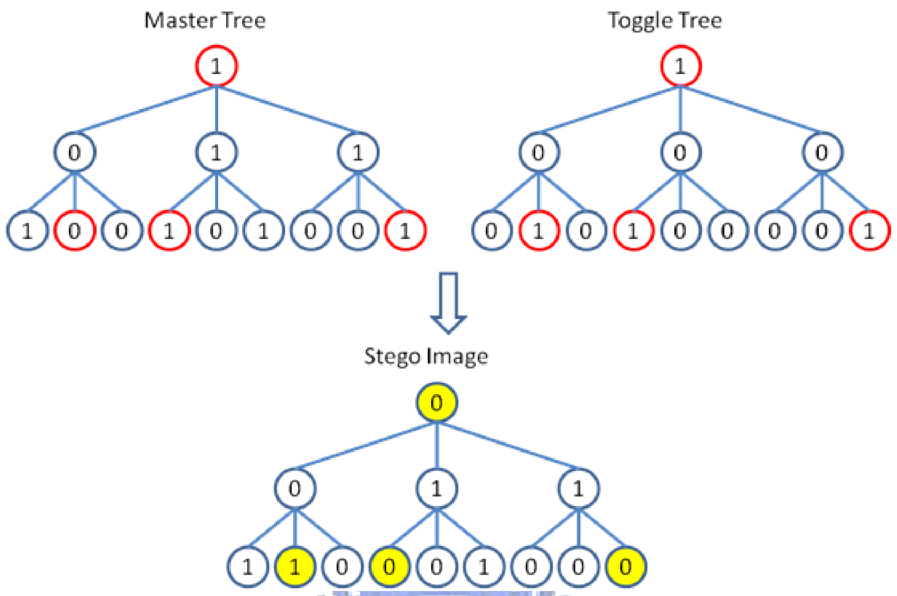
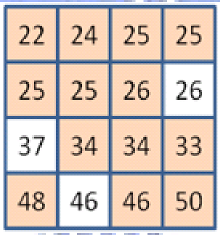


Figure 3.8: Master Tree after Modified



22	24	25	25
25	25	26	26
37	34	34	33
48	46	46	50

Figure 3.9: Final Stego-image

3.4 Extraction

Receiver receives a stego-image that is sent from sender, because they both have the same secret key and know the length of the secret message. Receiver can re-build a Master Tree with same structure as that is built by sender. Master Tree is filled up with the sequence that is generated by the key. After Party Calculation, the Information Array of Master Tree would be exactly the same as the information being embedded.

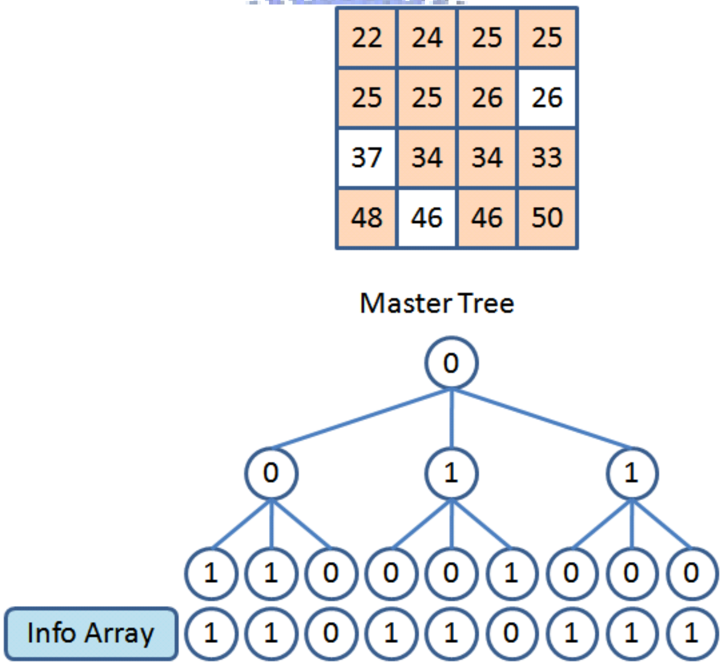


Figure 3.10: Step of Extraction

Chapter 4

Analysis

In this chapter, the performance of our scheme will be discussed. The main-concerns in data hiding process are maximum payload, visual quality and complexity. Section 4.1 describes the proof of Linear Stego-code. Section 4.2 shows the maximum payload of MPC as a function of N. Section 4.3 describes the accuracy of MPC. Section 4.4 shows the percentage of MPC saving. Section 4.5 shows the complexity of MPC.

4.1 Proof of Linear Stego-codes

In [12], W. Zhang and S. Li proposed linear *(n,k,t)* stego-code, There are several advantages in some model with the property of linear stego-code. In practically, we can use a generator to handle the mapping from $GF^n(q)$ to $GF^k(q)$ in very short time. Second, it can guarantee that the worst distortion is t, $wt(x - x') \leqslant t$. At the last, it is a effective method for embedding and extracting.

4.1.1 $(M, L, \frac{L}{2}) Stego - codes$

Definition 1 *An (n,k,t)Stego-code is satisfying the following condition. For any given $x \in GF^n(q)$ and $y \in GF^k(q)$, there exists a $x' \in GF^n(q)$ such that $wt(x - x')$ and $H(x') = y$.*

By Definition 1, we assume

$$n = M : Misthenumberofnodeinthetree \tag{4.1}$$

$$k = L : Listhenumberofleafnodes \tag{4.2}$$

Then, we map $GF^n(2)$ to $GF^k(2)$ by tree based MPC. In Fountain Investigation, we obtain a final Toggle Tree and the number of "1" in Toggle Tree means the quantity of modification. By observing the Toggle Tree, the number of "1" in parent and children must be less than or equivalent to half number due to MPC. A simple example is shown as following Figure.

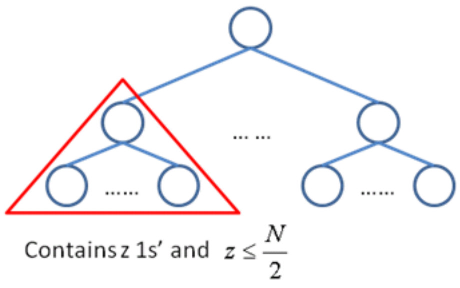


Figure 4.1: The Number of "1" in sub-tree

We can discover that the worst case is every sub-tree, between level 0 and level 1, with $\frac{N}{2}$ "1"s'. In this case, inverse process will never be processed. So we can defined $wt(x - x')$ as following equation.

$$\begin{aligned} wt(x-x') = t = \frac{N}{2} \cdot L(i-1) \\ = \frac{N^i}{2} \because L(i-1) = N^{i-1} \\ = \frac{L}{2} \because L = N^i \end{aligned} \tag{4.3}$$

By equation (4.1), (4.2) and (4.3), our scheme can be said as a $(M, L, \frac{L}{2})$ Stego-code.

4.1.2 Property of Linear

In [12],W. Zhang and S. Li defined linear stego-code as following.

Definition 2 *An (n,k,t)Linear Stego-code is satisfying the following condition. A $k \times n$ matrix H over $GF^n(q)$ is called an (n, k, t) Stego-coding matrix if for any given $y \in GF^k(q)$, there exists an $y \in GF^n(q)$ such that $wt(x) \leq t$ and $Hx^t r = y^t r$.*

We have proved that our scheme is a $(M, L, \frac{L}{2})$ Stego-code model in 4.1.1. If we could find or construct the generating matrix for our scheme, we can said our pro- posed scheme is a $(M, L, \frac{L}{2})$ linear Stego-code model.

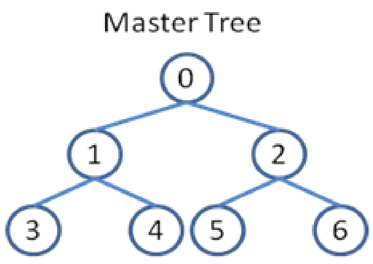


Figure 4.2: Example of Master Tree

In Figure 4.2, it is a simple example for a 2-level 2-ary tree. First of all, we label every node in ordering from root to leaf, and then we have to construct a 4×7 generating matrix. Each column is set as a relation according to the path from root to leaf in ordering. In this way, we can simply construct a generating matrix H for the tree shown as following matrix.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.3: Generating Matrix

If we assume x = [1,0,0,1,0,1,1], then we can obtain y as following equation.

$$Hx^t r = \begin{pmatrix} 1,1,0,1,0,0,0 \\ 1,1,0,0,1,0,0 \\ 1,0,1,0,0,1,0 \\ 1,0,1,0,0,0,1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = y^t r$$

4.2 Maximum Payload

By any data hiding algorithm, the number of embeddable locations is limited in a host image, so the size of Master Tree formed in Tree Formulation step is limited. In Tree Formulation step, we construct a x-level N-ary tree named Master Tree and the maximum payload of MPC is according to the degree N of Master Tree.

In the Master Tree, there are L leaf nodes. To form an N-ary complete tree with L leaf nodes, the total number of nodes, *nNodes*, can be found by the following equation.

$$\begin{aligned} nNodes &= 1 + N + N^1 + N^2 + \ldots + N^x, x = log_N L \\ &= \sum_{i=0}^{x} N^i = \frac{NL-1}{N-1}, \because N^x = L \\ &\approx (\frac{N}{n-1}L), \because NL \gg 1 \end{aligned} \tag{4.4}$$

From the equation (4.4), we can conclude that when N increase, the number of nodes in Master Tree decreases under the same L leaf nodes. In other words, the larger N is chosen, the higher payload can be embedded. The percentage of data that can be embedded in a host image, *pHidden*, can be found by the following equation.

$$pHidden = \frac{L}{\frac{NL}{N-1}} = \frac{N-1}{L} \tag{4.5}$$

For fixed data size L and the number of embeddable location M, the limit of N can be found by equation (4.6).

$$M \geq \frac{NL-1}{N-1} \Rightarrow MN - NL \geq M - 1 \Rightarrow N \geq \lceil \frac{M-1}{M-L} \rceil \qquad (4.6)$$

From equation (4.6), we can conclude that the minimum N should be chosen in Tree Formulation step and the maximum percentage of payload will be defined as equation (4.5).

4.3 Accuracy of MPC

In TBPC, Toggle Tree is formed in Fountain Investigation step. A simple majority checking enters into this step in MPC. This section will discuss that the Toggle Tree formed by MPC can work as the Toggle Tree formed by TBPC. Here is a simple example in TBPC with a *2*-ary tree and the Toggle Array is [11010010].

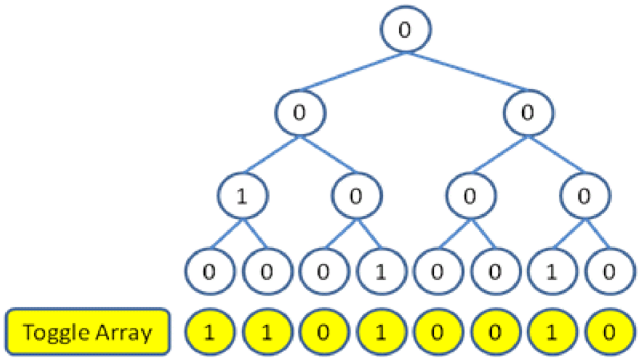


Figure 4.4: Toggle Tree

Take an observation on Toggle Tree in Figure 4.4. We can find that if the according toggle bit of leaf node is "1", there must be one node that contains "1" in the path from leaf to the root. In the other words, there is odd number of "1" in the path from leaf to the root if the toggle bit is "1". Otherwise, there is even number of "1" in the path. We have to ensure the concern of Toggle Tree when majority checking enters into.

Fountain Investigation step is a process that constructs Toggle Tree from bottom to root. In the process of constructing, we observe the sub tree in Toggle Tree and find the following conditions.

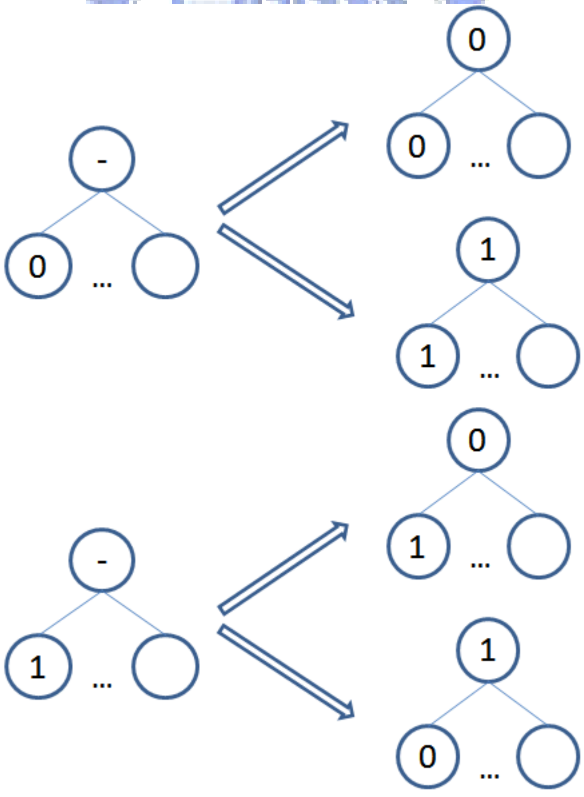
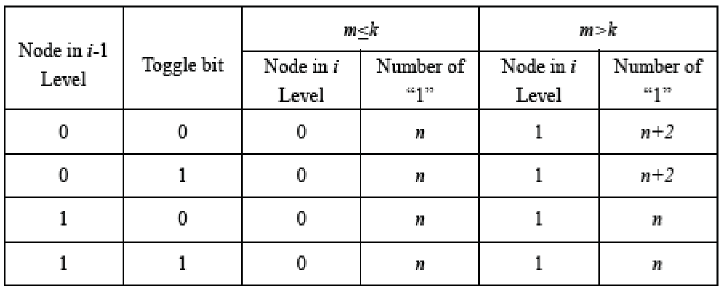


Figure 4.5: Change of Majority Checking

By induction, if the level of tree is 1, the number of ”1” in the path from leaf to root is according to toggle bit, odd for ”1” and even for ”0”. Assume that *i-1* level ary-tree is correct, there are k nodes contains ”1” in children, and the number of ”1” in *i-1* level path is n. We conclude the following table of these four conditions.



Node in *i*-1 Level	Toggle bit	$m \leq k$		$m > k$	
		Node in *i* Level	Number of “1”	Node in *i* Level	Number of “1”
0	0	0	n	1	$n+2$
0	1	0	n	1	$n+2$
1	0	0	n	1	n
1	1	0	n	1	n

Figure 4.6: Truth Table of MPC

In this table, if the occurrence of ”1” in *i-1* level path is odd, the occurrence of ”1” in i level path will be odd. Otherwise, the occurrence of ”1” will be even. So we can ensure the accuracy of MPC.

4.4 Percentage of MPC Saving

In most existing data hiding algorithms, the expected percentage of toggling the value in an embeddable location is 50%. In other words, if we try to embed L-bits data into host image, $0.5L$ times of modifications will occur in process. The percentage of modification, *pToggle*, is defined as the following equation.

$$pToggle = \frac{numbers\ of\ modification}{length\ of\ data} \tag{4.7}$$

In this section, we consider *pToggle* and Toggle Tree with the help of MPC. First, we give some definitions to be used in the calculation of *pToggle* with i-levels N-ary complete tree. An i-level tree consists of one root and N $(i-1)$-level trees. For an i-level tree, the number of leaf nodes is defined as $L(i)$. In Fountain Investigation, each node of leafs is "0" or "1" according to Toggle Array before investigating the Toggle Tree. The total different combinations of leaf nodes is defined as $C(i)$. The properties of $L(i)$ and $C(i)$ are shown in equation (4.8)-(4.11).

$$L(i) = N^i \tag{4.8}$$

$$L(i) - NL(i-1) \tag{4.9}$$

$$C(i) = 2^{L(i)} \tag{4.10}$$

$$C(i) = [C(i-1)]^N \tag{4.11}$$

Because of Fountain Investigation, we define leaf nodes are *0*-th level and the root is i-th level. For an i-level tree, we define $R(i)$ as the total modifications can be reduced in whole combinations from $(i-1)$-th level to i-th level with MPC. In other words, from $(i-1)$-th level to i-th level, the total number of "1" can be reduced in all combinations. Here is a simple example shown in Figure 4.7.

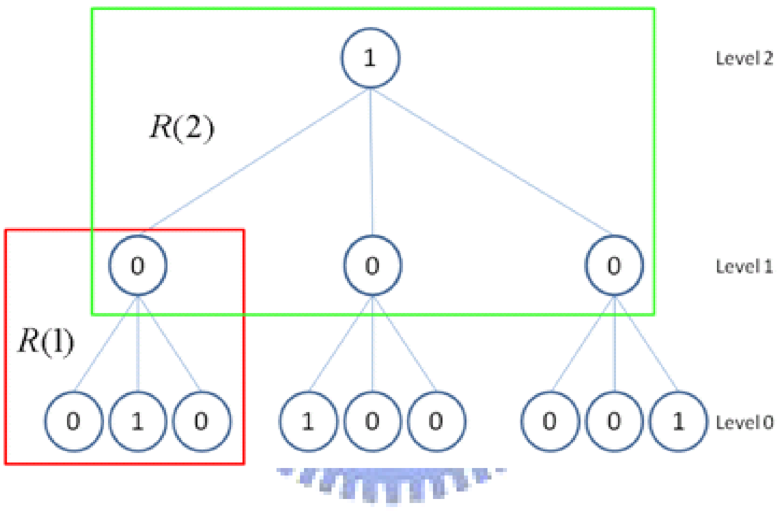


Figure 4.7: Example of $R(i)$

For data hiding algorithms without MPC, *pToggle* equals to 0.5. For an i-level N-ary tree, by MPC, the percentage of toggle saving from $(i-1)$-th level to i-th level can be defined as the following equation.

$$\frac{R(i)}{C(i)L(i)} \tag{4.12}$$

Considering with Fountain Investigation, we visit the Toggle Tree from bottom to root. The percentage of modification with MPC for i-level N-ary complete tree is defined as following equation.

$$pToggle = 0.5 - \frac{R(1)}{C(1)L(1)} - \frac{R(2)}{C(2)L(2)} - \ldots - \frac{R(i)}{C(i)L(i)} \tag{4.13}$$

Before we consider *pToggle*, we give some definitions to be used in the calculation of $R(i)$. In Fountain Investigation, we construct the Toggle Tree level by level, from bottom to root, so we can consider the following equations that have recursive relation between parent and children. First, we assume that N equals to *2k* or *2k+1*. We define D(t) as the total number of "1" that can be reduced when there are t "1" in children. If there are more than half children containing the value of "1", the process will inverse the parent and children. In the other words, when the number of "1" in parent and children is more than k, the number of "1" that can be reduced is defined as $R(t)$ as inverse is performing. The total number of combinations that can make root equals to "1" in i-level N-ary complete tree is defined as $P(i)$. The properties of $D(t)$ and $P(i)$ are shown in equation (4.14) and (4.15).

$$D(t) = t - [(N+1) - t] = 2t - N - 1 \tag{4.14}$$

$$P(j) = \sum_{j=k+1}^{N} C_j^N \cdot P(i-1)^j \cdot [C(i-1) - P(i-1)]^{N-j} \tag{4.15}$$

In equation (4.14), there are t children containing value of "1" and $(N+1)$ nodes. When t is more than k, inverse process will be performed. It will decrease t "1" but increase $[(N+1) - t]$ "1" in Toggle Tree, and make root contains the value "1". Each child of root is an $i-1$ level sub-tree. To calculate the number of combinations that the value of root is "1", we have to sum up the whole conditions from $j = k + 1$ to $j = N$.

To consider $R(i)$ in i-level N-ary complete tree, each condition that j is more than k makes Toggle Tree save $D(j)$ "1". So $R(i)$ can be defined as multiplication of $D(t)$ and $P(i)$, and shown in the following equation.

$$R(i) = \sum_{j=k+1}^{N} D(j) \cdot C_j^N \cdot P(i-1)^j \cdot [C(i-1) - P(i-1)]^{N-j} \tag{4.16}$$

By equation (4.8)-(4.16), we can calculate the average *pToggle* in whole combinations of Toggle Tree. There is a theoretic result in Chapter 5 for variable N.

If N is an odd number, $2k + 1$, equation (4.16) could be simplified. By observing $P(i)$, we can find the following relation.

$$\sum_{j=0}^{k} C_j^N = \sum_{j=k+1}^{N} C_j^N \tag{4.17}$$

This discovery of $P(i)$ makes every node in Toggle Tree have a property that the probability of containing value "1" is equal to the probability of containing value "0". For an i-level N-ary complete tree, $P(i)$ can be simplified as following equation.

$$P(i) = \frac{1}{2} \cdot 2^{L(i)} = 2^{L(i)-1} \tag{4.18}$$

Let (4.18) substitute for $P(i)$ in equation (4.16), and $R(i)$ can be simplified as Following equation.

$$R(i) = \sum_{j=k+1}^{N} D(j) \cdot C_j^N \cdot P(i-1)^j \cdot [C(i-1) - P(i-1)]^{N-j} \tag{4.19}$$

$$\begin{aligned} R(i) &= P(i-1)^N \cdot \sum_{j=k+1}^{N} C_j^N \cdot D(j) \\ &= R(1) \cdot P(i-1)^N \end{aligned} \tag{4.20}$$

Finally, let (4.22) substitute $R(i)$ in equation (4.13) and simplify as following equation.

$$\begin{aligned} pToggle &= 0.5 - \frac{R(1)}{C(1)L(1)} - \frac{R(2)}{C(2)L(2)} - \ldots - \frac{R(i)}{C(i)L(i)} \\ &= 0.5 - \frac{R(i)}{2^N}(\frac{1}{N} + \frac{1}{N^2} + \frac{1}{N^3 \ldots}) \\ &= 0.5 - \frac{R(1)}{2^N \cdot (N-1)} \end{aligned}$$

In TBPC model, *pToggle* is a fast converging variable, so it can not work efficiently when N is more than "4". In our scheme, majority checking extends the length of path that information contained by lead node could be past down in Fountain Investigation step. Theoretically, we prove that MPC is more efficient than TBPC on *pToggle*.

4.5 Complexity

MPC and TBPC both work as a plug-in function for other existing data hiding algorithm, so the complexity of them should be low. Thus, we compare the complexity of MPC with TBPC. Tree Formulation step constructs a Master Tree that is based on length of data L. It is reasonable to assume the L is much larger than N, so the complexity of Tree Formulation can be said to be $O(L)$ for MPC and TBPC. In Parity Calculation step, we calculate Info Array that is hidden in leaf nodes, so the complexity of Parity Calculation can be said to be $O(L\,log_N L)$. In TBPC[1], they presented a method to make the complexity of Parity Calculation be $O(L)$, and it also suits to our proposed scheme.

The total number of nodes in Toggle Tree is less than $2L$. In Fountain Investigation step, the worst case is that inverse process needs to be performed in every majority checking, so the times of external nodes in the tree being visited is twice and the times of internal nodes is thrice. Thus, we can say the complexity of Fountain Investigation is $O(L)$.

Chapter 5

Experimental Results

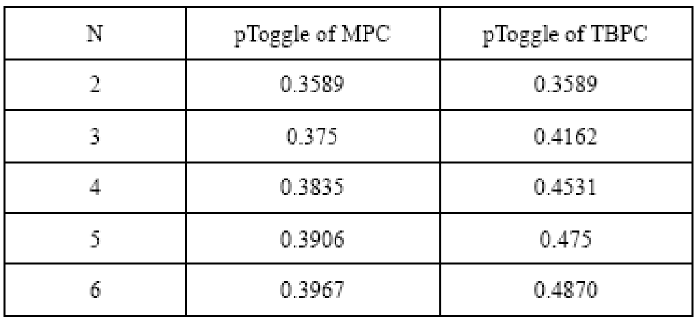
In this chapter, we try to verify the calculation shown in Chapter 4. Theoretically, MPC is more efficient than TBPC, so a set of experiment is carried out in this chapter. In 5.1, we give a theoretical comparison between MPC and TBPC. In 5.2, two hundred random sequences are generated as Toggle Array, and using different values of N to compare the efficiency between MPC and TBPC.

5.1 pToggle

In TBPC[1], they defined the pToggle of their scheme as following equation.

$$pToggle(i) = pToggle(i-1) - \frac{N-1}{C(i)L(i)} \tag{5.1}$$

We compare the result between MPC and TBPC and it is shown in the following Table.



N	pToggle of MPC	pToggle of TBPC
2	0.3589	0.3589
3	0.375	0.4162
4	0.3835	0.4531
5	0.3906	0.475
6	0.3967	0.4870

Figure 5.1: Table of pToggle Comparison

In Figure (5.1), we choose different N and compare the theoretical result. We can easy find that if N is more than 6, TBPC works un-efficiently. Even we can say that TBPC is no help if N is more than 8. It is shown in the following Figure (5.2).

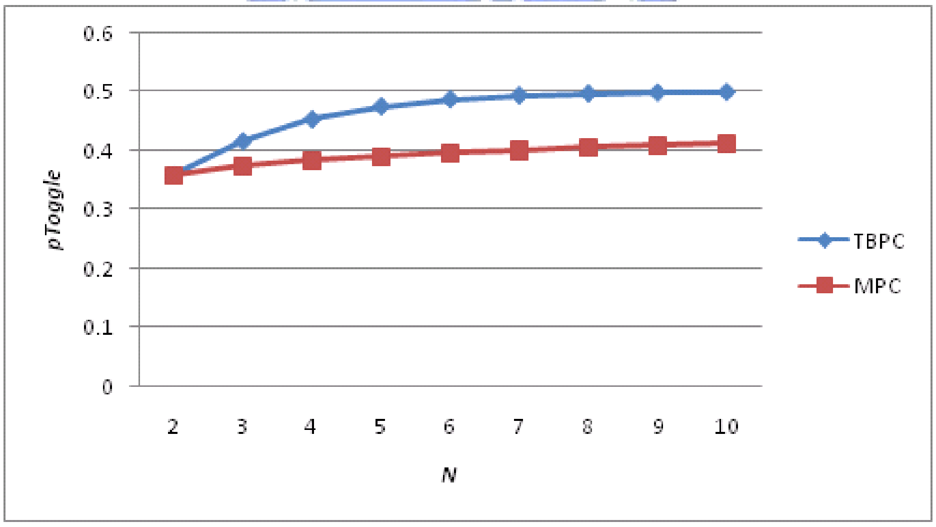
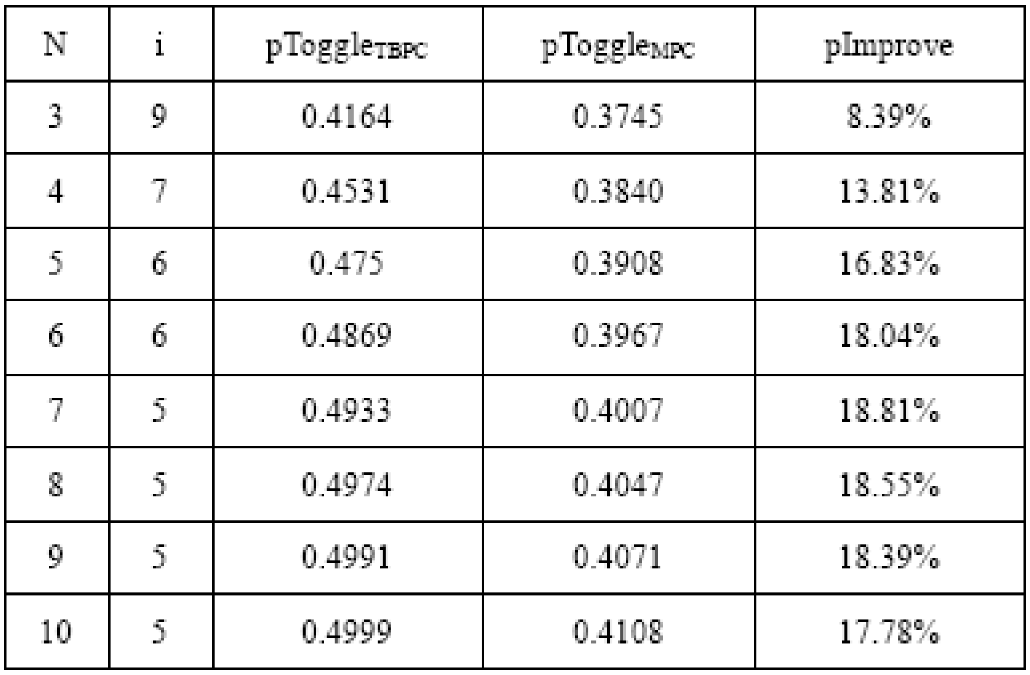


Figure 5.2: Comparison of pToggle

5.2 Analysis of Experimental Results

The efficiency of MPC and TBPC is distinguished by structure of Toggle Tree. So we choose random N and i to construct different size of Toggle Tree and let the number of leaf node is more than 15000. In each case, N and i, we carry out 200 times and the result is shown in following Figure (5.3).



N	i	$pToggle_{TBPC}$	$pToggle_{MPC}$	pImprove
3	9	0.4164	0.3745	8.39%
4	7	0.4531	0.3840	13.81%
5	6	0.475	0.3908	16.83%
6	6	0.4869	0.3967	18.04%
7	5	0.4933	0.4007	18.81%
8	5	0.4974	0.4047	18.55%
9	5	0.4991	0.4071	18.39%
10	5	0.4999	0.4108	17.78%

Figure 5.3: Table of Experimental Result

In the following Figure (5.4) and (5.5), we show the experimental result as a broken line graph, and the percentage of modification reduction is defined as *pReduce* and shown as following equation.

$$pReduce = \frac{the\ number\ of\ 1\ being\ reduced\ in\ Toggle\ Tree}{the\ original\ number\ of\ 1\ in\ Toggle\ Tree}$$

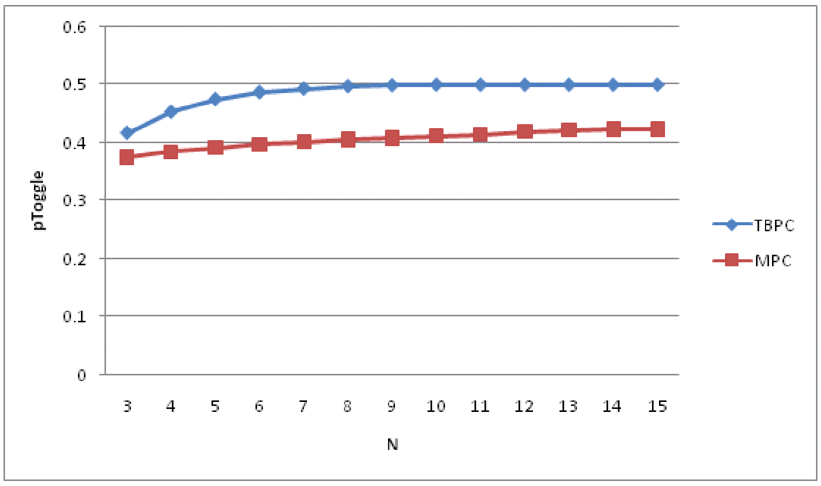


Figure 5.4: Diagram of pToggle

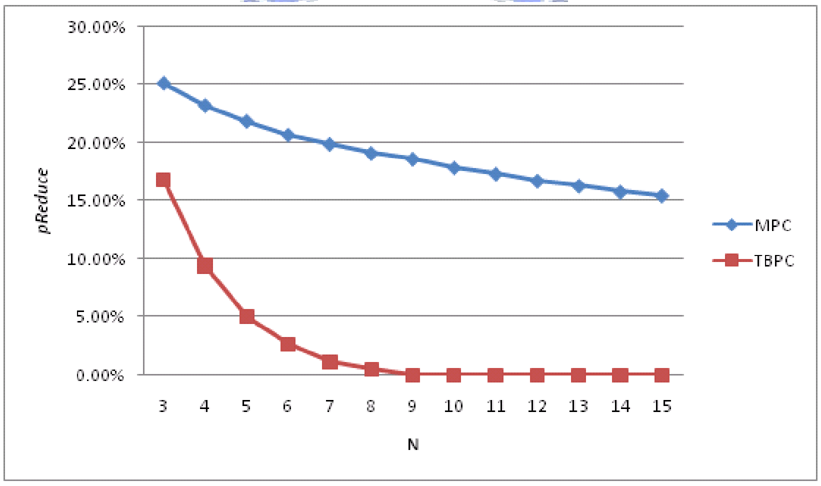


Figure 5.5: Diagram of pReduce

Chapter 6

Conclusion and Future Work

We have proposed an efficient scheme to help other data hiding algorithm to reduce pixel modification. Our scheme can achieve less pixel modification than TBPC in large N. We also prove that our scheme is a Linear Stego-code. Our scheme provides an efficient way to find a mapping from x to x'. According to these, our scheme can reduce distortion of host image and let the data hidden in host image be discovered more hardly. Because of being a plug-in process, it can be used extensively in steganography, not only for digital image, but also for video, music, or text files.

In our scheme, we sacrifice some space by a tree structure to achieve the main goal, reduction of modification, and lead a data coding concept into our scheme to solve the problem. Perhaps we can use much less modification to hide a same data by some other data coding method, and utilize idea of error correcting in data coding to resist attacks in Steganography.

References

[1] Richard Y.M. Li, Oscar C. Au, Kelvin K. Lai, Carman K.M. Yuk, Sui-Yuk Lam, Data hiding with tree based parity check, *IEEE International Conference on Multimedia and Expo(ICME)*, 2007.

[2] Akram M. Zeki, Azizah A. Manaf, A Novel Digital Watermarking Technique Based on ISB(Intermediate Significant Bit), *Proceeding of World Academy of Science, Engineering and Technology*, Volume 38, February 2009, ISSN: 2070-3740

[3] Zhe-Ming Lu, Zhen Sun, An Image Watermarking Method Based on Mean Removed Vector Quantization for Multiple Purposes, *IAPR Conference on Machine Vision Applications*, May 16-18, 2005

[4] Adnan M. Alttar, Reversible watermark using the difference expansion of a generalized integer transform, *IEEE Transaction on Image Processing*, vol. 13, no.8, Aug 2004

[5] Mohamed Bouchakour, Guillaume Jeanic, Florent Autrusseau, JND mask adaptation for wavelet domain watermarking, *IEEE International Conference on Multimedia and Expo(ICME)*, pp.201-204, 2008

[6] Veysel Aslantas, Saban Ozer, Serkan Ozturk, A novel fragile watermarking based on Particle Swarm Optimization, *IEEE International Conference on Multimedia and Expo*, pp. 269-272, 2008

[7] Lei Zheng, Ingemar J. Cox, JPEG Based Conditional Entropy Coding for Correlated Steganography, *IEEE International Conference on Multimedia and Expo*, pp.1251-1254, 2007

[8] G. Cohen, I. Honkala, S. Litsyn and A. Lobstein, Covering Codes, *North Holland*, 1997.

[9] R. Crandall, Some notes on steganography, *Posted on steganography mailing list*, 1998, http://os.inf.tu-dresden.de/westfeld/crandall.pdf

[10] J. Fridrich, M. Goljan, P. Lisonek and D. Soukal, Writing on wet paper, *IEEE Transactions on Signal Processing*, vol.53, pp. 3923-3935, 2005.

[11] Mahdad Khatirinejad and Petr Lisonek, Linear codes for high payload steganography, *Discrete Applied Mathematics*, vol.157, Issue 5, pp. 971-981, 2009.

[12] W. Zhang and S. Li, A coding problem in steganography, *Designs, Codes and Cryptography*, vol.46, no.1, 2008.

[13] S. Arora, L. Babai, J. Stern, Z. Sweedyk, The Hardness of Approximate Op-

tima in Lattices, Codes, and Systems of Linear Equations, *Journal of Computer and System Sciences*, vol.54, no.2, pp.317-331, 1997.

[14] A. Westfeld, F5: a steganographic algorithm, high capacity despite better steganalysis., *Proceedings of 4th International Workshop on Information Hiding. Lecture Notes in Computer Science*, vol.2137, pp.289-302, 2001.

[15] Ron M. Roth, Introduction to coding theory, *Cambridge University Press*, 2006.

[16] J. Fredrich, M.Goljan, and R. Du, Detecting LSB steganography in color and gray-scale images, *IEEE MultiMedia*, vol. 8, no. 4, pp. 22-28, 2001.

[17] A.D. Ker, Improved RS method for detection of LSB steganography, *Lecture Notes in Computer Science, Computational Science and Its Applications VICCSA 2005*, vol. 3481/2005, pp.508-516, 2005.

[18] A.Ker, Steganalysis of LSB matching in grayscale images, *IEEE Signal Processing Letters*, vol. 12, no.6, pp.441-444, June 2005.

[19] Jeng-Shyang Pan, Zhe-Ming Lu, Sheng-He Sun, An Efficient Encoding Algorithm for Vector Quantization Based on Sub-vector Technique, *IEEE Transactions on Image Processing*, vol.12, No. 3, March 2003.

[20] Feng-Hsing Wang, Jeng-Shyang Pan, Lakhmi C.Jain, Hsiang-Cheh, Huang-Cheh Huang, A VQ-Based Image-in-Image Data Hiding Scheme, *IEEE International Conference on Multimedia and Expo(ICME)*, 2004.

[21] C. C. Chang and P. Y. Lin, A Compression-based Data Hiding Scheme Using Vector Quantization and Principle Component Analysis, *3rd International Conference on Cyberworlds(CW 2004)*, pp. 369-375, 2004.

[22] Y. C. Hu, Grey-level Image Hiding Scheme Based on Vector Quantization, *Electronics Letters*, vol 39, no.2, pp.202-203, 23 Jan 2003.

[23] Shu-Kei Yip, Oscar C. Au, Hoi-Ming Wong, Chi-Wang Ho, Generalized Lossless Data Hiding by Multiple Predictors, *IEEE International Symposium on Circuits and Systems*, pp. 1426-1429, May 2006.

[24] Christian Cachin, Digital Steganography, *Encyclopedia of Cryptography and Security*, IBM Research, 2005.