An Optimal Data Hiding Scheme With Tree-Based Parity Check

Chung-Li Hou, ChangChun Lu, Shi-Chun Tsai, and Wen-Guey Tzeng

***Abstract*—Reducing distortion between the cover object and the stego object is an important issue for steganography. The tree-based parity check method is very efficient for hiding a message on image data due to its simplicity. Based on this approach, we propose a majority vote strategy that results in least distortion for finding a stego object. The lower embedding efficiency of our method is better than that of previous works when the hidden message length is relatively large.**

***Index Terms*—Image coding, image processing, information security.**

I. Introduction

Stenography studies the scheme to hide secrets into the communication between the sender and the receiver such that no other people can detect the existence of the secrets. A steganographic method consists of an embedding algorithm and an extraction algorithm. The embedding algorithm describes how to hide a message into the cover object and the extraction algorithm illustrates how to extract the message from the stego object. A commonly used strategy for steganography is to embed the message by slightly distorting the cover object into the target stego object. If the distortion is sufficiently small, the stego object will be indistinguishable from the noisy cover object. Therefore, reducing distortion is a crucial issue for steganographic methods. In this paper, we propose an efficient embedding scheme that uses the least number of changes over the tree-based parity check model.

Crandall [4] first introduced the idea of matrix embedding, which turned out to be very successful. Fridrich *et al.* [6] proposed a scheme, called the wet paper code, for the situation that some positions in the cover object are invariant. Fridrich and Soukal [8] discussed the scenario when the relative payload (the ratio of the hidden message length to the number of positions used for embedding in the cover object) is relatively large. Matrix embedding uses (n, k) linear codes, which is also called syndrome coding (this appears in [9] by Khatirinejad and Lisonek) or coset encoding ([2] introduced by Cohen *et al.*). It embeds and extracts a message by using the parity check matrix H of an (n, k) linear code. Zhang and Li [13] generalized the idea of matrix embedding and defined the codes with the matrix H as steganographic codes (abbreviated stego-codes). For matrix embedding, finding the stego object with least distortion is difficult in general. In some special cases, there exist constructive and fast methods. Fridrich *et al.* [7] utilized LT codes to improve the computational complexity of wet paper codes. Westfeld [12] derived a hash function to efficiently obtain the stego object. Li *et al.* [10] proposed a scheme called *tree-based parity check* (TBPC) to reduce distortion on a cover object based on a tree structure.

Manuscript received May 27, 2010; revised August 09, 2010; accepted August 19, 2010. Date of current version February 18, 2011. This work was supported in part by the National Science Council of Taiwan under Contracts NSC-97-2221-E-009-064-MY3, NSC-98-2221-E-009-078-MY3, NSC-98-2218-E-009-020, and NSC-96-2628-E-009-011-MY3. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Min Wu.

The authors are with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan 30050 (e-mail: clhou@csie.nctu.edu.tw; topple.cs94g@nctu.edu.tw; sctsai@cs.nctu.edu.tw; wgtzeng@cs.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2010.2072513

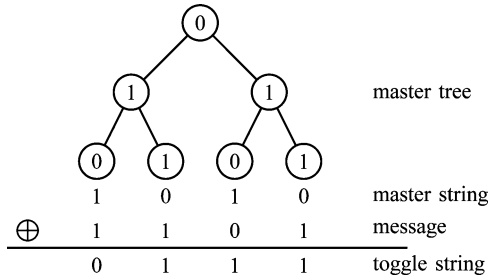

Fig. 1. Master and toggle strings of a master tree with $L = 4$ for LSBs 0, 1, 1, 0, 1, 0, 1 of the cover object.

The TBPC method can be formulated as a matrix embedding method, but is more efficient than those based on linear codes. Due to its simplicity, the TBPC method provides very efficient embedding and extraction algorithms. Recently, Zhang *et al.* [14] proposed a systematic method to generate codes with an arbitrary small relative payload from any code with a large relative payload. Since our method works naturally with large relative payloads, the result of Zhang *et al.* [14] implies that our method applies to small relative payloads as well.

We observe that the toggle criteria of a node in the TBPC method can be relaxed by the strategy of majority vote. Our strategy inherits the efficiency of the TBPC method and produces a stego object with least distortion under the tree based parity check model. The time complexity of our embedding (extraction as well) algorithm is asymptotically optimal, that is, it is linearly bounded by the hidden message length.

The *embedding efficiency* is defined to be the number of hidden message bits per embedding modification. Higher embedding efficiency implies better undetectability for steganographic methods. The *lower embedding efficiency* is defined to be the ratio of the number of hidden message bits to the maximum embedding modifications. The lower embedding efficiency is related to undetectability in the worst case. It implies steganographic security in the worst case. Thus, the lower embedding efficiency is also an important security factor for a steganographic system. In our method, it is $2 - \Theta(1/L)$, where L is the hidden message length and $\Theta(1/L)$ is a set of functions asymptotically bounded both above and below by $1/L$.

II. Preliminary and TBPC Method

Before embedding and extraction, a location finding method determines a sequence of locations that point to elements in the cover object. The embedding algorithm modifies the elements in these locations to hide the message and the extraction algorithm can recover the message by inspecting the same sequence of locations.

The TBPC method is a least significant bit (LSB) steganographic method. Only the LSBs of the elements pointed by the determined locations are used for embedding and extraction. The TBPC method constructs a complete N-ary tree, called the *master tree*, to represent the LSBs of the cover object. Then it fills the nodes of the master tree with the LSBs of the cover object level by level, from top to bottom and left to right. Every node of the tree corresponds to an LSB in the cover object. Denote the number of leaves of the master tree by L. The TBPC embedding algorithm derives an L-bit binary string, called the *master string*, by performing parity check on the master tree from the root to the leaves (e.g., see Fig. 1.).

The embedding algorithm hides the message by modifying the bit values of some nodes in the master tree. Assume that the length of the message is also L. Performing the bitwise exclusive-or (XOR) operation between the message and the master string, we obtain a *toggle string*

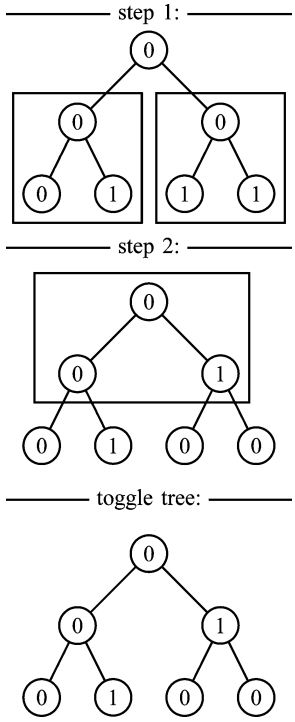

Fig. 2. Construction of a toggle tree with $L = 4$ for toggle string 0, 1, 1, 1.

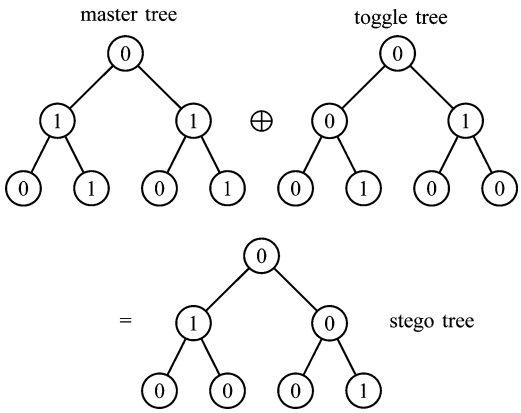

Fig. 3. Modify the master tree into the stego tree by the toggle tree constructed from the toggle string 0, 1, 1, 1.

(e.g., see Fig. 1). Then, the embedding algorithm constructs a new complete N-ary tree, called the *toggle tree* in the bottom-up order and fills the leaves with the bit values of the toggle string and the other nodes with 0. Then, level by level, from the bottom to the root, each nonleaf node together with its child nodes are flipped if all its child nodes have bits 1 (e.g., see Fig. 2). The embedding algorithm obtains the *stego tree* by performing XOR between the master tree and the toggle tree (e.g., see Fig. 3). The TBPC extraction algorithm is simple. We can extract the message by performing parity check on each root-leaf path of the stego tree from left to right.

III. Majority Vote Strategy

Two critical issues for a steganographic method are: 1) reducing distortion on cover objects and 2) better efficiency for embedding and extraction. We give a majority vote strategy on building the toggle tree. It uses the least number of 1's under the TBPC model. Since the number of 1's in the toggle tree is the number of modifications on the master tree (i.e., the cover object), the majority vote strategy can produce a stego tree with least distortion on the master tree.

A. Algorithm

Hereafter, we use majority-vote parity check (MPC) to denote our method due to its use of majority vote in deriving the parity check bit. We construct the toggle tree with the minimum number of 1's level by level in the bottom-up order using the following algorithm.

Algorithm MPC:

Input: a toggle string of length L;

1. Index the nodes of the initial toggle tree;
2. Set the leaves of the toggle tree from left to right and bit by bit with the toggle string and the other nodes 0;
3. **for** $i = 1$ **to** h

   for each internal node on level i **do**

   if the majority of its unmarked child nodes holds 1

   then flip the bit values of this node and its child nodes;

   else if the numbers of 0 and 1 in its unmarked child nodes are the same

   then mark this internal node;
4. **if** N is even **then**

   for $i = h - 1$ **for** 1

   for each marked internal node holding 1 on level i **do**

   flip the bit values of this node and its child nodes;

First, index all nodes of a complete N-ary tree with L leaves from top to bottom and left to right. Set the L-bit toggle string bit by bit into the L leaves from left to right and the other nodes 0. Assume that the level of the tree is h. Traverse all nonleaf nodes from level 1 to h. A nonleaf node and its child nodes form a simple complete subtree. For each simple complete subtree, if the majority of the child nodes hold 1, then flip the bit values of all nodes in this subtree. Since the construction is bottom-up, the bit values of the child nodes in every simple complete subtree are set after step 3. Note that marking a node at step 4 applies only for N being even. When N is even, after step 3, there may exist a two-level simple complete subtree with $N/2$ 1's in the child nodes and 1 in its root. In this case, flipping the bit values in this simple complete subtree results in one fewer node holding 1 and keeps the result of related root-leaf path parity check unchanged. Step 4 takes care of this when the condition applies, and it is done level by level from top to bottom. Also note that for the root of the whole toggle tree, the bit value is always 0 when half of its child nodes hold 1. Thus, after step 4, the bit values of the child nodes in each simple complete subtree are determined.

The number of 1's in the toggle tree is the number of modifications. When constructing the toggle tree, the original TBPC method flips a simple complete subtree only if all of child nodes have 1. We prove that the majority vote strategy actually obtains toggle trees with the least number of 1's.

We call a toggle tree with the least number of 1's corresponding to a toggle string an *optimal toggle tree*. We say that a toggle tree is in *majority form* if for each internal node at least half of its child nodes have bit value 0 and the internal node holds 0 when exactly half of its child nodes holding 1. The output of the algorithm is a toggle tree in majority form. The majority vote guarantees that at least half child nodes of an internal node hold 0. Note that every optimal toggle tree can

be transformed into majority form. It is obvious when N is even. When N is odd, we can check each 2-level simple complete subtree level by level in the top-down order and flip the bit values of the root node and its N child nodes if exactly $(N+1)/2$ of the child nodes hold 1. Note that, when this situation applies, the root node must hold 0 before flipping, otherwise the toggle tree is not optimal. This rearrangement does not introduce an extra 1 and the result of each root-leaf path parity check is not affected.

Theorem III.1: Algorithm MPC generates an optimal toggle tree with the least number of 1's under the tree based parity check model.

Proof: Let T_M be the N-ary toggle tree obtained by MPC and $T_{\rm opt}$ (in majority form) be an optimal N-ary toggle tree, which produces the same message bits as that of T_M by doing root-leaf path parity check. Let h denote the tree level of $T_{\rm opt}$ and T_M. We prove by induction on h and show that both trees have the same number of nodes holding 1 and both roots have the same bit value.

For $h = 1$, it is obvious that $T_{\rm opt}$ and T_M have the same distributing 0–1 values to the nodes, because they are both in both in majority form and generate the same message.

Assume our claim is true up to $h = k$, that is, for any optimal toggle tree $T_{\rm opt}$ of h levels, Algorithm MPC generates a toggle tree that has the same number of nodes as in $T_{\rm opt}$ holding 1 and produces the same message bits. For $h = k + 1$, assume T_{opt} have $k + 1$ levels. Let r be its root and $r_1, \ldots, r_N$ be the child nodes of r. Similarly, we let r' and r'_i's be the corresponding nodes of T_M.

Let m be the message produced by T_M and T_{opt}, where m can be partitioned into $m_1, \ldots, m_N$, and m_i can be obtained via the subtree rooted at r_i. If the bit value of r is 0, then the subtree rooted at r_i of $T_{\rm opt}$ is an optimal toggle tree that produces the message m_i; else it is an optimal toggle tree producing $\bar{m}_i$, which is the complimentary string of m_i. Since the subtrees rooted at r_i's have k levels, by induction hypothesis, each m_i can also be obtained by a k-level tree rooted at r'_i from MPC. The subtrees rooted at r_i and r'_i have the same number of nodes holding 1 and both roots have the same bit value. If r has bit value 0, then by making a majority vote over the bit values of r'_i's, we obtain an optimal toggle tree from MPC. If r has 1, then it needs more works to prove the correctness.

Observe that if for any two optimal toggle trees that produces two messages that are complimentary to each other, then the difference on the number of nodes holding 1 in both trees is at most 1, since we can always get a complimentary message by flipping the bit value of the root node. When r has bit value 1, there are more child nodes of r holding 0 and the optimal subtree rooted at r'_i from MPC, by induction hypothesis, actually produces $\bar{m}_i$. Note that r_i and r'_i have the same bit value, for $i = 1, \ldots, N$. If we flip the bit of r'_i, then the toggle tree rooted at r'_i will produce m_i but it may not be an optimal one for m_i. By the above observation, we know if r'_i had 1 before flipping, then after flipping it, the tree rooted at r'_i becomes optimal for m_i. On the other hand, if r'_i had 0 before flipping, then after flipping it, the tree rooted at r'_i may have one more node holding 1 than an optimal toggle tree that produces m_i. Thus, there will be more than half of r'_i's flipped from 0 to 1. Then by taking a majority vote over flipped r'_i's, we flip them back and have a new root, r', holding 1. Therefore, the total number of nodes holding 1 is the same as $T_{\rm opt}$, and it can be obtained by MPC. This completes the proof. ■

B. Binary Linear Stego-Code

Before showing that our method is actually a special binary linear stego-code, we briefly review the definition of linear stego-codes. With matrix embedding, given any message $m \in \mathbb{F}_2^{(n-k)}$ and any cover object $x \in \mathbb{F}_2^n$, the problem is to find a vector $\delta \in \mathbb{F}_2^n$ and an $(n-k)\times n$ matrix H over $\mathbb{F}_2$ such that $wt(\delta)$ is as small as possible and $Hx' = m$, where $x' = x + \delta$ and $wt(\delta)$ is the Hamming weight of δ. Zhang and Li [13] generalized this idea and defined the stego-coding matrix and the linear stego-code as follows.

Definition III.1: An $(n-k)\times n$ matrix H over $GF(q)$ is called an $(n, n-k, t)$ stego-coding matrix if, for any given $y \in GF^{(n-k)}(q)$, there exists a vector $v \in GF^n(q)$ such that $wt(v) \le t$ and $Hv = y$.

Definition III.2: Let H be an $(n, n-k, t)$ stego-coding matrix. For all $y \in GF^{(n-k)}(q)$, let $s_y = \{v : Hv = y,\ v \in GF^n(q)\}$. An $(n, n-k, t)$ linear stego-code is defined by $S = \{s_y : s_y \neq \emptyset\}$.

In comparison with matrix embedding, v is the distortion δ and y is $m - Hx$, where m is the message and x is the cover object. An $(n, n-k, t)$ linear stego-code guarantees that the distortion is at most t bits for any given message and cover object.

In practice, the sender and the receiver agree on a matrix H in advance. The cover object is represented as a binary vector x (e.g., for an image, take the LSBs of all pixels) and the message is also a binary vector m. For embedding, the sender identifies a vector x' such that $Hx' = m$. For extraction, the receiver extracts the hidden message m from the stego object x' by computing $Hx' = m$. Finding x' with least distortion is to solve $H\delta = m - Hx$ such that $wt(\delta)$ is minimum. Finding δ with least weight is the well known coset leader problem [9]. It is equivalent to the nearest codeword problem (NCP) for binary linear codes (see Section 2.4 in [11] by Roth). NCP is to find a codeword c such that $wt(y - c)$ is minimum, given a $k \times n$ matrix A over $GF(2)$ and a vector $y \in GF^n(2)$. Arora *et al.* [1] have proved that even approximating NCP within any constant factor is NP-hard. In general, NCP is extremely difficult. However under the tree-based structure, we can efficiently solve it.

Hiding a message with the tree-based parity check structure can be treated as a kind of linear binary stego-codes. The parity check operations on a tree can be formulated as a matrix operation. More specifically, consider a complete N-ary complete tree with n nodes, h levels and $L = N^h$ leaves. There are L paths and each with $h + 1$ nodes. Enumerate the paths from left to right. For path i, we define an n-dimensional binary vector v_i, where the j-th entry is 1 if and only if path i has a node with index j. Define H to be the $L \times n$ matrix, where the i-th row is v_i, $i = 1, 2, \ldots, L$. Use the n-dimensional binary vector x to represent the cover object, where x_i is associated with the node of the master tree with index i. Therefore, Hx has the result of tree-based parity check. In other words, the TBPC method is simply a special case of linear binary stego-codes.

Theorem III.2: Given a cover object x of length n, a message y with length L and an N-ary toggle tree, the tree based parity check steganographic method with majority vote strategy is equivalent to an $(n, L, (L+1)/2)$ linear stego-code.

Proof: Let H be the $L \times n$ matrix corresponding to the tree based structure, and x' be the n-dimensional vector corresponding to the stego tree. Therefore, $H \times x' = y$. According to the definition of linear stego-codes, the remaining is to analyze the distortion between x and x'. The distortion is the number of 1's in the toggle tree. Since the construction of the toggle tree is in the bottom-up order, only leaf nodes hold 1 initially. For even N, the majority vote always reduces the number of 1's in the toggle tree while flipping. Therefore, the worst case for even N is that all the simple complete subtrees with leaf nodes as child nodes have $N/2$ child nodes holding 1. The maximum number of 1's in the toggle tree for even N is $(L/N)(N/2) = L/2$. When N is odd, every simple complete subtree of the toggle tree in majority form has at most $\lfloor N/2 \rfloor$ child nodes holding 1. Let $K = \lfloor N/2 \rfloor$. The worst case for odd N is that the root holds 1 and K child nodes of every simple complete subtree hold 1. The maximum number of 1's in the toggle tree for odd N is

$$1 + \sum_{i=0}^{(\log_N L)-1} N^i K = 1 + K\left(\frac{(L-1)}{(N-1)}\right) = \frac{(L+1)}{2}.$$

Therefore, the distortion is at most $(L+1)/2$. This completes the proof of the theorem. ■

IV. Analysis and Experimental Results

A. Average Modifications per Hidden Bit

It is easy to construct a method that achieves the expected embedding modifications per hidden bit of 0.5. In other words, if we try to embed an L-bit message into the cover object, $0.5\ L$ modifications will occur on average. We use

$$pToggle = \frac{D_a}{L} \tag{1}$$

to denote the expected embedding modifications per hidden bit, where D_a is the average number of embedding modifications for an L-bit message.

Recall that the MPC method performs majority vote on every simple complete subtree to construct the toggle tree in the bottom-up order. Therefore, we are going to calculate the expected reduced number of 1's for every simple complete subtree and sum up the expected reduced number of 1's for all simple complete subtrees.

For convenience, we use *i-level tree* to denote a complete N-ary tree of i levels. An i-level tree consists of one root and N $(i-1)$-level trees. An i-level simple complete subtree is a two-level tree containing a node v at level i and all its child nodes.

For an h-level toggle tree, the level of the root is h and the level of a leaf is 0. Let $P(i)$ be the probability that the root of an i-level simple complete subtree holds 1 after performing majority vote. For the leaf nodes, $P(0)$ is $1/2$ because the leaf nodes are uniformly filled with 0 or 1. For every i-level simple complete subtree, $P(i)$ is the same by symmetry. Let $\lfloor N/2 \rfloor = K$. Since the toggle tree is an N-ary complete tree constructed by the majority vote strategy, $P(i)$ can be expressed as follows:

$$P(i) = \sum_{j=K+1}^{N} \binom{N}{j} P(i-1)^j [1-P(i-1)]^{N-j}. \tag{2}$$

Let $R(t)$ be the reduced number of 1's after flipping the bit values of a simple complete subtree that holds t 1's. Therefore, $R(t) = t - (N+1-t) = 2t - N - 1$. The expected reduced number of 1's for an i-level simple complete subtree is as follows:

$$E(i) = \sum_{j=K+1}^{N} R(j) \binom{N}{j} P(i-1)^j [1-P(i-1)]^{N-j}. \tag{3}$$

For an L-bit toggle string, the expected number of 1's in the toggle string is $0.5\ L$. In the first step for the toggle tree construction, we fill each leaf with one bit of the toggle string. Before majority vote, the number of 1's in the toggle tree is $0.5\ L$. After majority vote, the number of 1's in the toggle tree is $0.5\ L - \sum_{i=1}^{h} N^{h-i} E(i)$. Since the number of modifications is the number of 1's in the toggle tree, we finally have the following equation:

$$pToggle = 0.5 - \frac{1}{L} \sum_{i=1}^{h} N^{h-i} E(i). \tag{4}$$

If $N = 2K+1$ is an odd integer, (3) can be further simplified as

$$E(i) = \sum_{j=K+1}^{N} R(j) \binom{N}{j} \left(\frac{1}{2}\right)^N \tag{5}$$

since

$$\sum_{j=0}^{K} \binom{N}{j} = \sum_{j=K+1}^{N} \binom{N}{j} = 2^{N-1}$$

$$P(i) = \sum_{j=K+1}^{N} \binom{N}{j} \left(\frac{1}{2}\right)^N = \frac{1}{2}.$$

TABLE I
Comparison of Embedding Time for the MPC Method and the Simplex Code-Based Method Under the Same Relative Payload α

α	$(2^q-1, q)$ Simplex	(n, L) MPC	embedding time(ns) per hidden bit	
			Simplex	MPC
0.73	(15,4)	$(85, 4^3)$	433	138
0.83	(31,5)	$(259, 6^3)$	365	96
0.9	(63,6)	$(1111, 10^3)$	351	82
0.945	(127,7)	$(6175, 18^3)$	377	75
0.99	(1023,10)	$(9901, 99^2)$	604	64

The *pToggle* of the TBPC method is

$$pToggle(i) = pToggle(i-1) - \frac{N-1}{C(i)L(i)} \tag{6}$$

where $L(i)$ is the number of leaves and $C(i)$ is the number of possible 0–1 configurations in leaves for an i-level tree.

B. Time Complexity of MPC

For embedding of the MPC method, the construction of an L-bit master string from a master tree is to perform parity check on L simple root-leaf paths. The number of parity check operations for each simple root-leaf path is the number of edges in this path. Since we perform parity check once for every edge, the total number of parity check operations is the number of edges in the master tree. Since the number of nodes in the master tree is

$$\sum_{i=0}^{\log_N L} N^i = \frac{(NL-1)}{(N-1)} = L + \frac{(L-1)}{(N-1)}$$

the time complexity to obtain a master string is $O(L)$. The time complexity to obtain the toggle string is $O(L)$ since the toggle string is derived by performing bitwise exclusive-or between the L-bit message and the L-bit master string. Thus, the total time complexity of the embedding algorithm is $O(L)$. For the extraction algorithm, we perform parity check on L simple root-leaf paths in the stego tree. Thus, the complexity of the extraction algorithm is also $O(L)$.

C. Comparison for Large Payloads

Fridrich and Soukal [8] proposed two matrix embedding methods based on random linear codes and simplex codes. The time complexity of embedding algorithms for matrix embedding is bounded by the complexity of the decoding algorithms for codes, i.e., the complexity of finding the coset leader. The decoding algorithms for (n, k) random linear codes and (n, k) simplex codes in [8] have time complexity $O(n2^k)$ and $O(n \log n)$, respectively, where n is the code length and k is the dimension of the code. Both methods have the hidden message length $n-k$. The time complexity $O(L) = O(n-k)$ of our method is much better.

Table I describes the experimental embedding time for our method and the method based on simplex codes. For a fixed relative payload, we compare the embedding time (in nanoseconds) per hidden message bit. Our method is at least three times faster than the method based on simplex codes. The experiment was run on a Windows XP system with Athlon 2.21 GHz CPU, 1 GB RAM and implemented in JAVA. Under the same experimental environment, we simulated embedding for a 1280×1024 image. The comparison of embedding time with a similar block length and relative payload is in Table II. The embedding time of the MPC method is better. Fridrich and Soukal [8] simulated embedding for a 1280×1024 image using (n, k) random codes with block length $n = 100$ and $k = 10,\ 12$, and 14. The experiment run by Fridrich and Soukal [8] was on a Linux system with Pentium IV

TABLE II
COMPARISON OF EMBEDDING TIME FOR A 1280×1024 IMAGE WITH A SIMILAR RELATIVE PAYLOAD α AND BLOCK LENGTH

		embedding time (ms, in JAVA)	α
(n,k) Random	(100,10)	13536	0.9
	(100,12)	52297	0.88
	(100,14)	207834	0.86
$(2^q-1,q)$ Simplex	(127,7)	471.64	0.94
(n,L) MPC	$(91,9^2)$	115.92	0.89
	$(111,10^2)$	113.62	0.9
	$(133,11^2)$	105.96	0.91

3.4-GHz CPU, 1-GB RAM and implemented in C++. The embedding time for $k = 10, 12$, and 14 is 0.82, 2.42, and 8.65 s, respectively. The embedding time for the MPC method even implemented in JAVA is better than the random code-based method implemented in C.

For extraction algorithms, both of simplex code-based and random code-based methods need to calculate Hy, where H is the parity check matrix and y is the stego object. The number of 1's in H dominates the time complexity for extraction algorithms. For (n,k) simplex codes (the dual of hamming codes), the generator matrix $G_{k\times n}$ consists of all possible nonzero k-dimension column vectors. The number of 1's in $G_{k\times n}$ is $nk/2$. Since the generator matrix $G_{k\times n}$ can be rearranged in systematic form, say $G_{k\times n} = [I_{k\times k}|A]$, the parity check matrix $H_{(n-k)\times n}$ is $[A^T|I_{(n-k)\times(n-k)}]$, where A^T is the transpose of A. Therefore, the number of 1's in H is $nk/2 - k + (n-k) = nk/2 + n - 2k$. The time complexity of extraction algorithms based on simplex codes is $O(nk/2 + n - 2k)$. For random linear codes, the number of 1's in the parity check matrix, $H = [I_{(n-k)\times(n-k)}|D]$, depends on the distribution of the random source. Therefore, for random linear codes, its time complexity is $O(n-k+k^2)$ on average and for MPC its time complexity is simply bounded by the hidden message length, $O(L) = O(n-k)$.

Fridrich and Soukal [8] also considered the relative payload $\alpha = L/P_{\text{cover}}$ where L is the number of hidden message bits and P_{cover} is the number of positions (e.g., pixels for an image) used for embedding in the cover object. Let $q \geq 3$ be integer. The relative payloads for methods based on an (n,k) random linear code and a practical $(2^q - 1, q)$ simplex code of dimension q and code length 2^q-1 are $(n-k)/n$ and $(2^q-1-q)/(2^q-1) = 1-q/(2^q-1)$, respectively. The relative payload for MPC is

$$\alpha = \frac{L}{\frac{NL-1}{N-1}} = 1 - \frac{L-1}{NL-1} = 1 - \frac{N^h-1}{N^{h+1}-1}.$$

Recall that D_a is the average number of modifications for embedding L bits. The *embedding efficiency* is defined as

$$e = \frac{L}{D_a} = \frac{1}{pToggle}.$$

By experiments, we observed that the embedding efficiency of the MPC method is slightly smaller (within 0.5 when $\alpha > 0.8$) than those of Fridrich and Soukal [8]. There is a tradeoff between time complexity and embedding efficiency. Our method has lower time complexity with slight sacrifice on embedding efficiency. Fig. 4 shows the relation between embedding efficiency and the relative payloads for MPC and the two methods of Fridrich and Soukal [8]. For simplex codes, we choose augmented simplex code (adding an all 1's row vector to the generator matrix) for $q = 4, 5, \ldots, 11$. For random linear codes, we only consider the codes with $k = 14$ and the relative payloads which are close to simplex codes for $q = 4, 5, 6$. For MPC, we calculate embedding efficiency for fixing $h = 3$ and 10 with the relative payloads

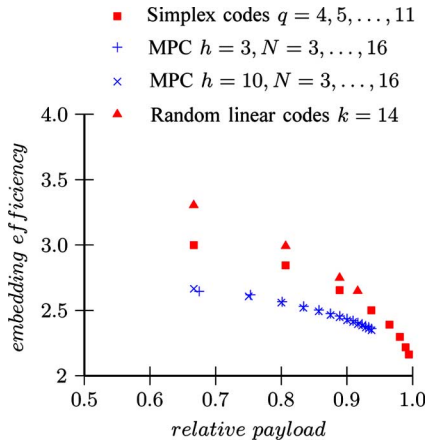

Fig. 4. Embedding efficiency versus relative payload.

TABLE III
COMPARISON OF EMBEDDING EFFICIENCY e AND THE HIDDEN MESSAGE LENGTH L FOR THE MPC METHOD AND THE AUGMENTED SIMPLEX CODE-BASED METHOD UNDER SIMILAR EMBEDDING TIME t

t (10^3ns)	$(2^q-1,q+1)$ Augmented Simplex	(n,L) MPC	e ; L	
			Simplex	MPC
6	(15,5)	$(40,3^3)$	3 ; 10	2.63 ; 27
13	(31,6)	$(156,5^3)$	2.84 ; 25	2.55 ; 125
29	(63,7)	$(400,7^3)$	2.66 ; 56	2.49 ; 343
60	(127,8)	$(820,9^3)$	2.5 ; 119	2.45 ; 729

$1-(N^h-1)(N^{h+1}-1)$, where $N = 3, 4, \ldots, 16$. Table III describes the comparison of embedding efficiency under similar embedding time. Recall that the hidden message length is L for the MPC method and $2^q - 1 - (q+1)$ for the augmented simplex code-based method. Our method embeds more bits with slight sacrifice on embedding efficiency under similar embedding time.

The embedding efficiency focuses on the average modifications. On the other hand, the *lower embedding efficiency* $\underline{e}$ concerns about the maximum modifications. It is defined as

$$\underline{e} = \frac{L}{D}$$

where D is the maximum modifications for embedding L bits. The covering radius of the codes used for matrix embedding determines the maximum modifications. For $(2^q-1,q)$ simplex codes, the covering radius is $2^{q-1}-1$ (see [3, Appendix B] proposed by Cohen *et al.*). The lower embedding efficiency is

$$\frac{(2^q-1-q)}{(2^{q-1}-1)} = 2 - \Theta\left(\frac{\log n}{n}\right).$$

By Theorem III.2, MPC is equivalent to an $(n, L, (L+1)/2)$ linear stego-code. The maximum embedding modifications for MPC is $(L+1)/2$. The lower embedding efficiency for MPC is

$$\frac{L(L+1)}{2} = \frac{2-2}{(L+1)} = 2 - \Theta\left(\frac{1}{L}\right) = 2 - \Theta\left(\frac{1}{n}\right)$$

when $\alpha = L/n \to 1$. Note that $2 - \Theta(1/n)$ is better than $2 - \Theta(\log n/n)$, when L is relatively large.

Our method is a natural stego-code with a large relative payload. For N-ary trees with $N \geq 2$, the relative payload of our method is $1 - N^h - 1/N^{h+1} - 1$, larger than 0.5. Recently, Zhang *et al.* [14] gave a construction (called the ZZW construction) to generate a family of codes with arbitrary small relative payloads from any code with

TABLE IV
SUMMARY OF COMPARISON

	$\underline{e}$	$e(\alpha \approx 0.8)$	time complexity
MPC	$2-\Theta(1/n)$	2.55	$O(L)=O(n-k)$
Simplex	$2-\Theta(\log n/n)$	2.84	$O(n \log n)$
Random	-	2.99	$O(n2^k)$

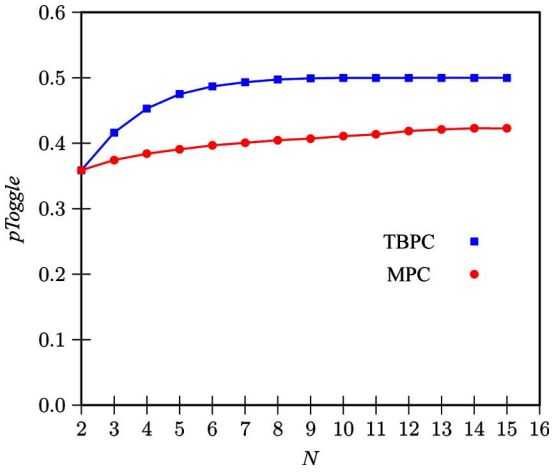

Fig. 5. *pToggle* comparison of MPC and TBPC with different N and about 15000 leaf nodes.

TABLE V
EXPERIMENTAL RESULTS OF *pToggle*

N	TBPC	MPC	N	TBPC	MPC
2	0.3589	0.3589	9	0.4991	0.4071
3	0.4164	0.3744	10	0.4999	0.4108
4	0.4531	0.384	11	0.4998	0.4136
5	0.475	0.3908	12	0.4999	0.4187
6	0.4869	0.3967	13	0.4999	0.4212
7	0.4933	0.4007	14	0.4999	0.4229
8	0.4974	0.4047	15	0.4999	0.4228

a large relative payload. Fridrich [5] proved that the embedding efficiency of the family of codes generated by the ZZW construction follows the upper bound on embedding efficiency. By applying the ZZW construction, we can generate codes with small relative payloads and good embedding efficiency. Table IV summarizes the comparison of our (n, L) stego-codes and the methods based on (n, k) simplex codes and (n, k) random linear code.

D. Experimental Results for MPC and TBPC

We implemented our MPC method and the TBPC method for a comparison between their *pToggle* values. We constructed N-ary toggle trees with more than 15000 leaf nodes for $N = 2, 3, \ldots, 15$. For each N, we randomly generated 200 distinct toggle strings. The results are shown in Fig. 5 and Table V. The results show that MPC is always better than TBPC for $N \geq 3$. When $N = 2$, they are the same.

To make it clear, we define the percentage of reduced modifications as follows:

$$pReduce = \frac{R_t}{D_t}$$

where R_t is the reduced number of 1's in the toggle tree and D_t is the number of 1's in the toggle string. The *pReduce* values of both methods are shown in Fig. 6 and Table VI. The results show that the MPC method significantly improves previous TBPC results.

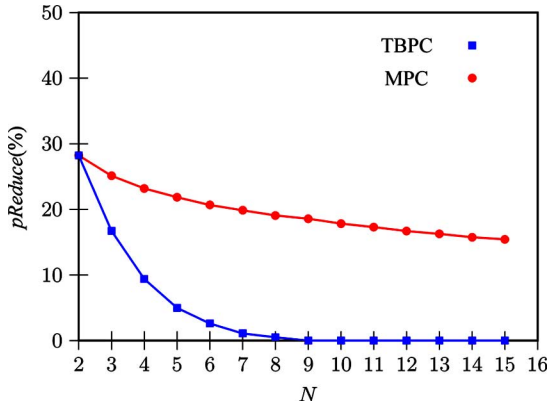

Fig. 6. Comparison of *pReduce*.

TABLE VI
EXPERIMENTAL RESULTS OF *pREDUCE*

N	MPC	TBPC	N	MPC	TBPC
2	28.21%	28.21%	9	18.57%	0.00%
3	25.11%	16.71%	10	17.83%	0.00%
4	23.19%	9.38%	11	17.29%	0.00%
5	21.84%	4.98%	12	16.69%	0.00%
6	20.66%	2.60%	13	16.26%	0.00%
7	19.85%	1.08%	14	15.75%	0.00%
8	19.07%	0.50%	15	15.43%	0.00%

E. Applications

Our method is based on an N-ary complete tree structure. Fixed the level of the tree, given a larger N we can hide more message bits and the relative payload is larger. Like the previous works proposed by Fridrich and Soukal [8], our method can be applied to the situation that the relative payload is large. On the other hand, since our method is asymptotically optimal, the embedding and extraction algorithms are efficient and can be used on online communications.

V. CONCLUSION

By introducing the majority vote strategy, we effectively construct the stego object with least distortion under the tree structure model. We also show that our method yields a binary linear stego-code. In comparison with the TBPC method, our method significantly reduces the number of modifications on average.

REFERENCES

[1] S. Arora, L. Babai, J. Stern, and Z. Sweedyk, "The hardness of approximate optima in lattices, codes, and systems of linear equations," *J. Comput. Syst. Sci.*, vol. 54, no. 2, pp. 317–331, 1997.
[2] G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein, *Covering Codes*. Amsterdam, The Netherlands: North-Holland, 1997.
[3] G. Cohen, M. Karpovsky, H. Mattson, and J. Schatz, "Covering radius-survey and recent results," *IEEE Trans. Inf. Theory*, vol. IT-31, no. 3, pp. 328–343, May 1985.
[4] R. Crandall, "Some Notes on Steganography, Posted on Steganography Mailing List," 1998 [Online]. Available: http://os.inf.tu-dresden.de/westfeld/crandall.pdf
[5] J. Fridrich, "Asymptotic behavior of the ZZW embedding construction," *IEEE Trans. Inf. Forensics Security*, vol. 4, no. 1, pp. 151–154, Mar. 2009.
[6] J. Fridrich, M. Goljan, P. Lisonek, and D. Soukal, "Writing on wet paper," *IEEE Trans. Signal Process.*, vol. 53, no. 10, pp. 3923–3935, Oct. 2005.

[7] J. Fridrich, M. Goljan, and D. Soukal, "Efficient wet paper codes," in *Proc. 7th Int. Workshop Inf. Hiding (IHW 05), Lecture Notes in Computer Science*, 2005, vol. 3727, pp. 204–218.

[8] J. Fridrich and D. Soukal, "Matrix embedding for large payloads," *IEEE Trans. Inf. Forensics Security*, vol. 1, no. 3, pp. 390–395, Sep. 2006.

[9] M. Khatirinejad and P. Lisonek, "Linear codes for high payload steganography," *Discrete Applied Math.*, vol. 157, no. 5, pp. 971–981, 2009.

[10] R. Y. M. Li, O. C. Au, K. K. Lai, C. K. Yuk, and S.-Y. Lam, "Data hiding with tree based parity check," in *Proc. IEEE Int. Conf. Multimedia and Expo (ICME 07)*, 2007, pp. 635–638.

[11] R. M. Roth, *Introduction to Coding Theory*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[12] A. Westfeld, "F5: A steganographic algorithm, high capacity despite better steganalysis," in *Proc. 4th Int. Workshop Inf. Hiding,*, 2001, vol. LNCS 2137, pp. 289–302.

[13] W. Zhang and S. Li, "A coding problem in steganography," *Designs, Codes Cryptogr.*, vol. 46, no. 1, pp. 68–81, 2008.

[14] W. Zhang, X. Zhang, and S. Wang, "Maximizing steganographic embedding efficiency by combining hamming codes and wet paper codes," in *Proc. Int. Workshop Inf. Hiding (IH 08)*, 2008, vol. LNCS 5284, pp. 60–71.