IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 2 Issue 4, April 2015.
www.ijiset.com

ISSN 2348 – 7968

Information Cloaking Technique with Tree Based Similarity

C.Bharathipriya[1], K.Lakshminarayanan[2]

[1]Final Year, Computer Science and Engineering, Mailam Engineering College,

[2]Assistant Professor, Computer Science and Engineering, Mailam Engineering College.

Abstract
Steganography and Cryptography deals major role in this technique. The process of distortion reduction is done by tree based parity check. Tree based parity check uses majority vote strategy. For cloaking a message tree based parity check is very optimal. Security Enhancement is implemented by SHA-1. Large payload can be given as input to the proposed system.

***Keywords*— Image coding, covering code, embedding efficiency, selection channel**

I. INTRODUCTION

In steganography, the detectability of hidden data in a stego object is mostly influenced by four basic ingredients. 1) the cover object, 2) the selection rule used to identify individual elements of the cover that might be modified during embedding, 3) the type of embedding operation that modifies the cover elements, and 4) the number of embedding changes (related to the message length).

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.

The Matrix embedding uses *(n,k)* linear codes, which is also called syndrome coding or coset encoding). It embeds and extracts a message by using the parity check matrix H of an *(n,k)* linear code. For matrix embedding, finding the stego object with least distortion is difficult in general. In some special cases, there exist constructive and fast methods. It utilized LT codes to improve the computational complexity of wet paper codes. It derived a hash function to efficiently obtain the stego object. The proposed a scheme called *tree-based parity check* (TBPC) to reduce distortion on a cover object based on a tree structure.

The Wet Paper Code with Improved Efficiency provides a new tool for steganography. a coding method that empowers the steganographer with the ability to use arbitrary selection channels while substantially decreasing the number of embedding changes, assuming the embedded message length is shorter than 70% of maximal embedding capacity. The method can be flexibly incorporated as a module into majority of existing steganographic methods.

The “Wet Paper” channel is highly relevant to steganography and arises in numerous different situations. One of them is adaptive steganography, where the sender selects the location of pixels that will carry message bits based on pixels neighbourhood in the cover image. A fundamental problem with adaptive schemes is that the requirement that the recipient be able to recover the same message-carrying pixels from the stego image undermines the security of the algorithm because it gives an attacker a starting point for mounting an attack. Another potential problem is that the recipient may not be able to recover the same set of message carrying pixels from the stego image, which is modified by the embedding act itself. This problem is usually solved either by increasing the message redundancy using error correction to recover from random bit losses and inserts or by employing some artificial measures, such as special embedding operations matched to the selection rules. These measures, however, usually limit the capacity, complicate the embedding algorithm, and do not give the sender the ability to fully utilize his side-information the cover image. Moreover, the pixel selection rule is often ad hoc and it is not always possible to justify it from the point of view of steganographic security. In fact, ideally, the sender should be able to use any available side information in an unrestricted manner and perform embedding while focusing on important steganographic design principles and necessary security considerations rather than the receiver’s ability to read the message.

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

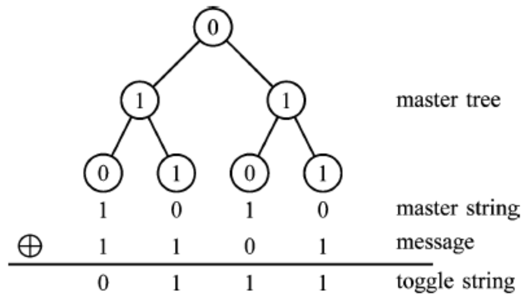

Fig.1. Master and Toggle string of master tree with L=4.

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* (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.

The proposed method uses a set of standard measures, to capture image properties before or after the embedding process, which effect
the performance of steganalysis techniques. These measures are divided into two categories. First cover image properties, and second cover-stego based distortion measures.

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

ISSN 2348 – 7968

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.

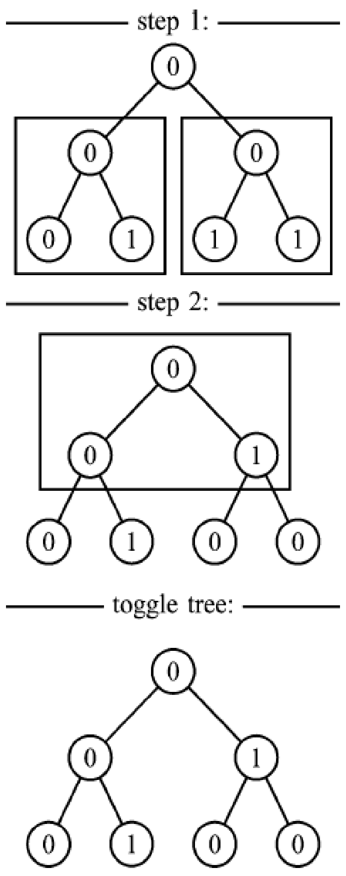

Fig.2. Construction of a toggle tree with *L=4*.

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 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.

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 $m \in F_2^{(n-k)}$and any cover object $x \in F_2^{(n)}$, the problem is to find a vector $\delta \in F_2^{(n)}$ and an $(n\text{-}k)\times n$ matrix H over F_2 such that $wt(\delta)$ is as small as possible and $H\,x' = m$, 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.

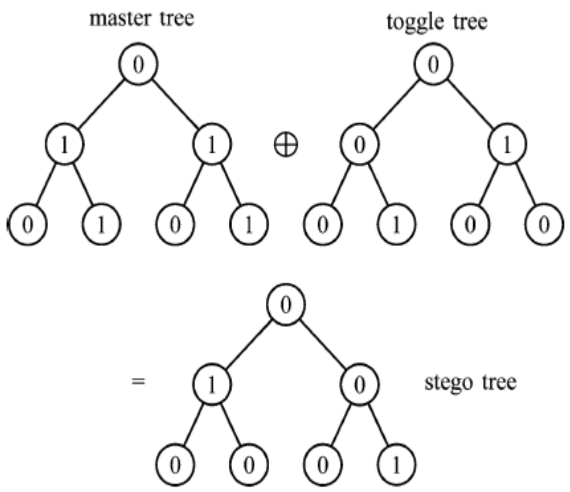

Fig.3. Modify the Master tree into stego tree by the toggle tree constructed from the toggle string.

Theorem III.1: 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 analyse 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 *N*/2 child nodes holding 1. Let $K=N/2$. 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

$$pToggle=$$

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 -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 be *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 ½ 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 *N/2=K*. Since the toggle tree is an *N*-ary complete tree constructed by the majority vote strategy, *P(i)* can be expressed as follows:

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

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 _ 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*, and 14. The experiment run by Fridrich and Soukal [8] was on a Linux system with Pentium IV

TABLE I
COMPARISON OF EMBEDDING TIME FOR 1280×1024 IMAGE WITH A SIMILAR RELATIVE PAYLOAD

		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-GBRAMand implemented in C++. The embedding time for k=10, 12, and 14 is 0.82, 2.42, and 8.65 s, respectively. The SR can be defined differently based on other heuristics, the image format, and properties of image pixels/coefficients. The embedding time for the MPC method even implemented in JAVA is better than the random code-based method implemented in C.

D. Experimental Results of 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,...15. For each N, we randomly generated 200 distinct toggle strings. The results are shown in Fig. 4 and Table II. 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. 4 . The results show that the MPC method significantly improves previous TBPC results.

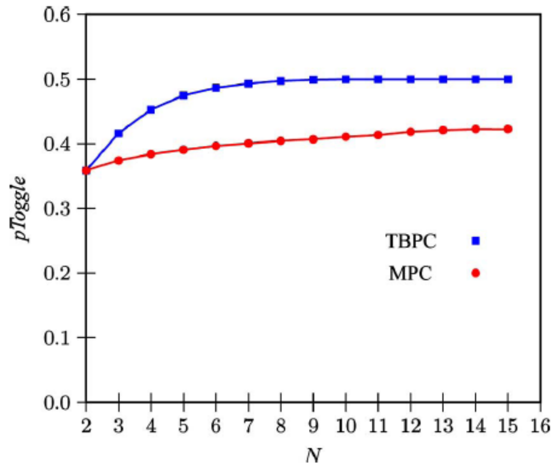

Fig.4. *pToggle* comparison of MPC and TBPC.

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.

While this decrease in embedding distortion is quite substantial by itself, we point out another important consequence of improved embedding efficiency. We can now embed up to 2.2 × 80 = 176 bits with the same embedding distortion as the one due to embedding 80 bits using regular wet paper codes. Thus, instead of decreasing the embedding distortion, we may choose to improve the robustness of embedded data by applying strong error correction code to the payload. Therefore, the improved embedding efficiency can be utilized either for decreasing the visual impact of

embedding or to improve the robustness of the embedded data to channel noise.

TABLE III
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

E. Applications

The proposed 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.

The Majority vote strategy provides an efficient, general, and elegant tool to solve the problem of non shared selection channel, which is quite common in steganography. For example, the sender can now use arbitrary side information, such as a high-resolution version of the cover, for selecting the placement of embedding changes within the cover image and further minimize the embedding distortion. This selection channel helps minimize the total distortion due to quantization (e.g., as in lossy compression) and embedding.

Another application that greatly benefits from the proposed method is adaptive steganography. In adaptive steganography, the pixels are selected for embedding based on their local context (neighbourhood). However, the act of embedding itself will modify the cover image and thus the recipient may not be able to identify the same set of message-carrying pixels from the stego image. This problem becomes especially pronounced and hard to overcome for data hiding in binary images.

V.CONCLUSION

By introducing the majority vote strategy, the stego object is constructed with least distortion under the tree structure model. We also show that our method yields a binary linear stego-code and preserves the secrecy of the hidden data. In comparison with the TBPC method, the proposed MPC method significantly reduces the number of modifications on average.

REFERENCES

[1] S. Chang Chun Lu, Shi-Chun Tsai, and Wen-Guey Tzeng, " An Optimal Data Hiding Scheme With Tree-Based Parity Check" in IEEE Trans. on Image Processing, vol. 20, no. 3, Mar 2011.

[2] J. Fridrich, M. Goljan, P. Lisonek, and D. Soukal, "Writing on wet paper," in IEEE Trans. on Signal Processing., vol. 53, no. 10, pp. 3923–3935, Oct. 2005.

[3] J. Fridrich, M. Goljan, and D. Soukal, "Efficient wet paper codes," in Proc. 7th Int. Workshop Information Hiding (IHW 05), vol. 3727, pp. 204–218, Apr 2005.

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

[5] M. Khatirinejad and P. Lisonek, "Linear codes for high payload steganography," in IEEE Trans. on Discrete Applied Mathematics, vol. 157, no. 5, pp. 971–981, 2009.

[6] W. Zhang and X. Wang, "Generalization of the ZZW embedding construction for Steganography," in IEEE Trans. on Information Forensics and Security, pp.564-569, Jan 2009.

[7] A. Westfeld, "F5: A steganographic algorithm, high capacity despite better steganalysis," in Proc. 4th Int. Workshop Information Hiding, vol. LNCS 2137, pp. 289–302, Jan 2002.

[8] Wu, M., Fridrich, J., Goljan, M., and Gou, H.: "Data Hiding in Digital Binary Images," in Proc. SPIE, Electronic Imaging, Security, Steganography, and Watermarking, pp. 194–205, May 2005.

[9] J. Fridrich, M. Goljan, and D. Soukal, "Wet Paper Code with Improved embedding Efficiency," in IEEE Trans. on Information Forensics and Security, Jul 2006.

[10] A. Westfeld, "High capacity despite better steganalysis ," in Proc. Information Hiding: 4th International Workshop, IHW, vol. 2137 , pp. 289–302, Apr 2001.

[11] J. Fridrich, M. Goljan, D. Soukal, and T. Holotyak, "Forensic Steganalysis: Determining the Stego Key", submitted to IEEE Trans. on Sig. Proc., June 2004.

[12] M. Wu and B. Liu, "Data hiding for binary image for authentication and annotation," in IEEE Trans. on Multimedia, vol. 6, pp. 528–538, Aug 2004.