Correspondence

Fast Matrix Embedding by Matrix Extending

Chao Wang, Weiming Zhang, Jiufen Liu, and Nenghai Yu

***Abstract*—When designing steganographic schemes, matrix embedding is an efficient method for increasing the embedding efficiency that is defined as an average number of bits embedded via per change on the cover. Random linear code-based matrix embedding can achieve high embedding efficiency but cost much in computation. In this paper, we propose a method to increase the embedding speed of matrix embedding by extending the matrix via some referential columns. Compared with the original matrix embedding, the proposed method can exponentially reduce the computational complexity for equal increment of embedding efficiency. Experimental results also show that this novel method achieves higher embedding efficiency and faster embedding speed than previous fast matrix embedding methods, and thus is more suitable for real-time steganogaphic systems.**

***Index Terms*—Embedding efficiency, embedding rate, embedding speed, matrix embedding, steganography.**

I. Introduction

The purpose of steganography is to send secret messages after embedding them into public digital multimedia. It is desired to embed as many messages as possible per change of the cover-object. In general, for given messages and covers, the steganography that introduces fewer embedding changes will be less detectable, i.e., more secure. Matrix embedding, that is based on the parity check matrix of linear codes, is the most popular coding method to reduce the embedding changes.

The data embedding codes usually are measured by embedding efficiency versus embedding rate. Embedding rate is defined as the average number of bits embedded into each pixel, and embedding efficiency is defined as the average number of bits embedded by per embedding change. Although it is proven that random linear code-based matrix embedding can approach the theoretical upper bound of embedding efficiency, the encoder will suffer huge computational complexity. To apply matrix embedding with feasible complexity, Fridrich *et al.* [1] proposed random linear codes with small dimensions, which can achieve high embedding efficiency for only large embedding rates. However, codes for large embedding rates are important because the ZZW construction [2], [3] implies that small-embedding-rate codes can be generated from large-embedding-rate codes. What is more, the large payload matrix embedding in [1] can also be used to design ± 1 or ± 2 embedding schemes [4].

However, the computational complexity of the method in [1] is still high, and cannot reach real-time embedding speed that is needed in some stegnographic applications, e.g., embedding data into audio or video streams of instant communication systems [5]. Therefore, how to reduce the computational complexity of matrix embedding while keeping high embedding efficiency is a critical problem. Recent literature [6]–[8] proposes some fast matrix embedding algorithms. In [6], by employing a specific matrix, a secret message can be embedded with linear complexity. Li *et al.* [7] proposed a scheme to reduce embedding changes via a tree structure of the cover. The method in [7] can be formulated as another specific matrix embedding, which is improved by Hou *et al.* [8] with Majority-vote Parity Check (MPC). Compared with the original matrix embedding [1], these embedding methods [6]–[8] achieve fast embedding speed but at the cost of a sharp fall of embedding efficiency.

In the present paper, we propose a novel method to reduce the computational complexity of random linear code-based matrix embedding [1]. We refer to this new method as matrix extending because we design the fast algorithm by appending some referential columns to the parity check matrix. Analysis and experimental results show that the proposed method can flexibly trade embedding efficiency for embedding speed, or vice versa. Compared with the original matrix embedding [1], the proposed method can exponentially decrease computational complexity by increasing the number of the referential columns while achieving an equal increment of embedding efficiency. Compared with the fast matrix embedding methods in [6] and [8], the novel method can reach higher embedding efficiency with faster embedding speed.

The rest of this paper is organized as follows. Section II briefly introduces matrix embedding. The matrix extending method is described in Section III. Experimental results and analysis are presented in Section IV. The paper is concluded with a discussion in Section V.

II. Matrix Embedding

We will use boldface font for matrices and vectors, and denote the Hamming weight of vector $\mathbf{a}$ by $w(\mathbf{a})$ and the Hamming distance between two vectors $\mathbf{a}$ and $\mathbf{b}$ by $d(\mathbf{a}, \mathbf{b})$. The complement of one bit a denoted by $\bar{a}$ means flipping a from 0 to 1 or 1 to 0, and the complement of a vector $\mathbf{a}$ denoted by $\bar{\mathbf{a}}$ means flipping all elements of $\mathbf{a}$. We denote the all-zero vector having dimension n by $\mathbf{0}_n^T = (0, 0, \ldots, 0)$, and the all-one vector by $\mathbf{1}_n^T = (1, 1, \ldots, 1)$, where "$T$" means transpose.

In this paper, we directly assume the cover is a binary sequence, e.g., the least significant bits (LSBs) of gray values of pixels or the LSBs of quantized DCT coefficients. Denote the n-length cover block by $\mathbf{a}^T = (a_1, a_2, \ldots, a_n) \in GF^n(2)$. Because the message is usually encrypted before being embedded, it can be considered as a binary random sequence. Denote the m-length message block by $\mathbf{m}^T = (m_1, m_2, \ldots, m_m) \in GF^m(2)$, which is independent of the cover.

If we can embed m bits of message $\mathbf{m}$ into n bits of cover $\mathbf{a}$ by using R_a changes on average, we define embedding rate $\alpha = m/n$, and embedding efficiency $e = m/R_a$. In [1], it is proven that, for the given embedding rate, the embedding efficiency is upper bounded by

$$e(\alpha) \leq \frac{\alpha}{H_2^{-1}(\alpha)}, \quad 0 \leq \alpha \leq 1. \tag{1}$$

Manuscript received April 26, 2011; revised July 29, 2011; accepted July 29, 2011. Date of publication August 15, 2011; date of current version January 13, 2012. This work was supported by the Natural Science Foundation of China (60803155, 61170234), by the National Science and Technology Major Project of China (2010ZX03004-003), and by the High-Tech Research and Development Program of China "innovative high efficiency computer architectures and systems" (2009AA012201). The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Mauro Barni.

C. Wang and J. Liu are with the Department of Information Research, Zhengzhou Information Science and Technology Institute, Zhengzhou 450002, China.

W. Zhang is with the Department of Information Research, Zhengzhou Information Science and Technology Institute, Zhengzhou 450002, China, and also with the School of Information Science and Technology, University of Science and Technology of China, Hefei, 230026, China (e-mail: zwmshu@gmail.com).

N. Yu is with the School of Information Science and Technology, University of Science and Technology of China, Hefei, 230026, China.

Digital Object Identifier 10.1109/TIFS.2011.2164907

Herein, $H_2(x) = -x\log_2 x - (1-x)\log_2(1-x)$ is the binary entropy function.

The most popular data embedding code is called matrix embedding which is based on linear codes. Assume $m \times n$ matrix $\mathbf{H}$ is the parity check matrix of $[n, n-m]$ linear code, with which we can embed m bits of messages $\mathbf{m}^T = (m_1, m_2, \ldots, m_m)$ into n bits of cover $\mathbf{a}^T = (a_1, a_2, \ldots, a_n)$ by the following manner. First, calculate the difference between $\mathbf{Ha}$ and $\mathbf{m}$ with exclusive-or operation, i.e., $\mathbf{Ha} \oplus \mathbf{m}$. Second, solve the system of linear equations

$$\mathbf{Hx} = \mathbf{Ha} \oplus \mathbf{m} \tag{2}$$

to find a solution vector $\mathbf{x}_{\min}$ such that

$$\mathbf{x}_{\min} = \underset{\mathbf{x}\in GF^n(2), \mathbf{Hx}=\mathbf{Ha}\oplus\mathbf{m}}{\arg\min} w(\mathbf{x}). \tag{3}$$

Finally, the stego-object $\mathbf{b}$ is obtained by

$$\mathbf{b} = \mathbf{a} \oplus \mathbf{x}_{\min}. \tag{4}$$

The recipient can extract $\mathbf{m}$ by computing

$$\mathbf{Hb} = \mathbf{H}(\mathbf{a} \oplus \mathbf{x}_{\min}) = \mathbf{Ha} \oplus \mathbf{Hx}_{\min} = \mathbf{Ha} \oplus \mathbf{Ha} \oplus \mathbf{m} = \mathbf{m}. \tag{5}$$

To reduce the number of changes, the key problem is to search for a solution with minimum Hamming weight for the system of (2), which, however, is a computationally hard problem when $\mathbf{H}$ is a random matrix.

To solve (2) with feasible computational complexity, Fridrich *et al.* [1] proposed to only use random linear codes $[n, k]$ with small dimension k. The parity check matrix used in [1] has the following form:

$$\mathbf{H} = [\mathbf{I}_{n-k}, \mathbf{D}] \tag{6}$$

where $\mathbf{I}_{n-k}$ is an $(n-k)\times(n-k)$ unit matrix, and $\mathbf{D}$ is $(n-k)\times k$ random matrix. When using (6) as the coefficient matrix of the system of (2), the solution space consists of 2^k vectors, which can be looked through via all linear combinations of the k random columns of $\mathbf{D}$. The computational complexity for finding the solution with minimum Hamming weight is $O(n2^k)$. We call this method the original matrix embedding.

III. Matrix Embedding by Matrix Extending

In the original matrix embedding, when the number of random columns k increases, the solution space of (2) is exponentially expanded, and thus we have more chances to find a solution with smaller Hamming weight. That is why the embedding efficiency can be improved when k increases, but the computational complexity of searching for this solution exponentially grows. In this section, we propose a novel method, by which we can also exponentially expand the solution space, but only cost linearly increasing time to search the solution space. The key idea of the proposed method is to append some referential columns to the matrix (6).

The referential column is defined as a parity-check column of some columns of the unit matrix $\mathbf{I}_{n-k}$, and the referential column is equal to the exclusive-or of the corresponding columns. For the case of appending one referential column, the referential column is obtained by exclusive-or operation of all columns of the unit matrix $\mathbf{I}_{n-k}$, and, therefore, is just an all-one column.

A. Appending One Column

When appending only one referential column to the matrix $\mathbf{H}$ in (6), we use the all-one column $\mathbf{1}_{n-k}^T = (1, \ldots, 1)$, and get a new parity check matrix as follows:

$$\mathbf{E}_1 = [\mathbf{H}, \mathbf{1}_{n-k}] = [\mathbf{I}_{n-k}, \mathbf{D}, \mathbf{1}_{n-k}]. \tag{7}$$

Now we analyze the relation between the solution spaces of the following two system of equations:

$$\mathbf{Hx} = \mathbf{s} \tag{8}$$

$$\mathbf{E}_1\mathbf{y} = \mathbf{s}. \tag{9}$$

We divide one solution of (8), $\mathbf{x}$, into two parts such that

$$\mathbf{x}^T = (\mathbf{e}, \mathbf{d}) \tag{10}$$

where $\mathbf{e}$ is the first $n-k$ bits of $\mathbf{x}$, and $\mathbf{d}$ is the last k bits of $\mathbf{x}$. Thus we can construct two solutions of (9) from $\mathbf{x}$ as follows:

$$\mathbf{y}_1^T = (\mathbf{e}, \mathbf{d}, 0),\ \mathbf{y}_2^T = (\bar{\mathbf{e}}, \mathbf{d}, 1). \tag{11}$$

For $\mathbf{y}_1$, one referential bit "0" is appended, which means that the referential column $\mathbf{1}_{n-k}^T$ of $\mathbf{E}_1$ will not be added when computing (9), and thus (9) will hold if $\mathbf{x}$ is the solution of (8). When taking $\mathbf{y}_2$ into (9), the referential column will be added, but that can be canceled by taking the complement of $\mathbf{e}$, because the referential column is the parity check of the first $n-k$ columns of the matrix $\mathbf{H}$. The canceling process is shown as

$$\begin{aligned}\mathbf{E}_1\mathbf{y}_2 &= (\mathbf{I}_{n-k}, \mathbf{D}, \mathbf{1}_{n-k}) \cdot (\bar{\mathbf{e}}, \mathbf{d}, 1)^T \\ &= \mathbf{I}_{n-k}\bar{\mathbf{e}}^T \oplus \mathbf{Dd}^T \oplus \mathbf{1}_{n-k}1 \\ &= \mathbf{I}_{n-k}\mathbf{e}^T \oplus \mathbf{I}_{n-k}\mathbf{1}_{n-k} \oplus \mathbf{Dd}^T \oplus \mathbf{1}_{n-k} \\ &= \mathbf{I}_{n-k}\mathbf{e}^T \oplus \mathbf{Dd}^T \\ &= (\mathbf{I}_{n-k}, \mathbf{D})(\mathbf{e}, \mathbf{d})^T = \mathbf{Hx} = \mathbf{s}.\end{aligned} \tag{12}$$

On one hand, the solution space of (9) is twice as large as that of (8). On the other hand, from each solution of (8), we can construct two corresponding solutions of (9) by appending a "0" or appending a "1" and flipping the first $n-k$ bits as shown in (11). Thus we can find the minimal-weight solution of (9) by only exhausting the solution space of (8) in the following manner. For each solution $\mathbf{x}$ of (8), denote its weight by $w(\mathbf{x}) = w(\mathbf{e}) + w(\mathbf{d})$. With (11), the weights of two corresponding solutions of (9) are equal to $w(\mathbf{e}) + w(\mathbf{d})$ and $n - k - w(\mathbf{e}) + w(\mathbf{d}) + 1$, respectively. We only need to record the minimal weight and construct the corresponding solution by (11).

B. Appending h Columns

For the case of appending h referential columns, we divide the columns of $\mathbf{I}_{n-k}$ into h disjoint groups, and the ith referential column is equal to the exclusive-or of the ith column group. Therefore, the ith referential column has all-one at the ith segment and all-zero at other segments, which is used as the parity-check of the ith column group. Denote the size of the ith segment by t_i that satisfies $\sum_{i=1}^{h} t_i = n-k$. We hope each segment has equal size because the effect of the referential column is independent with its positions. In practice, We usually take t_i as

$$t_i = \begin{cases}\lfloor\frac{n-k}{h}\rfloor, & i = 1, \ldots, h-1 \\ (n-k) - (h-1)\lfloor\frac{n-k}{h}\rfloor, & i = h.\end{cases} \tag{13}$$

The ith referential columns $\mathbf{r}_i$ are constructed with the following form:

$$\mathbf{r}_i^T = (\mathbf{0}_{t_1}, \ldots, \mathbf{0}_{t_{i-1}}, \mathbf{1}_{t_i}, \mathbf{0}_{t_{i+1}}, \ldots, \mathbf{0}_{t_h}), 1 \le i \le h. \quad (14)$$

By appending these referential columns, we get the extending matrix

$$\mathbf{E}_h = \mathbf{H} \parallel \mathbf{r}_1 \parallel \cdots \parallel \mathbf{r}_h = [\mathbf{I}_{n-k}, \mathbf{D}, \mathbf{r}_1, \ldots, \mathbf{r}_h]. \quad (15)$$

On the other hand, we also need to divide the subvector $\mathbf{e}$ of the solution $\mathbf{x} = (\mathbf{e}, \mathbf{d})$ of (8) into h segments, that is

$$\mathbf{x} = (\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_h, \mathbf{d}) \quad (16)$$

where the dimension of $\mathbf{e}_i$ is equal to t_i.

When embedding $n-k$ bits of messages $\mathbf{m}$ into $n+h$ bits of cover $\mathbf{a}$ with $\mathbf{E}_h$, we should search the solution space of following system of equations:

$$\mathbf{E}_h \mathbf{y} = \mathbf{E}_h \mathbf{a} \oplus \mathbf{m}. \quad (17)$$

To do that, we only need to search the solution space of

$$\mathbf{H}\mathbf{x} = \mathbf{E}_h \mathbf{a} \oplus \mathbf{m}. \quad (18)$$

In fact, for any solution $\mathbf{x}^T = (\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_h, \mathbf{d})$ of (18), there are 2^h corresponding solutions of (17), having the form

$$\mathbf{y}^T = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_h, \mathbf{d}, r_1, r_2, \ldots, r_h) \quad (19)$$

where $\mathbf{f}_i = \mathbf{e}_i$ if $r_i = 0$, and $\mathbf{f}_i = \bar{\mathbf{e}}_i$ if $r_i = 1$. With the similar process as (12), we can verify the 2^h vectors constructed by (19) are solutions of (17).

In fact, setting the ith referential bit $r_i = 1$ means adding the ith referential columns, which is equivalent to adding the columns in the ith groups of $\mathbf{I}_{n-k}$. Therefore, (17) will hold when we set $r_i = 1$ and flip the bits in the ith segment of the solution of (18) simultaneously. In other words, when constructing solutions of (17) from one solution $\mathbf{x}$ of (18), for each segment $\mathbf{e}_i$ of $\mathbf{x}$, there are two cases according to $r_i = 0$ or 1, and we select the case with small Hamming weight.

In summary, the minimal Hamming weight of these 2^h solutions of (17) can be calculated by adding $w(\mathbf{d})$ to

$$\sum_{i=1}^{h} \min(w(\mathbf{e}_i), t_i - w(\mathbf{e}_i) + 1). \quad (20)$$

Based on the above method, the embedding and extraction processes are elaborated in Algorithm 1.

Algorithm 1 Matrix Embedding by Matrix Extending (ME&ME Algorithm)

Embedding Process

a) Generate an $n \times (n-k)$ parity check matrix $\mathbf{H}$ having the form (6), and extend it to a matrix $\mathbf{E}_h$ by appending h columns with (15).

b) Take an $(n+h)$-length cover block $\mathbf{a}$, and an $(n-k)$-length message block $\mathbf{m}$.

c) Search for all 2^k solutions of $\mathbf{H}\mathbf{x} = \mathbf{E}_h\mathbf{a} \oplus \mathbf{m}$, and write each solution with the form $\mathbf{x}^T = (\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_h, \mathbf{d})$, from which calculate a weight with (20). Thus we calculate 2^k weights in total, but only need to save the minimal weight and one corresponding solution denoted by $\mathbf{x}^{*T} = (\mathbf{e}_1^*, \mathbf{e}_2^*, \ldots, \mathbf{e}_h^*, \mathbf{d})$.

d) For $1 \le i \le h$, if $w(\mathbf{e}_i^*) \le t_i - w(\mathbf{e}_i^*) + 1$, let $\mathbf{f}_i^* = \mathbf{e}_i^*$ and $r_i = 0$; otherwise, let $\mathbf{f}_i^* = \bar{\mathbf{e}}_i^*$ and $r_i = 1$. Construct the minimal-weight solution $\mathbf{y}^*$ of (17) such that $\mathbf{y}^{*T} = (\mathbf{f}_1^*, \mathbf{f}_2^*, \ldots, \mathbf{f}_h^*, \mathbf{d}, r_1, r_2, \ldots, r_h)$.

e) Modify the cover block $\mathbf{a}$ to generate the stego block $\mathbf{b}$ such that $\mathbf{b} = \mathbf{a} \oplus \mathbf{y}^*$.

Extraction Process

Extract the message block $\mathbf{m}$ by calculating $\mathbf{m} = \mathbf{E}_h\mathbf{b}$.

Now we use a simple example with $h = 2$ to show the process of Algorithm 1.

Example 1: Take $n = 6, k = 2, t_1 = t_2 = 2$. Generate the original parity check matrix as

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \quad (21)$$

After appending two columns, the matrix $\mathbf{H}$ is extended as follows:

$$\mathbf{E}_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (22)$$

Suppose the cover block is $\mathbf{a}^T = (1,0,0,0,1,1,1,0)$, the message block to be embedded is $\mathbf{m}^T = (0,0,1,1)$. To solve

$$\mathbf{E}_2\mathbf{x} = \mathbf{E}_2\mathbf{a} \oplus \mathbf{m} \quad (23)$$

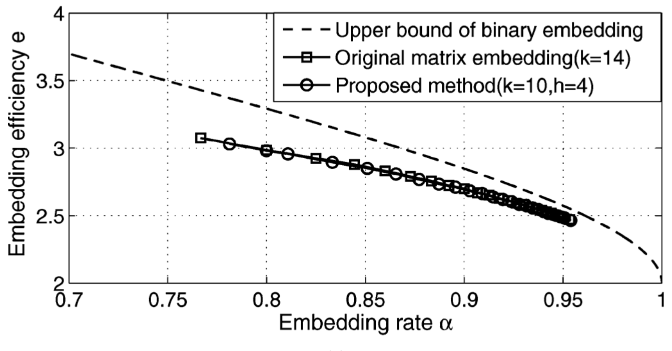
we first solve $\mathbf{H}\mathbf{x} = \mathbf{E}_2\mathbf{a} \oplus \mathbf{m}$, which has four solution vectors: $\mathbf{x}_1^T = (0,0,1,1,0,0)$, $\mathbf{x}_2^T = (0,0,0,1,1,0)$, $\mathbf{x}_3^T = (0,1,0,1,0,1)$, $\mathbf{x}_4^T = (0,1,1,1,1,1)$. By (20), we obtain four weights $\{1,2,3,4\}$, in which the minimal weight 1 is calculated from $\mathbf{x}_1$, i.e., $\mathbf{x}^* = \mathbf{x}_1$. By using the construction in Step d) of Algorithm 1, we get the minimal-weight solution of (23) such that $\mathbf{y}^{*T} = (0,0,0,0,0,0,0,1)$. Thus, by only one modification, we generate the stego block $\mathbf{b}^T = (\mathbf{a} \oplus \mathbf{y}^*)^T = (1,0,0,0,1,1,1,1)$. It is easy to verify that $\mathbf{m} = \mathbf{E}_2\mathbf{b}$.

IV. Experiment Results

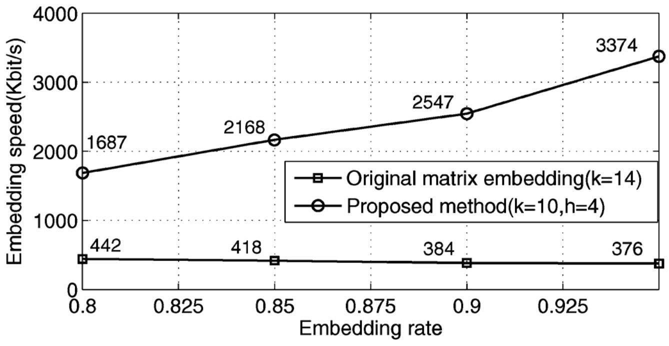
A. Comparison With Original Matrix Embedding

In Algorithm 1, by appending h referential columns, the size of the solution space is expanded from 2^k to 2^{k+h}; therefore, the expectation of Hamming weight of the minimal-weight solution will decrease, which will lead to increasing embedding efficiency. In the original matrix embedding[1], by adding h random columns, i.e., let the code dimension $k' = k + h$, we can also get a solution space with the same expanded size, and thus expect a similar increment of embedding efficiency, which, however, will make the computational complexity exponentially increase from $O(n2^k)$ to $O(n2^{k+h})$. In Algorithm 1, to find an optimal solution in the expanded solution space with a size of 2^{k+h}, we only need to search a small solution space with a size of 2^k, and do h comparisons and additions by (20) for each solution. Therefore, the computational complexity is equal to $O((n+h)2^k)$, which linearly increases with h, and thus Algorithm 1 can achieve equal embedding efficiency with faster embedding speed when compared with the original matrix embedding[1].

To verify the conclusion above, we embed messages into a random cover by using the original matrix embedding of [1] and Algorithm 1, respectively. For the method in [1], we take $k = 14$ and vary the cover block length n from 66 to 280 to get various embedding rates. For Algorithm 1, we take $k = 10, h = 4$, and also vary n from 66 to 280. For each embedding rate, we embed 1000 blocks of random messages, and

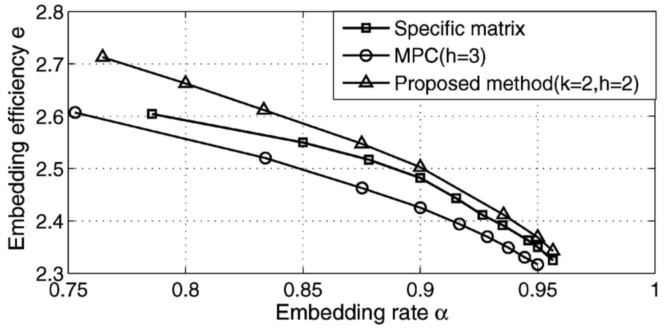


(a)

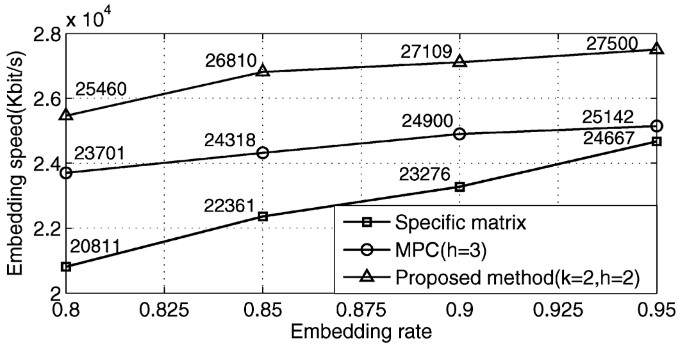


(b)

Fig. 1. Comparison between proposed method and the original matrix embedding [1]. (a) Embedding efficiency versus embedding rate. (b) Embedding speed versus embedding rate.



(a)



(b)

Fig. 2. Comparison between proposed method, specific matrix method [6], and MPC method [8]. (a) Embedding efficiency versus embedding rate. (b) Embedding speed versus embedding rate.

calculate embedding efficiency by the average number of changes. As shown in Fig. 1, the two methods achieve equal embedding efficiency [Fig. 1(a)], while the embedding speed of the proposed method significantly outperforms the original matrix embedding [Fig. 1(b)]. The embedding speed is measured by Kbits of covers per second for four kinds of embedding rates. The test was performed on Intel Core i5 running at 2.67 GHz with 4-GB RAM. The algorithm was implemented in C and compiled under Microsoft Visual Studio 2008.

B. Comparison With Fast Matrix Embeddings

We also compared Algorithm 1 with previous fast matrix embedding in [6] and [8]. Gao [6] *et al.* proposed a specific parity check matrix, which can embed messages with linear computational complexity according to the length of the cover block. The Majority-vote Parity Check (MPC) method [8] is an improving version of the Tree-Based Parity Check (TBPC) method [7], which embeds messages with linear computational complexity according to the length of the message block. The embedding efficiency of both methods in [6] and [8] is lower than the original matrix embedding with dimension $k = 4$. To compare with methods in [6] and [8], we take $k = 2$ and $h = 2$ in Algorithm 1. As shown in Fig. 2, the proposed method can reach higher embedding efficiency as well as faster embedding speed.

C. Discussion on Parameters

The most important parameter for Algorithm 1 is the number of referential columns h. Although embedding efficiency can be improved by increasing h, the improvement will stop when h is too large. Fig. 3 shows how embedding efficiency varies with h when taking $k = 4, 6$, and 8 at embedding rate $\alpha = 0.8$. The peak of embedding efficiency appears at $h = 8$ for $k = 4$, appears at $h = 6$ for $k = 6$, and appears at $h = 8$ for $k = 8$. Thus, we draw a conclusion that the maximum available h for improving embedding efficiency is limited by k. However, how to derive the general relation between h and k is still an open problem.

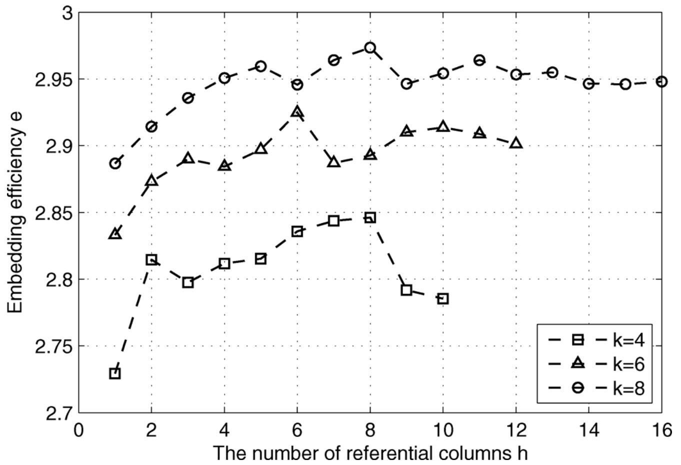


Fig. 3. Variation of embedding efficiency with increasing h for $k = 4, 6$, and 8 at embedding rate $\alpha = 0.8$.

The segment size t_i is the other important parameter used by Algorithm 1, in which we try to set uniform size for every segment by (13), because each referential column has equal effect that is independent with its position. In fact, adding referential columns in a flexible

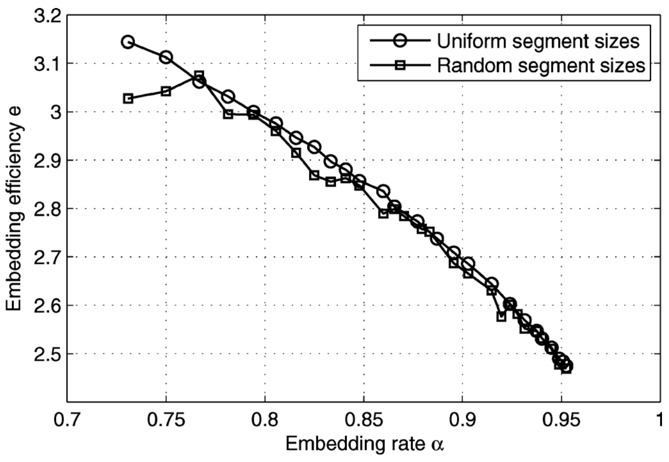


Fig. 4. Comparison of embedding efficiency obtained by two kinds of segment sizes.

manner will not improve the performance, which is illustrated by the following experiments. We take $k = 10$, $h = 4$, and embed messages with various embedding rates in two manners by setting uniform segment sizes with (13) and random segment sizes, respectively. As shown in Fig. 4, random segment sizes will decrease the embedding efficiency. On the other hand, from Algorithm 1, it is obvious that segment sizes have no effect on computational complexity.

V. Conclusion

In this paper, we proposed a fast matrix embedding method for data hiding by appending referential columns to the parity check matrix. The novel method can significantly reduce the computational complexity of the original matrix embedding [1], and achieve higher embedding efficiency and embedding speed than previous fast matrix embedding methods [6], [8]. By adjusting the parameters k and h, the user can flexibly make a trade-off between the embedding efficiency and embedding speed. This method can be used to design real-time steganographic system.

References

[1] J. Fridrich and D. Soukal, "Matrix embedding for large payloads," *IEEE Trans. Inf. Security Forensics*, vol. 1, no. 3, pp. 390–394, Sep. 2006.
[2] W. Zhang, X. Zhang, and S. Wang, "Maximizing steganographic embedding efficiency by combining Hamming codes and wet paper codes," in *Proc. 10th Information Hiding*, 2008, vol. 5284, LNCS, pp. 60–71.
[3] J. Fridrich, "Asymptotic behavior of the ZZW embedding construction," *IEEE Trans. Inf. Forensics Security*, vol. 4, no. 1, pp. 151–154, Mar. 2009.
[4] X. Zhang, "Efficient data hiding with plus-minus one or two," *IEEE Signal Process. Lett.*, vol. 17, no. 7, pp. 635–638, Jul. 2010.
[5] M. H. Shirali-Shahreza and S. Shirali-Shahreza, "Real-time and MPEG-1 layer III compression resistant steganography in speech," *IET Inf. Security*, vol. 4, no. 1, pp. 1–7, 2010.
[6] Y. Gao, X. Li, and B. Yang, "Constructing specific matrix for efficient matrix embedding," in *Proc. IEEE Int. Conf. Multimedia and Expo*, 2009, pp. 1006–1009.
[7] 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.
[8] C. Hou, C. Lu, S. Tsai, and W. Tzeng, "An optimal data hiding scheme with tree-based parity check," *IEEE Trans. Image Process.*, vol. 20, no. 3, pp. 880–886, Mar. 2011.