Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

PNA

Probability, Networks and Algorithms

Probability, Networks and Algorithms

Wavelet Techniques for Reversible Data Embedding into Images

Lute Kamstra, Henk J.A.M. Heijmans

REPORT PNA-R0402 MARCH 23, 2004

CWI is the National Research Institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organization for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-3711

Wavelet Techniques for Reversible Data Embedding into Images

ABSTRACT
The proliferation of digital information in our society has enticed a lot of research into data embedding techniques that add information to digital content like images, audio and video. This additional information can be used for various purposes and different applications place different requirements on the embedding techniques. In this paper, we investigate high capacity lossless data embedding methods that allow one to embed large amounts of data into digital images (or video) in such a way that the original image can be reconstructed from the watermarked image. The paper starts by briefly reviewing three existing lossless data embedding techniques as described by Fridrich and co-authors, by Tian, and by Celik and co-workers. We then present two new techniques: one based on least significant bit prediction and Sweldens' lifting scheme and another that is an improvement of Tian's technique of difference expansion. The various embedding methods are then compared in terms of capacity-distortion behaviour, embedding speed, and capacity control.

2000 Mathematics Subject Classification: 42C40, 68U10, 94A12
Keywords and Phrases: Reversible data embedding, Digital watermarking, Difference expansion, Lifting scheme, Haar wavelet
Note: This work was carried out under project PNA4.2 "Wavelets and Morphology". The research of the first author is sponsored (grant no. 613.006.570) by the Dutch Science Foundation (NWO).

Wavelet Techniques for Reversible Data Embedding into Images

Lute Kamstra and Henk J.A.M. Heijmans

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT

The proliferation of digital information in our society has enticed a lot of research into data embedding techniques that add information to digital content like images, audio and video. This additional information can be used for various purposes and different applications place different requirements on the embedding techniques. In this paper, we investigate high capacity lossless data embedding methods that allow one to embed large amounts of data into digital images (or video) in such a way that the original image can be reconstructed from the watermarked image.

The paper starts by briefly reviewing three existing lossless data embedding techniques as described by Fridrich and co-authors, by Tian, and by Celik and co-workers. We then present two new techniques: one based on least significant bit prediction and Sweldens' lifting scheme and another that is an improvement of Tian's technique of difference expansion. The various embedding methods are then compared in terms of capacity-distortion behaviour, embedding speed, and capacity control.

2000 Mathematics Subject Classification: 42C40, 68U10, 94A12.

Keywords and Phrases: Reversible data embedding, Digital watermarking, Difference expansion, Lifting scheme, Haar wavelet.

Note: This work was carried out under project PNA4.2 "Wavelets and Morphology". The research of the first author is sponsored (grant no. 613.006.570) by the Dutch Science Foundation (NWO).

1. Introduction

During the last decade, the availability of information in digital form has increased dramatically. Digital media have numerous advantages over analog media, such as higher quality, easy editing, lossless copying, and fast and efficient distribution. These advantages allow for new applications and enable new opportunities. For example, due to the Internet, publishing and accessing information has never been so easy. At the same time, these advantages of digital media pose problems too. For example, easy editing makes it difficult to determine the authenticity of documents and lossless reproduction combined with easy distribution has resulted in a dramatic increase of illegal copying of information.

But digital media also offer the possibility to embed additional data into the original media data in a way that is perceptually, and sometimes also statistically, undetectable. This data embedding potential can be exploited to build protection mechanisms against the threats mentioned before, or to provide additional functionalities. Today there is a huge literature (see e.g. [2, 3, 5, 10, 11] and the references mentioned there) on data embedding technologies such as digital watermarking and steganography. In this paper, we focus on *reversible data embedding*, also called *lossless data embedding*, which is a fragile technique in the sense that the embedded data will mostly be destroyed by small distortions of the image. Reversible data embedding allows one to embed a relatively large amount of data into an image in such a way that the original image can be reconstructed from the watermarked image. This makes it

an ideal technique for applications where one wants to store metadata directly into the image and where loss of quality is not always acceptable. Since the watermarked image resembles the original image closely, a legacy viewer that does not know how to reconstruct the original image can still be used to view the watermarked image.

The paper is organised as follows. Section 2 discusses some general aspect of reversible data embedding and list some important aspects of embedding techniques. Section 3 briefly reviews three existing reversible data embedding techniques as described by Fridrich and co-authors [7], by Tian [13], and by Celik and co-workers [4]. Section 4 introduces a novel reversible data embedding technique that utilises Sweldens' wavelet lifting scheme [12] in combination with least significant bit prediction. In Section 5, we present a variant of Tian's method that greatly improves[1] the performance of the original technique. We performed some experiments with implementations of the aforementioned embedding techniques to compare their capacity-distortion behaviour, their embedding speed, and their ability to control the capacity. The results of these experiments can be found in Section 6. Section 7 concludes with a discussion of the presented techniques and some ideas on future research.

Since we will often manipulate bits of integers in this paper, we introduce the following notation. For an integer h with binary representation $h_n \cdots h_1 h_0$, we write $h =_2 h_n \cdots h_1 h_0$. For the least significant bit (LSB) h_0 we introduce the notation $h_0 = \mathrm{LSB}(h)$. Flipping the last bit of h means replacing h_0 by $\bar{h}_0 = 1 - h_0$, and we denote the resulting h by $h^\bullet$. Furthermore, we denote by $h \bullet b$, where $b = 0, 1$, the integer that is obtained if the LSB h_0 is replaced by b, that is $h \bullet b =_2 h_n \cdots h_1 b$.

2. Reversible Data Embedding

Most existing data-embedding algorithms distort the original image in an irreversible manner and then one of the challenges is to minimise distortion against capacity. But there are various applications, e.g. in medical or military imaging, where any distortion, no matter how small, is intolerable. In such cases one has to take recourse to reversible data embedding methods. This means that the original image can be recovered after extraction of the embedded message (or watermark). That such reversible embedding is possible at all, is due to the fact that images usually possess strong spatial correlations. In fact, such correlations are exploited by compression algorithms such as JPEG, to reduce their size.

In practise, the main ingredient of a reversible data embedding algorithm is the introduction of an alternative representation which, as much as possible, decorrelates the data. In

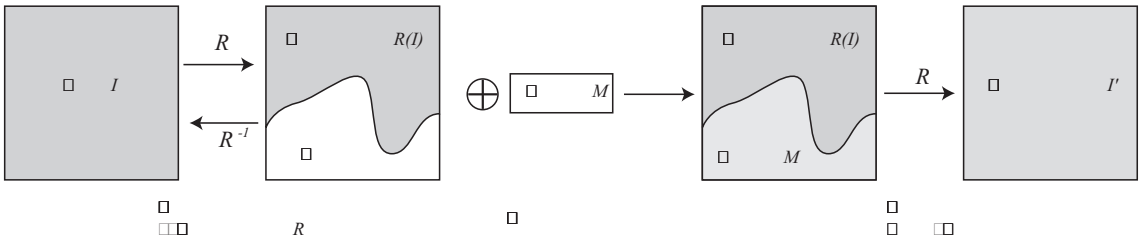

Figure 1: *General scheme of reversible data embedding.*

[1] That our modification of Tian's algorithm indeed is an improvement, follows from our experimental results in § 6.4.

Figure 1, this new representation manifests itself as a transform R with inverse R^{-1}. When applied to an image, it creates a certain amount of "free space" which can be used to insert the message M. After back-transformation to the image domain we obtain a modified image I'. Symbolically:

$$I' = R^{-1}(R(I) \oplus M).$$

Here the '$\oplus$' denotes the message embedding algorithm, which is assumed to be an invertible process with inverse denoted by '$\ominus$'. Thus we can reconstruct I at the decoder side once we have identified M:

$$I = R^{-1}(R(I') \ominus M).$$

Preferably, R is chosen in such a way that it creates a large free space, i.e., has a strong decorrelating effect. Furthermore, filling the free space with random values should not lead to great distortions in the reconstructed image.

High capacity and low distortion are conflicting requirements in practise: the higher the capacity that is created by a reversible embedding technique, the higher the distortion introduced by the embedding. Therefore embedding techniques are typically parametrised to allow for varying embedding capacities at various corresponding distortions. By changing this capacity parameter, one can construct a so-called *capacity-distortion curve* for the embedding method that shows the distortion introduced by using a certain capacity. The objective is to choose that value for the capacity parameter that results in just enough capacity to embed the desired payload as this will keep the distortion minimal. For some reversible data embedding techniques it is not possible to determine *a priori* which value to use for the parameter to achieve a certain capacity. As a result, the embedding algorithm has to be iterated for various values of the capacity parameter to determine the value that achieves the necessary capacity at the lowest distortion.

The literature on reversible data embedding is quite limited. Some interesting work is done by Fridrich and co-workers [7], by Celik *et al.* [4], and Tian [13]. They presented various practical techniques for doing reversible data embedding. The next sections gives a brief review of these techniques. Kalker and Willems [8, 9] have established some fundamental results regarding the capacity of reversible data embedding methods.

3. Existing Approaches

In this section we discuss three recent methods for reversible data embedding.

3.1 Fridrich et al

Fridrich and co-workers [7] were among the first authors to propose techniques for reversible data embedding. In this section we present a global description of their approach. Readers that are interested in all the technical details are referred to the original papers [7].

Consider a grey-scale image $I : [1, M_r] \times [1, M_c] \to \{0, 1, \ldots, 255\}$. Here M_r, M_c are the number of rows and columns, respectively. Subdivide the domain into L groups $G_1, G_2, \ldots, G_L$ of p pixels each. With every group $G = \{(i_1, j_1), \ldots, (i_p, j_p)\}$ we associate its *state* $s = s(G)$ given by

$$s = (s_1, \ldots, s_p) = (I(i_1, j_1), \ldots, I(i_p, j_p)).$$

Thus $s \in \Sigma = \{0, 1, \ldots, 255\}^p$, and Σ is called the *state space*. Assume that there exists a so-called *flipping operator* $\Phi : \Sigma \to \Sigma$ with the property that $\Phi^2 = \mathsf{id}$, where id is the identity

operator. When no confusion is possible about the choice of Φ, we write $\Phi(s) = s'$, and call s' the *adjoint state* of s. Obviously, s is the adjoint state of s' in this case as $s'' = s$.

Often, Φ is computed componentwise, i.e.,

$$\Phi(s_1, \ldots, s_p) = (\Phi(s_1), \ldots, \Phi(s_p)) \,, \tag{3.1}$$

where Φ maps $\{0, 1, \ldots, 255\}$ onto itself[2], with $\Phi^2 = \mathsf{id}$. By an *image model* we mean a disjoint partition of the state space Σ into three parts,

$$\Sigma = \Sigma_R \cup \Sigma_S \cup \Sigma_U \,, \tag{3.2}$$

with the property that

(i) $s \in \Sigma_R \iff s' \in \Sigma_S$;

(ii) $s \in \Sigma_U \iff s' \in \Sigma_U$.

We call $\Sigma_R,\ \Sigma_S,\ \Sigma_U$ respectively the *regular*, *singular*, and *undetermined states*. Similarly, the corresponding groups are called regular, singular, and undetermined. The key idea underlying this approach is that for a typical image with groups being created by some regular subdivision of the domain, most groups will be regular. Every image I corresponds with L groups $G_1, \ldots, G_L$, and the more of these groups are regular, the more I is said to obey the underlying image model. If all groups would be regular then we would be able to embed one bit b for each group, simply by replacing its state s by s' if $b = 1$ and maintaining s if $b = 0$. At the decoder we could simply read the embedded bits and flip all states that are singular. Unfortunately, in practise, not all groups will be regular, but nevertheless there will be more regular than singular groups. As we will see below, this fact can be exploited to embed data in a reversible manner.

The method by Fridrich *et al.* [7] has four ingredients that need to be discussed in more detail:

- the subdivision of the image domain into groups;
- the definition of the flipping operator Φ;
- the description of the image model, i.e., the definition of $\Sigma_R,\ \Sigma_S$, and Σ_U;
- the actual embedding of the data.

First, the image domain is subdivided into groups $G_1, \ldots, G_L$, each containing p pixels. For the time being we assume that the groups are disjoint and uniformly shaped, more precisely,

$$G_n = G + (i_n, j_n) = \{(i + i_n, j + j_n) \mid (i, j) \in G\} \,, \tag{3.3}$$

in other words, G_n is obtained by positioning a so-called *mask group* G with p pixels at the location (i_n, j_n). An important case, discussed in detail by Fridrich *et al.* [7] is the case that G consists of 4 consecutive pixels in a row. In Fig. 2 below, G is taken to be a 2×2 window. The arrangement of the centres (i_n, j_n) may be a pseudo-random sequence, generated by means of a secret key.

[2]Strictly speaking, the notation in (3.1) is ambiguous, but in practise it will not give rise to any confusion.

The flipping operator can be of the general form

$$\Phi(s_1, \ldots, s_p) = (\Phi_1(s_1), \Phi_2(s_2), \ldots, \Phi_p(s_p)) \,, \tag{3.4}$$

where Φ_k maps $\{0, 1, \ldots, 255\}$ onto itself with $\Phi_k^2 = \mathsf{id}$. Recall that $s' = \Phi(s)$ is called the adjoint state. A simple choice is $\Phi_k(s) = s^\bullet$, which flips the LSB of s. More generally, we can flip any combination of bits of s. Let $h \in \{0, 1, \ldots, 255\}$ and $h =_2 h_7 \cdots h_1 h_0$, we denote by ϕ_h the mapping on $\{0, 1, \ldots, 255\}$ that flips all the i'th bits where i is such that $h_i = 1$. For example, $\phi_5(s) =_2 s_7 \cdots s_3 \bar{s}_2 s_1 \bar{s}_0$. Note that $\phi_0(s) = s$ and $\phi_1(s) = s^\bullet$.[3]

In the case where $p = 4$ we can choose, for example,

$$\Phi(s) = (\phi_1(s_1), \phi_2(s_2), \phi_2(s_3), \phi_1(s_4)) \,.$$

For example, $\Phi(16, 18, 19, 15) = (17, 16, 17, 14)$ in this case. We point out that the definition in (3.4) is not the most general one. In fact, any partition of Σ into pairs of states and adjoint states yields a valid flipping operator. If Φ is componentwise identical like in (3.1), then we define the *amplitude* of the flipping operator by

$$A(\Phi) = \frac{1}{256} \sum_{s=0}^{255} |\Phi(s) - s| \,.$$

It is easy to verify that $A(\phi_k) = 2^l$ if $2^l \leq k < 2^{l+1}$.

The image model description is based on a regularity function $\mu : \Sigma \to \mathbb{R}_+$ which assigns to every state s a value describing the regularity or homogeneity of that state, in the sense that small $\mu(s)$ corresponds with a high regularity. Based on μ we can make a partition as given in (3.2) by putting

$$\begin{aligned} \Sigma_R &= \{s \in \Sigma \mid \mu(s) < \mu(s')\} \\ \Sigma_S &= \{s \in \Sigma \mid \mu(s) > \mu(s')\} \\ \Sigma_U &= \{s \in \Sigma \mid \mu(s) = \mu(s')\} \,. \end{aligned}$$

Thus a state s is said to be regular if it has higher homogeneity than its adjoint state s'. The same terminology will be used for groups, for example, a group will be called regular if its state lies in Σ_R.

The corresponding image model involves images that contain mostly regular groups. More precisely, the greater the percentage of regular groups, the better the image complies with the image model. The groups with undetermined states will not be used for embedding[4] and their presence diminishes the capacity of the embedding scheme, as we will see below.

Assume for example, that the mask group G is a 2×2 window like in Fig. 2. Now the states are of the form $S = (s_1, s_2, s_3, s_4)$ (from left to right and top to bottom), and we can take

$$\mu(s) = |s_1 - s_2| + |s_1 - s_3| + |s_2 - s_4| + |s_3 - s_4| \,.$$

For every group we can determine its state and we construct a binary bitstream $a_1 a_2 \cdots a_N$ where $N = N_R + N_S$ and N_R, N_S are the number of regular and singular groups, respectively,

[3] Observe that ϕ_h can also be easily expressed as a bitwise XOR operation: $\phi_h(s) =_2 (s_7 + h_7)(s_6 + h_6) \cdots (s_0 + h_0)$, where $+$ is the modulo 2 addition also known as the XOR operation.

[4] Fridrich *et al.* [7] speak of *unusable* rather than *undetermined* states.

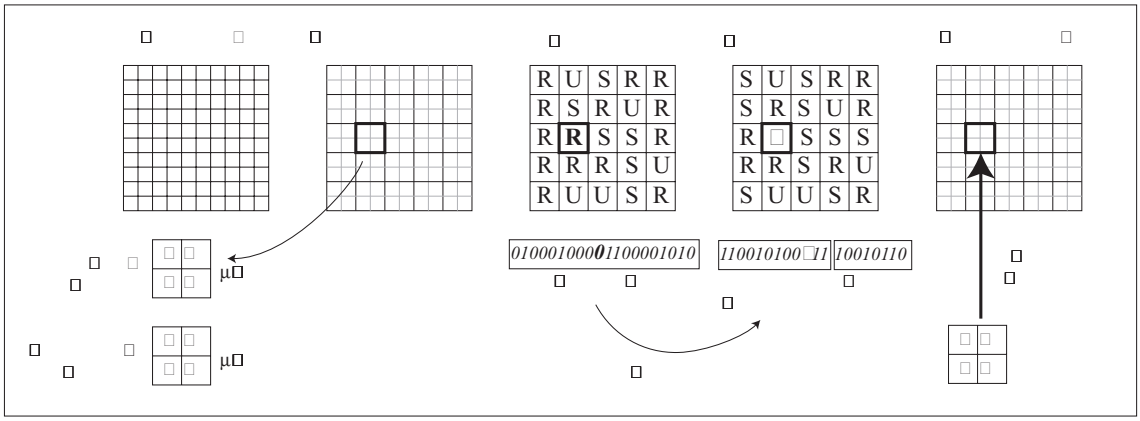

Figure 2: *A schematic example of the data embedding method of Fridrich* et al. *[7].*

and $a_k = 0$, resp. $a_k = 1$, if the k'th group in the sequence comprising only the regular and the singular groups is regular, resp. singular. In Fig. 2, for example, we have 25 groups, 5 of which are undetermined. Thus $N = 20$ in this case and the resulting binary bitstream, called *state bitstream*, comprises 20 bits. The better an image complies with an image model the larger the fraction $N_R/(N_R + N_S)$ of regular states, and the more 0's occur in the state bitstream. Now we can compress the state bitstream to a smaller size N_c and use the remaining bits $N - N_c$ to embed the message data. This yields a modified state bitstream $b_1 b_2 \cdots b_N$ of length N. This can be embedded into the image by flipping the states of the groups for which $a_k \neq b_k$, again skipping all the undetermined groups.

In the algorithm below, we denote by n the smallest number of groups that are needed to embed a message payload $\mathcal{P}$. That means that $\mathrm{cap}(n) \geq |\mathcal{P}|$, where $\mathrm{cap}(n)$ is the *net capacity* after n groups $G_1, \ldots, G_n$ have been scanned, that is,

$$\mathrm{cap}(n) = n - n_c\,,$$

where n_c is the length of the compressed state bitstream. In the algorithm below, we use an adaptive arithmetic coder denoted by AAC.

```
/* Determine states of consecutive groups */
/* This yields state bitstream a_1 a_2 ... a_n */
/* Compress until necessary capacity is reached */
/* Output is compressed state bitstream B = b_1 b_2 ... b_{n_c} */
move_to_first(G);
n = 0;
while cap(n) < |P| do
  n = n + 1;
  while s(G) ∈ Σ_U do
    move_to_next(G);
  end while
  if s(G) ∈ Σ_R then
    a_n = 0;
```

else
$a_n = 1$;
end if
AAC_encode(a_n);
move_to_next(G);
end while

/* Concatenate $\mathcal{P}$ to $\mathcal{B}$ */
$b_1 b_2 \cdots b_n = \mathcal{B} \cdot \mathcal{P}$;

/* Flip states of groups for which corresponding bit has changed */
move_to_first(G);
for $k = 1$ to n **do**
while $s(G) \in \Sigma_U$ **do**
move_to_next(G);
end while
if $a_k \neq b_k$ **then**
$s(G) = \Phi(s(G))$;
end if
move_to_next(G);
end for

3.2 Celik et al

In [4], Celik *et al.* present a reversible data embedding method which is based on a generalisation of the well-known LSB modification method. Instead of the standard binary LSB, one can extract the residual I_{res} of an image I as follows:

$$I_{\text{res}} = I - Q_p(I)\,, \tag{3.5}$$

where p is an integer and Q_p is the quantisation operator given by

$$Q_p(I) = p \cdot \lfloor I/p \rfloor\,, \tag{3.6}$$

which assumes values $0, 1, \ldots, p-1$. Note that for $p = 2$, the residual I_{res} is nothing but the standard LSB. Embedding is done simply by replacing the residual by a given watermark image W, resulting in a watermarked image I' given by

$$I' = Q_p(I) + W\,, \tag{3.7}$$

where W can only take integer values between 0 and $p-1$, and where I' does never exceed the maximum possible grey value. This means that

$$Q_p(I) = Q_p(I')\,,$$

and hence W can be extracted at the decoder by

$$W = I' - Q_p(I) = I' - Q_p(I')\,.$$

Assuming that the message payload is given by a binary string, one needs a binary-to-p-ary conversion map which is such that the values computed in (3.7) do never exceed the maximum

grey value. Towards that goal Celik *et al.* [4] introduce a conversion method which is basically a variant of the arithmetic coding algorithm.

In order to enable inversion at the decoder, one needs to include the original data contained in I_{res} in the embedded data. In other words, the data W to be embedded comprises both the residual I_{res} of the original signal and the message payload. Obviously, this requests that the residual data I_{res} is lossless compressed in order to create room for the message data. Celik *et al.* [4] use a dedicated algorithm for the residual data that comprises three main ingredients:

(*i*) Prediction: this uses the quantised image $Q_p(I)$ as side-information.

(*ii*) Context modelling and quantisation: again the quantised image is being used as side-information.

(*iii*) Conditional entropy coding.

In Celik *et al.* [4] these steps are discussed at length and we will not go any further into the matter here.

A serious problem in the whole approach is the capacity control problem. The approach adopted by Celik *et al.* is to vary the quantisation level p and the distortion caused by the embedding of W in order to achieve the desired capacity.

3.3 Tian's method

In [13] Tian introduces a high capacity method for reversible data embedding. Below it will be described for grey-scale images $I : [1, M_r] \times [1, M_c] \to [0, 255]$ but, as Tian observes, it can also be extended to colour images as well as audio and video data. The basic idea is to decompose I by an (integer) Haar wavelet into a low-frequency band L and a high-frequency band H. The actual data embedding takes place inside band H. Let h be a given coefficient inside band H with binary representation $h =_2 h_n \cdots h_1 h_0$. Rather than always replacing the least significant bit h_0 by a message bit b, most of the time b is added as the new least significant bit, i.e., $h' =_2 h_n \cdots h_1 h_0 b$, which is equivalent to $h' = 2h + b$. Such an operation is called *difference expansion*. We say that h is *expandable* if such a modification is possible without destroying the 'invertibility' of the Haar transform; see below for details.

We describe Tian's method in more detail. For the sake of brevity, some less important details are skipped here and we refer to Tian's original paper [13] for a complete description. For simplicity we restrict to a Haar transform in horizontal direction, but this can easily be extended to the more general case where the original is subdivided into pixel pairs in an arbitrary way. Let x, y be the values at two neighbouring pixels, i.e., $x = I(i, 2j),\ y = I(i, 2j + 1)$. The integer Haar transform maps the pair (x, y) onto another pair (l, h) given by

$$l = \left\lfloor \frac{x + y}{2} \right\rfloor \quad \text{and} \quad h = x - y\,. \tag{3.8}$$

Here $\lfloor \cdot \rfloor$ is the floor function, i.e., $\lfloor x \rfloor$ is the largest integer $\leq x$. Note that (3.8) can be inverted by means of

$$x = l + \left\lfloor \frac{h + 1}{2} \right\rfloor \quad \text{and} \quad y = l - \left\lfloor \frac{h}{2} \right\rfloor\,. \tag{3.9}$$

Thus the Haar transform maps the original image I onto a low-pass image L given by $L(i, j) = l$ and a high-pass image H with $H(i, j) = h$. Both images have half the width of the original image (we always assume even dimensions), and we denote their common domain by D.

As said, embedding takes place inside H, but this may result in x and/or y values reconstructed from (3.9) that lie outside the original range $[0, 255]$. It is easy to verify that (3.9) yields $x, y \in [0, 255]$ iff

$$|h| \leq 2(255 - l) \text{ and } |h| \leq 2l + 1\,, \tag{3.10}$$

for which we use the shorthand notation

$$h \in R(l)\,. \tag{3.11}$$

This *invertibility region* $R(l)$ is strongly dependent on l: if l is close to 0 or 255 this region is small, but if l is close to 128 it is large.

We distinguish two important subsets of D: the set $C \subseteq D$, called *changeable* locations, comprises all pixels (i, j) such that the LSB of $h = H(i, j)$ can be flipped without affecting invertibility, i.e., $h^\bullet \in R(l)$, where $l = L(i, j)$. The set $E \subseteq D$, called *expandable* locations, consists of pixels (i, j) such that another bit $b = 0, 1$ can be added to h (that is, h is replaced by $2h + b$) without destroying invertibility, i.e. $2h, 2h + 1 \in R(l)$. It is easy to check that E is a subset of C. If $H(i, j) = -1$ or 0 then $(i, j) \in C$ iff $(i, j) \in E$, and we denote this set of pixels by E_0. Thus we have

$$E_0 \subseteq E \subseteq C\,. \tag{3.12}$$

We can partition the domain D of L, H into four disjoint subsets:

$$D = E_0 \cup (E \setminus E_0) \cup (C \setminus E) \cup (D \setminus C)\,.$$

The first two subsets comprise all expandable pixels, the third subset contains pixels that are changeable but not expandable, and the last subset contains all pixels that are not changeable. A schematic illustration is given in Fig. 3. Data embedding takes place by a combination

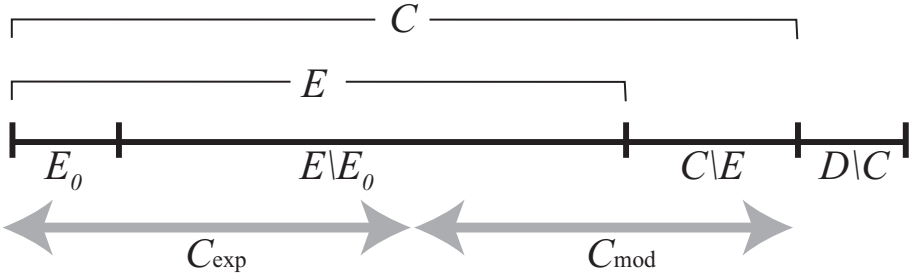

Figure 3: *Partition of the domain D.*

of expansion and modification. For expansion, one uses pixels in C_{exp}, which is a union of E_0 and a subset of $E \setminus E_0$. The latter set is chosen by means of some selection criterion; see below. Pixels in $C_{\text{mod}} = C \setminus C_{\text{exp}}$ are modified. To enable reconstruction of the original data at the decoder, one needs information about the subdivision of C into C_{exp} and C_{mod}. Towards that goal, Tian creates a binary image on D, the *location map*, that represents all locations that are selected for expansion. This location map is then compressed losslessly, e.g. by JBIG2 or run-length coding, and the resulting bitstream is denoted by $\mathcal{L}$.

Since the LSB's of difference values associated with locations in C_{mod} are overwritten, we need to insert these bits into the bitstream that is embedded in order to enable reconstruction. The bitstream formed by the LSB's of $H(i, j)$, with $(i, j) \in C_{\text{mod}}$, is denoted by $\mathcal{C}$.

Thus the overall bitstream that needs to be embedded is given by

$$\mathcal{B} = \mathcal{L} \cdot \mathcal{C} \cdot \mathcal{P}\,, \tag{3.13}$$

where $\mathcal{P}$ is the (message) payload, and '$\cdot$' denotes concatenation.

Assume that the bitstream $\mathcal{B}$ is given by $\mathcal{B} = b_1 b_2 \cdots b_m$, then the embedding algorithm looks as follows:

$(L, H) = \text{Haar\_transform}(I)$;
$(i, j) = (0, 0)$; `/* first pixel in` D `*/`
for $k = 1$ to m **do**
  while $(i, j) \notin C$ **do**
    $H'(i, j) = H(i, j)$;
    move_to_next (i, j);
  end while
  $h = H(i, j)$;
  if $(i, j) \in C_{\text{exp}}$ **then**
    $h' = 2h + b_k$;
  else
    $h' = h \bullet b_k$; `/*` $(i, j) \in C_{\text{mod}}$ `*/`
  end if
  $H'(i, j) = h'$;
  move_to_next (i, j);
end for
$I' = \text{inverse\_Haar\_transform}(L, H')$;

Embedding is indeed possible if

$$m = |\mathcal{L}| + |\mathcal{C}| + |\mathcal{P}| \leq |C|\,. \tag{3.14}$$

Since $|\mathcal{C}| = |C_{\text{mod}}|$ and $|C| = |C_{\text{mod}}| + |C_{\text{exp}}|$, the previous inequality can be rewritten as

$$|\mathcal{L}| + |\mathcal{P}| \leq |C_{\text{exp}}|\,. \tag{3.15}$$

The algorithm is based on the empirical observation that for natural grey-scale images most of the difference values h are expandable, i.e., between 90% and 99%. This gives rise to location maps that have high compression potential. Now a final issue is the selection of locations in $E \setminus E_0$ that will be used for expansion, i.e., the exact choice of C_{exp}. The guiding principle here is that small difference values h give rise to small distortions in x' and y' after expansion, whereas large difference values lead to large distortions. Tian considers two selection mechanisms. The first gives preference to small difference values h, i.e., it chooses locations (i, j) for which $|H(i, j)|$ is below a threshold T. Now a serious problem is that it is hard to estimate the capacity for a given T. This makes it difficult to choose T in such a way that embedding is possible, i.e., (3.14) is satisfied. The other selection mechanism is based on the so-called *hiding ability*. We refer to Tian's paper [13] for more information.

3.4 Other approaches

Vleeschouwer *et al.* [6] propose a completely different technique for reversible data embedding. It is based on a combination of the 'classical' *patchwork algorithm* [3] and the concept of a

circular histogram. Like in the patchwork approach, the image is partitioned into a given number of blocks, comprising, e.g., 8×8 pixels, and subsequently, each block is randomly subdivided into two subsets A and B of equal size. The two histograms associated with the luminance values of the pixels in A and B are mapped onto the unit circle. Now one can compute the two vectors, both located inside the unit disk, that point towards the respective centres of mass of both projected histograms. Embedding of message bits can be realised by rotation of both vectors in opposite ways. At the decoder one can extract the embedded bits and reverse the corresponding rotations. It is obvious that the capacity can never be larger than the number of blocks, and in fact, it will be smaller in practise since some of the blocks cannot be used for embedding. Vleeschouwer *et al.* [6] present several bench tests (but unfortunately, no tests relating capacity to distortion). Furthermore, they show that their method is robust under modest JPEG compression.

4. LSB prediction

The reversible data embedding technique presented by Fridrich and co-authors [7] has two drawbacks. Firstly, the capacity is at most 1 bit per group of pixels. Since the size of a group is typically four pixels, this means that the maximum capacity is 0.25 bits per pixels. Note that this maximum capacity is never attained due to the overhead of storing the compressed state bitstream. Secondly, groups need to be disjoint and no information of neighbouring groups can be used when creating the state bitstream and embedding bits into groups. In this section, we will present a novel technique that has a (theoretical) maximum capacity of 1 bit per pixel and can use information of neighbouring pixels during the embedding process. We will refer to this method as *least significant bit (LSB) prediction* since it utilises a wavelet-like decomposition of the image that involves predicting the least significant bitplane from the most significant bitplanes.

In this section we will use the binary addition, or exclusive or (XOR) operation on bits and matrices of bits. We will use the notation $+_2$ to refer to this addition. Recall that $(a +_2 b) +_2 b = a$ for all binary numbers $a, b \in \{0, 1\}$.

4.1 Global description of the algorithm

The main idea of the LSB prediction embedding technique is the same as in Fridrich's method: extract a binary bitstream from the image, compress it, and embed this compressed bitstream plus the payload by replacing the original bitstream. We will extract a binary bitstream by using a variant[5] of Sweldens' *lifting scheme* [12].

Consider a grey-scale image $I : [1, M_r] \times [1, M_c] \to \{0, 1, \ldots, 255\}$. Again, M_r, M_c are the number of rows and columns, respectively. We can split the image into the seven most significant bitplanes $M : [1, M_r] \times [1, M_c] \to \{0, 1, \ldots, 127\}$ and the least significant bitplane $L : [1, M_r] \times [1, M_c] \to \{0, 1\}$ as follows:

$$M(i, j) = \lfloor I(i, j)/2 \rfloor \,, \tag{4.1a}$$

$$L(i, j) = \mathrm{LSB}(I(i, j)) = I(i, j) - 2M(i, j) \,. \tag{4.1b}$$

Note that both M and L have the same size as the image I. Obviously, I can be reconstructed from L and M:

$$I(i, j) = L(i, j) + 2M(i, j) \,.$$

[5] The lifting scheme is usually applied differently. Normally, the data is split spatially before the scheme is applied, while here the data is split into bitplanes

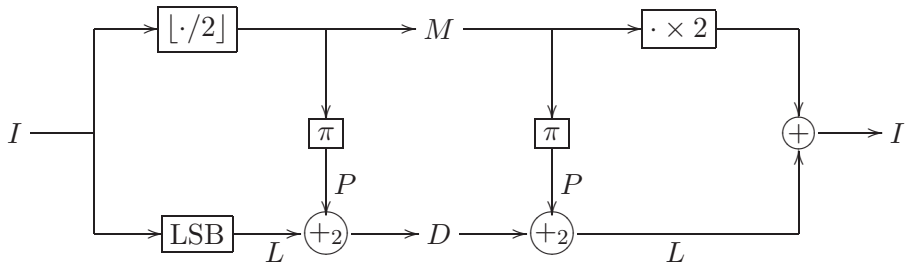

Figure 4: *Using the lifting scheme to construct the* (M, D) *representation of the image* I*.*

If we were able to compress the least significant bitplane L, then we could use this simple transform as the basis of a reversible embedding algorithm. Unfortunately, L is usually very hard to compress for natural images I. However, we can use the lifting scheme and predict the least significant bitplane L, using the seven most significant bitplanes M:

$$P = \pi(M). \tag{4.2}$$

This gives us a new representation (M, D) of the image, D being the difference $L +_2 P$ between the actual least significant bitplane L and the prediction P:

$$D(i,j) = L(i,j) +_2 P(i,j) = \begin{cases} 0 & \text{if } L(i,j) = P(i,j) \\ 1 & \text{if } L(i,j) \neq P(i,j). \end{cases} \tag{4.3}$$

The map $I \mapsto (M, D)$ is bijective, so we can reconstruct I from (M, D):

$$I = 2M + (D +_2 \pi(M)). \tag{4.4}$$

Fig. 4 shows this transform and its inverse.

So the better the prediction $\pi(M)$, the more zeros D will contain and the better D can be compressed. Note that the invertibility of the transform does not depend on π, so we can use *any* operator for π. More specifically, we can use the pixel values of M in any window $W(i,j)$ surrounding (i,j) to predict the least significant bit at (i,j). This is an advantage over Fridrich's method. For example, we can use the window:

$$W(i,j) = \{(i-1,j), (i,j), (i+1,j), (i,j-1), (i,j+1)\}, \tag{4.5a}$$

which gives us a prediction operator of the type:

$$\pi(M) = \pi_W(M(i-1,j), M(i,j), M(i+1,j), M(i,j-1), M(i,j+1)). \tag{4.5b}$$

We construct a prediction operator π that is based on the assumption that local extrema are more likely to be less pronounced. Many local extrema can be detected by means of the seven most significant bitplanes M. For convenience, we calculate

$$\begin{aligned} H_{\mathrm{W}}(i,j) &= \operatorname{sign}(M(i,j) - M(i,j-1)), \\ H_{\mathrm{E}}(i,j) &= \operatorname{sign}(M(i,j) - M(i,j+1)), \\ H_{\mathrm{N}}(i,j) &= \operatorname{sign}(M(i,j) - M(i-1,j)), \\ H_{\mathrm{S}}(i,j) &= \operatorname{sign}(M(i,j) - M(i+1,j)). \end{aligned}$$

Here the subindex 'W' in H_{W} refers to the fact that it concerns the difference with the neighbour at the 'West' (same for the other 3 neighbours). Furthermore, sign is the function defined by

$$\operatorname{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{4.6}$$

If, for example, $H_{\mathrm{W}}(i,j) = H_{\mathrm{E}}(i,j) = H_{\mathrm{N}}(i,j) = H_{\mathrm{S}}(i,j) = 1$, then M, and therefore also I, has a local maximum at location (i,j). In this case, we predict that the maximum is as small as possible and set $\pi(M)(i,j) = 0$. More generally, we define

$$\pi(M)(i,j) = \begin{cases} 0 & \text{if } (H_{\mathrm{W}}(i,j) + H_{\mathrm{E}}(i,j) > 0 \text{ and } H_{\mathrm{N}}(i,j) + H_{\mathrm{S}}(i,j) \geq 0) \\ 0 & \text{if } (H_{\mathrm{W}}(i,j) + H_{\mathrm{E}}(i,j) \geq 0 \text{ and } H_{\mathrm{N}}(i,j) + H_{\mathrm{S}}(i,j) > 0) \\ 1 & \text{if } (H_{\mathrm{W}}(i,j) + H_{\mathrm{E}}(i,j) < 0 \text{ and } H_{\mathrm{N}}(i,j) + H_{\mathrm{S}}(i,j) \leq 0) \\ 1 & \text{if } (H_{\mathrm{W}}(i,j) + H_{\mathrm{E}}(i,j) \leq 0 \text{ and } H_{\mathrm{N}}(i,j) + H_{\mathrm{S}}(i,j) < 0) \\ p & \text{otherwise,} \end{cases} \tag{4.7}$$

where p can be either 0 or 1, as our assumption does not allow us to predict the least significant bit in these cases.

Like with Fridrich's method, the embedding capacity can be controlled well. We can go through the pixels in D in some previously agreed on manner and start compressing bits by using an adaptive arithmetic coder (AAC) and continue encoding until the net capacity $\operatorname{cap}(n)$ has surpassed the size of the desired payload $|\mathcal{P}|$. We can then append the payload bits $\mathcal{P}$ to the compressed bitstream and replace this with the bits in D that were compressed.

In general, it is quite hard to construct prediction operators. In our experience, a good prediction operator π typically predicts 55% of the pixels correctly. Such a prediction accuracy results in a capacity-distortion behaviour that is comparable to Fridrich's method using a flipping operator that has an amplitude of 1. We can improve upon this by sorting the pixels in D according to an *estimate of the correctness of the least significant bit prediction* $\mu(M)$. So we not only use the most significant bitplanes M to predict the least significant bit at (i,j) by means of the operator π, but we also compute the likeliness $\mu(M)(i,j)$ that this prediction is correct. This allows us to sort the pixels in D according to our estimate of the correctness of the least significant bit prediction (the best predictions first). This will not change the global ratio of zeros and ones in D, but it will change this ratio locally: the ratio at the beginning of the bitstream will be more skewed than at the end. The simple adaptive arithmetic coder we used, does not perform well on these type of bitstreams, so we compress the bitstream block by block to ensure that the global ratio of zeros and ones (within a block) does not differ much from the local ratio. As a result the first blocks of the sorted bitstream can be compressed *much* better than the beginning of the unsorted bitstream, while the last blocks of the sorted bitstream can hardly be compressed. All in all, using such a sorting technique gives a better capacity-distortion behaviour, especially at small capacities.

We can quite easily construct a function μ associated with the prediction operator π defined

in (4.7):

$$\mu(M)(i,j) = \begin{cases} 2 & \text{if } (H_{\mathrm{W}}(i,j) + H_{\mathrm{E}}(i,j) > 0 \text{ and } H_{\mathrm{N}}(i,j) + H_{\mathrm{S}}(i,j) > 0) \\ 2 & \text{if } (H_{\mathrm{W}}(i,j) + H_{\mathrm{E}}(i,j) < 0 \text{ and } H_{\mathrm{N}}(i,j) + H_{\mathrm{S}}(i,j) < 0) \\ 1 & \text{if } (H_{\mathrm{W}}(i,j) + H_{\mathrm{E}}(i,j) \neq 0 \text{ and } H_{\mathrm{N}}(i,j) + H_{\mathrm{S}}(i,j) = 0) \\ 1 & \text{if } (H_{\mathrm{W}}(i,j) + H_{\mathrm{E}}(i,j) = 0 \text{ and } H_{\mathrm{N}}(i,j) + H_{\mathrm{S}}(i,j) \neq 0) \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

This definition stems from the rationale that if both directions agree on the type of extremum, then the prediction is likely to be better.

4.2 Encoding algorithm

The algorithm below describes the embedding of payload data $\mathcal{P}$ into an image I. To simplify matters, the block by block compression is not made explicit.

/* Calculate the (M, D) representation of the image I */
$M = \lfloor I/2 \rfloor$;
$L = \mathrm{LSB}(I)$;
$P = \pi(M)$; /* Predict the least significant bits */
$D = L +_2 P$;

/* Sort the locations using an estimate of the prediction quality */
$\mu =$ compute_correctness_measure(M);
sort_pixel_domain(μ); /* output is sorted list of pixels $(i_1, j_1), (i_2, j_2), \ldots$ */

/* Compress until the necessary capacity is reached */
/* Output is the bitstream $\mathcal{D}$ */
$n = 0$;
while $\mathrm{cap}(n) < |\mathcal{P}|$ **do**
  $n = n + 1$;
  AAC_encode$(D(i_n, j_n))$;
end while

/* Concatenate $\mathcal{D}$ and $\mathcal{P}$ to form the bitstream $\mathcal{B} = b_1 b_2 \cdots b_n$ */
$\mathcal{B} = \mathcal{D} \cdot \mathcal{P}$;

/* Embed the bitstream $\mathcal{B}$ into D and do the inverse transform */
/* Unused locations in D are not changed */
$D' = D$;
for $k = 1$ to n **do**
  $D'(i_k, j_k) = b_k$;
end for
$L' = D' +_2 P$;
$I' = 2M + L'$

4.3 Decoding algorithm

The algorithm below describes the extraction of an embedded payload $\mathcal{P}$ from an image I and the reconstruction of the original image I'. It assumes that the adaptive arithmetic coder stores some header information at the front of its output so that the decoder can retrieve the size of the compressed bitstream by means of the function AAC_size.

/* Calculate the (M, D) representation of the image I */
$M = \lfloor I/2 \rfloor$;
$L = \mathrm{LSB}(I)$;
$P = \pi(M)$; /* Predict the least significant bits */
$D = L +_2 P$;

/* Sort the locations using an estimate of the prediction quality */
$\mu = \mathrm{compute\_correctness\_measure}(M)$;
sort_pixel_domain(μ); /* output is sorted list of pixels $(i_1, j_1), (i_2, j_2), \ldots$ */

/* Read the sorted bitstream $\mathcal{B} = b_1 b_2 \cdots$ from D */
$\mathcal{B} = D(i_1, j_1) D(i_2, j_2) \cdots$

/* Decode the overwritten values of D */
/* Output is $\mathcal{D} = d_1 d_2 \cdots d_n$ */
/* Extract the payload as well */
$n_c = \mathrm{AAC\_size}(\mathcal{B})$
$\mathcal{D} = \mathrm{AAC\_decode}(b_1 b_2 \cdots b_{n_c})$
$n = |\mathcal{D}|$
$\mathcal{P} = b_{n_c+1} b_{n_c+2} \cdots b_n$

/* Restore the original values of D */
/* Apply inverse transform to reconstruct the original image */
$D' = D$;
for $k = 1$ to n **do**
  $D'(i_k, j_k) = d_k$;
end for
$L' = D' +_2 P$;
$I' = 2M + L'$

4.4 Extensions of LSB prediction

The LSB prediction method modifies only the least significant bitplane when embedding data. As such, the distortion is typically low. On the other hand, the maximum embedding capacity is low as well. To be able to use the LSB prediction method to embed larger payloads, we could use the generalised-LSB decomposition proposed by Celik *et al.* in [4]. This decomposes the image I into a quantised image $M = Q_p(I)$ and a residue image $L = I_{\mathrm{res}}$ as described by equations (3.5) and (3.6). When this decomposition is used, the theoretical maximum capacity is $\log_2(p)$ bits per pixel. The quality of the prediction π and the estimation of the correctness of the prediction μ will decrease, since they depend on M and M contains less information when p increases.

5. Improving Tian's Method

The approach of Tian [13] has two serious drawbacks. The first one concerns the capacity control problem: how to choose the set of locations so that embedding is possible but distortion is limited? The second problem concerns the overhead costs caused by the embedding of the (compressed) location map which covers all locations in D.

In this section we present an alternative of Tian's method which avoids the second pitfall and, at the same time, makes it easier to deal with the capacity control problem. The two main new ingredients are the following: (i) we use the low-pass image L to predict which locations in D will be expandable; (ii) the resulting bitstream which represents the correctness of the prediction is incrementally compressed using an arithmetic coder.

5.1 Global description of the algorithm

The first part of the algorithm is the same as in [13]: the image I is Haar-transformed into a low-pass and high-pass image L and H, respectively. Embedding is done by expanding or changing H-values. We have seen before that it is advantageous to choose for expansion those locations (i, j) for which $|H(i, j)|$ is small. Applying the inverse Haar transform yields a modified image I' which closely resembles the original image. This image I' is received by the decoder, whose task it is to extract the embedded message payload and to recover the original image I. Applying the Haar wavelet transform returns the images L and H', where the low-pass image L is the same as for the original image I. We exploit this fact by using L to predict the nature of the H-pixels, both at the encoder and at the decoder side. In fact, our modelling assumption is that the value of H at a location $(i, j) \in C$ is expected to be small if there is little variation of L in the vicinity of (i, j). Furthermore, such locations are likely to be expandable. More precisely, we define a *regularity measure* $\mu : C \to \mathbb{R}_+$ which measures the regularity or smoothness of L in the neighbourhood of (i, j) in the sense that the smaller $\mu(i, j)$, the more regular the image L near (i, j). For example, let $W(i, j)$ be a window surrounding (i, j), then we can define $\mu(i, j)$ as the local variance near (i, j), i.e.,

$$\mu(i,j) = \frac{1}{|W(i,j)|} \sum_{(i',j') \in W(i,j)} \big(L(i',j') - \bar{L}(i,j)\big)^2 \,, \tag{5.1}$$

where $\bar{L}(i, j)$ is the average of L inside $W(i, j)$. We can use the regularity measure μ to sort the locations in C into a list C_μ with ascending μ-values, i.e.,

$$C_\mu = \{(i_1, j_1), (i_2, j_2), \ldots\} \,,$$

where $\mu(i_k, j_k) \leq \mu(i_{k+1}, j_{k+1})$. Our model assumes that for natural images, locations at the beginning of the list are more likely to be expandable than locations that occur more towards the end. Moreover, H tends to be small at such locations. The order of embedding follows that of the list C_μ. Now the *location map* is described by a bitstream $a_1 a_2 \cdots$ where $a_k = 0$ if $(i_k, j_k) \in E$ and $a_k = 1$ if $(i_k, j_k) \in C \setminus E$. We call $a_1 a_2 \cdots$ the *location bitstream* and a_k the k'th *location bit*.

The aforementioned approach is likely to result in location bitstreams that contain mostly 0's (especially in the beginning) and only very few 1's, and that therefore allow strong lossless compression. We perform such compression by an adaptive arithmetic coder (AAC).

The details of the algorithm are discussed in the following section. Here we address the capacity control problem. The net capacity that results when the first n locations in C_μ

are used for embedding, is denoted by $\mathrm{cap}(n)$. The corresponding bitstream $\mathcal{B}(n)$ that is embedded consists of the same three parts as in in (3.13), that is

$$\mathcal{B}(n) = \mathcal{L}(n) \cdot \mathcal{C}(n) \cdot \mathcal{P}\,, \tag{5.2}$$

where $\mathcal{L}(n)$ is the encoded (i.e., compressed) location bitstream, $\mathcal{C}(n)$ are the bits that are overwritten, and $\mathcal{P}$ is the given payload. We call $\mathcal{C}(n)$ the *correction bitstream*.

We introduce some notation. First,

$$\kappa(n) = |\mathcal{L}(n)|$$

is the length of the bitstream $\mathcal{L}(n)$ that results from the AAC encoding of the location bitstream $a_1 a_2 \cdots a_n$. Furthermore,

$$\sigma_1(n) = |\mathcal{C}(n)| = \sum_{k=1}^{\kappa(n)} a_k$$

is the number of 1's in the first $\kappa(n)$ bits of the location bitstream. Note that the corresponding locations (i_k, j_k) for which $a_k = 1$, are changeable but not expandable. Since these bits are overwritten, they are stored in $\mathcal{C}(n)$, whence the second equality. Finally

$$\sigma_2(n) = \sum_{k=\kappa(n)+1}^{n} a_k$$

is the number of 1's in the remaining part of the location map. Locations $(i_k, j_k) \in C \setminus E$ (i.e., with $a_k = 1$) are useless for embedding as they are not expandable. In such cases, either the LSB of $H(i_k, j_k)$ is overwritten, in which case we have to add this bit to the bitstream $\mathcal{C}$ that is embedded, or location (i_k, j_k) is skipped. To minimise distortion, this last option is to be preferred, but skipping is only possible if the decoder is aware of the non-expandability of this location. This is only the case *after* the location bitstream has been decoded, i.e., if $k \geq \kappa(n)$.

Thus we conclude that

$$\sigma(n) = \sigma_1(n) + \sigma_2(n)$$

locations among the first n are ineffective. Therefore

$$\mathrm{cap}(n) = n - \kappa(n) - \sigma(n)\,,$$

and in order to have enough capacity to embed the payload $\mathcal{P}$, we choose the smallest n such that $|\mathcal{P}| \leq \mathrm{cap}(n)$, i.e.,

$$|\mathcal{P}| \leq n - \kappa(n) - \sigma(n)\,. \tag{5.3}$$

For natural images, the net capacity $\mathrm{cap}(n)$ is expected to increase with n. Obviously, it is negative for small n due to some overhead such as the header comprising κ_0 bits.

5.2 Encoding algorithm

For a good understanding of the algorithm it is useful to visualise the various ingredients of the bitstream $\mathcal{B}(n)$ in (5.2). This is done in Fig. 5. The output of our AAC, given that the first n bits $a_1 a_2 \cdots a_n$ of the location bitstream are inserted, consists of a fixed header

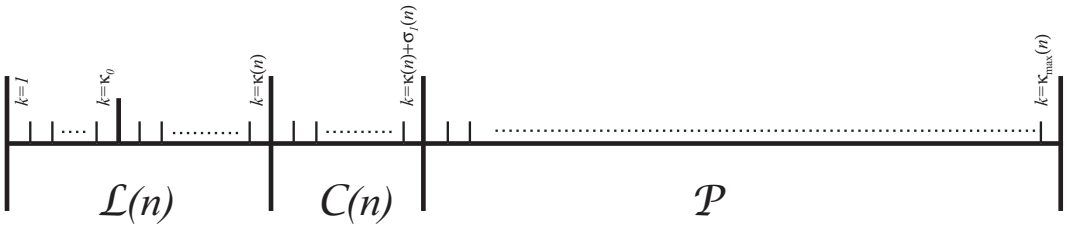

Figure 5: *The bitstream* $\mathcal{B}(n)$.

part $b_1 b_2 \cdots b_{\kappa_0}$ followed by a body part $b_{\kappa_0+1} \cdots b_{\kappa(n)}$ with length depending on n. Thus the compressed location bitstream is

$$\mathcal{L} = b_1 b_2 \cdots b_{\kappa_0} b_{\kappa_0+1} \cdots b_{\kappa(n)} \,.$$

The header is used to store the length of the body part, i.e., $\kappa(n) - \kappa_0$. Obviously, $\kappa(n) - \kappa_0 \leq |D|$ and we take $\kappa_0 = \lceil \log_2 |D| \rceil$. The compressed location bitstream $b_1 b_2 \cdots b_{\kappa(n)}$ is embedded in the first $\kappa(n)$ locations of C_μ, either by expansion (if $a_k = 0$) or by modification (if $a_k = 1$). In the latter case, the LSB of $H(i_k, j_k)$ has to be stored in order to enable inversion at the decoder. This results in a correction bitstream $\mathcal{C}(n)$ of length $\sigma_1(n)$. Then we concatenate the payload $\mathcal{P}$ to $\mathcal{C}(n)$ and embed $\mathcal{C}(n) \cdot \mathcal{P}$ at the remaining locations of C_μ, or rather $C_\mu \cap E$. In other words, locations for which $a_k = 1$ are skipped. The total length of $\mathcal{B}(n)$ is $\kappa_{\max}(n) = \kappa(n) + \sigma_1(n) + |\mathcal{P}|$, and using $|\mathcal{P}| \leq \mathrm{cap}(n)$ and (5.3), we get that $\kappa_{\max}(n) \leq n - \sigma_2(n)$. We end up with a modified high-pass image H', and now I' is found by taking the inverse Haar transform.

```
/* Compute Haar transform, regularity measure, and sorted list C_μ */
(L, H) = Haar_transform(I);
μ = compute_regularity_measure(L);
C_μ = sort(C, μ);      /* output is sorted list (i_1, j_1), (i_2, j_2), ... */

/* Calculate the location bitstream a_1 a_2 ··· a_|C| */
for k = 1 to |C| do
  if (i_k, j_k) ∈ E then
    a_k = 0;
  else
    a_k = 1;
  end if
end for

/* Compress location bitstream till required capacity is available */
/* Output bitstream is b_1 ··· b_κ(n), including header */
n = 0;
while cap(n) < |P| do
  n = n + 1;
```

```
    AAC_encode(a_n);
  end while

  /* Embed compressed location map by expansion and modification */
  /* Add overwritten bits to bitstream B */
  k = κ(n);
  for l = 1 to κ(n) do
    h = H(i_l, j_l);
    if a_l = 0 then
      h' = 2h + b_l;
    else
      h' = h • b_l;
      k = k + 1;
      b_k = LSB(h);
    end if
    H'(i_l, j_l) = h';
  end for

  /* Concatenate P to B */
  for l = 1 to |P| do
    k = k + 1;
    b_k = p_l;      /* p_l is l'th payload bit */
  end for
  κ_max = k;

  /* Embed C(n) · P by expansion; locations in C \ E are skipped */
  k = κ(n) + 1;
  l = κ(n) + 1;
  while k ≤ κ_max do
    while a_l = 1 do
      l = l + 1;      /* skip locations in C \ E */
    end while
    H'(i_l, j_l) = 2H(i_l, j_l) + b_k;
    k = k + 1;
  end while

  I' = inverse_Haar_transform(L, H');
```

5.3 Decoding algorithm

In the decoding algorithm, the modified image I' is used as input, and the output consists of the original image I along with the embedded payload $\mathcal{P} = p_1 p_2 \cdots p_{|\mathcal{P}|}$. In the first step we apply the Haar transform to I' which returns the lowpass image L, which is the same as for the original image, and the high-pass image which contains all the embedded data. We use L to compute the regularity measure μ, which is used to sort the locations in C in the same order as was done at the encoder resulting in the sorted list C_μ. Now we can read

the header $b_1 b_2 \cdots b_{\kappa_0}$ of $\mathcal{L}(n)$ which is used to compute the length $\kappa(n)$ of the compressed location bitstream; note, however, that n is unknown at this point. Then we read the body $b_{\kappa_0+1} \cdots b_{\kappa(n)}$ of $\mathcal{L}(n)$, which, after AAC-decoding, gives us the location bitstream $a_1 a_2 \cdots a_n$. Using this bitstream, we can read the remaining bits of $\mathcal{B}(n)$, that is, $\mathcal{C}(n) \cdot \mathcal{P}$, and restore the original H-values by means of $h = \lfloor h'/2 \rfloor$. Since we can dispose of the location map, we are able to skip locations (i_k, j_k) in $C \setminus E$, characterised by $a_k = 1$. In a next step we restore the H-values at locations $k = 1, 2, \ldots, \kappa(n)$: if $a_k = 0$ then $h = \lfloor h'/2 \rfloor$, otherwise $h = h' \bullet b$, where b was the bit overwritten by the encoder and stored in the correction bitstream $\mathcal{C}(n)$. The remaining part of the bitstream is then $\mathcal{P}$, the message payload that was embedded by the encoder. Applying the inverse Haar transform to L and H returns the original image I.

```
/* Compute Haar transform, regularity measure, and sorted list C_μ */
(L, H') = Haar_transform(I');
μ = compute_regularity_measure(L);
C_μ = sort(C, μ);      /* output is sorted list (i_1, j_1), (i_2, j_2), ... */

/* Read header of L(n) and compute κ_n */
/* κ_n is the same as κ(n) but n is yet undetermined */
for k = 1 to κ_0 do
   b_k = LSB(H'(i_k, j_k));
end for
κ_n = Σ_{k=0}^{κ_0} b_k 2^k;

/* Read body of L(n) */
/* AAC_decode location bitstream a_1 ··· a_n */
for k = κ_0 + 1 to κ_n do
   b_k = LSB(H'(i_k, j_k));
end for
AAC_decode(b_{κ_0+1} ··· b_{κ_n});

/* Read C(n) · P */
/* Restore original H-values at locations with k > κ(n) */
k = κ_n + 1;      /* k corresponds with the first C-bit */
for l = κ_n + 1 to n do
   if a_l = 0 then
      k = k + 1;
      b_k = LSB(H'(i_l, j_l));
      H(i_k, j_k) = ⌊H'(i_l, j_l)/2⌋;
   end if
end for
κ_max = k;

/* Restore original H-values at locations with k ≤ κ(n) */
k = κ_n + 1;      /* k corresponds with the first C-bit */
for l = 1 to κ_n do
```

if $a_l = 0$ **then**
$H(i_l, j_l) = \lfloor H'(i_l, j_l)/2 \rfloor$;
else
$H(i_l, j_l) = H'(i_l, j_l) \bullet b_k$;
$k = k + 1$;
end if
end for

/* k has reached position $\kappa(n) + \sigma_1(n) + 1$ in Fig. 5 */
/* Put remaining bits b_k into $\mathcal{P}$ */
$t = 1$;
while $k \leq k_{\max}$ **do**
$p_t = b_k$;
$t = t + 1$;
$k = k + 1$;
end while

$I = \text{inverse\_Haar\_transform}(L, H)$;

6. Experimental Results

We implemented the reversible data embedding method of Fridrich *et al.*, Tian's method, our method of LSB prediction, and our improvement of Tian's method. Due to its complexity, we did not implement the method of Celik *et al.* We tested the four methods we implemented on each of the four grey-level images shown in Fig. 6. Since Celik and co-authors also use these four images in their paper [4], one should be able to compare their results with ours. This section first discusses the implementation details of the various data embedding methods and then compares them by considering their capacity versus distortion (PSNR) behaviour, their ability to control the embedding capacity and their embedding time.

Barbara F-16 Lena Mandrill

Figure 6: *The four* 512×512 *test images.*

We implemented all the reversible data embedding methods as MATLAB functions. In the following subsections we discuss some of the details of these implementations.

6.1 Implementation of method by Fridrich et al. [7]

Our implementation of the method of Fridrich and co-workers [7] uses a mask group G of 4×1 pixels; see (3.3). We tested the method for a number of flipping operators with various

amplitudes. More precisely, for amplitudes $a = 1, 2, \ldots, 8$, we used the flipping operator

$$\Phi_a(s_1, s_2, s_3, s_4) = (\phi_a(s_1), \phi_a(s_2), \phi_a(s_3), \phi_a(s_4)), \tag{6.1a}$$

with

$$\phi_a(s) = \begin{cases} s + a & \text{if} \quad 0 \leq (s \bmod 2a) < a \quad \text{and} \quad s + a \leq 255 \\ s & \text{if} \quad 0 \leq (s \bmod 2a) < a \quad \text{and} \quad 255 < s + a \\ s - a & \text{if} \quad a \leq (s \bmod 2a) < 2a. \end{cases} \tag{6.1b}$$

Note that the amplitude of Φ_a is exactly a if $2a$ divides 256; otherwise the amplitude is slightly less than a:

$$\begin{aligned} A(\Phi_a) = A(\phi_a) &= \frac{1}{256} \sum_{s=0}^{255} |\phi_a(s) - s| \\ &= \begin{cases} \dfrac{256 - (256 \bmod 2a)}{256} \cdot a & \text{if} \quad 0 \leq (256 \bmod 2a) \leq a \\ \dfrac{256 - (2a - (256 \bmod 2a))}{256} \cdot a & \text{if} \quad a \leq (256 \bmod 2a) < 2a \end{cases} \\ &\leq \frac{256 - a}{256} \cdot a. \end{aligned}$$

Table 1 lists the actual amplitudes of $\Phi_1, \Phi_2, \ldots, \Phi_8$.

To determine which groups are regular, singular and undetermined, we use the regularity function

$$\mu(s_1, s_2, s_3, s_4) = |s_1 - s_2| + |s_2 - s_3| + |s_3 - s_4|. \tag{6.2}$$

Table 1 also shows the number of regular, singular, and undetermined groups for each of the four test images and for each flipping operator. Note that the percentage of undetermined groups –groups that we cannot use to embed data– is quite large for all images and flipping operators. Observe as well that the ratio of regular and singular groups gets better for increasing amplitudes.

Like Fridrich, we used an adaptive arithmetic coder to compress the state bitstream.

6.2 Implementation of method by Tian [13]

Our implementation of Tian's original method uses a horizontal pairing of pixels resulting in a Haar transform in the horizontal direction. For Tian's algorithm and our improvements of it, the capacity versus distortion behaviour depends on the number of expandable, changeable and non-changeable pairs. For each of the four test images, these numbers are listed in Table 2.

Tian presented two methods for selecting C_{exp}, the locations to use for difference expansion. In our experiments, we used the one that minimises the mean square error introduced by the embedding. This method gives the best results when PSNR is used to measure distortion. In his experiments, Tian used a JBIG2 encoder to compress the location map. Since we did not have such an encoder at our disposal, we used an adaptive arithmetic coder in our implementation of Tian's original method. This compression method gives slightly worse results, but the difference is rather marginal.

Flipping operator		Φ_1	Φ_2	Φ_3	Φ_4	Φ_5	Φ_6	Φ_7	Φ_8
Amplitude		1	2	2.98	4	4.92	5.91	6.89	8
Barbara	Regular	39.6%	48.8%	55.6%	59.4%	61.2%	61.8%	61.9%	61.1%
	Singular	24.8%	20.8%	17.6%	15.1%	13.2%	11.8%	10.3%	9.6%
	Ratio R/S	1.60	2.35	3.16	3.94	4.65	5.24	5.99	6.36
	Undetermined	35.6%	30.5%	26.8%	25.5%	25.6%	26.4%	27.8%	29.3%
F-16	Regular	43.9%	56.6%	60.5%	61.0%	60.3%	58.3%	56.7%	53.9%
	Singular	17.5%	13.0%	10.2%	8.1%	6.8%	5.9%	4.7%	4.2%
	Ratio R/S	2.51	4.35	5.94	7.50	8.90	9.97	12.06	12.77
	Undetermined	38.6%	30.4%	29.3%	30.9%	32.9%	35.8%	38.5%	41.8%
Lena	Regular	42.1%	53.8%	61.7%	66.0%	67.6%	67.3%	65.6%	63.4%
	Singular	23.4%	17.8%	13.4%	10.4%	8.1%	6.4%	5.1%	4.1%
	Ratio R/S	1.80	3.02	4.60	6.36	8.39	10.59	12.98	15.43
	Undetermined	34.5%	28.4%	24.9%	23.6%	24.4%	26.4%	29.3%	32.5%
Mandrill	Regular	37.6%	41.6%	45.6%	49.1%	52.1%	54.7%	56.7%	58.2%
	Singular	32.0%	30.1%	28.2%	26.2%	24.6%	23.1%	21.9%	20.7%
	Ratio R/S	1.17	1.38	1.62	1.88	2.12	2.36	2.59	2.82
	Undetermined	30.4%	28.3%	26.2%	24.7%	23.3%	22.2%	21.4%	21.1%

Table 1: *The percentage of regular, singular, and undetermined* 4×1 *groups for each test image and each flipping operator* Φ_a *as defined in* (6.1a) *and* (6.1b) *when using the regularity function* μ *defined in* (6.2). *The table additionally shows the ratio of regular and singular groups for each image and flipping operator, as well as the amplitude of each flipping operator.*

	E_0		$E \setminus E_0$		$C \setminus E$		$D \setminus C$	
Barbara	16094	(12.3%)	113548	(86.6%)	1430	(1.1%)	0	(0.0%)
F-16	36658	(28.0%)	94398	(72.0%)	16	(0.0%)	0	(0.0%)
Lena	21893	(16.7%)	109173	(83.3%)	6	(0.0%)	0	(0.0%)
Mandrill	8916	(6.8%)	122065	(93.1%)	90	(0.1%)	1	(0.0%)

Table 2: *The number of expandable pairs with* $h \in \{-1, 0\}$ *(*E_0*), the number of expandable pairs with* $h \notin \{-1, 0\}$ *(*$E \setminus E_0$*), the number of changeable, but not expandable pairs (*$C \setminus E$*), and the number of non-changeable pairs (*$D \setminus C$*) in the four test images.*

6.3 Implementation of LSB prediction method

We tested a number of different methods of predicting the least significant bitplane and estimating the correctness of the prediction. In Section 4.1 we presented one particular example of a prediction operator π and correctness measure μ. Here we present another pair π, μ which performs well for our test images.

The prediction operator π is based on a five pixel window $W(i,j) = \{(i-1,j),(i+1,j),(i,j),(i,j-1),(i,j+1)\}$:

$$\pi(M)(i,j) = \begin{cases} 0 & \text{if } \bar{M}(i,j) \leq M(i,j) \\ 1 & \text{if } \bar{M}(i,j) > M(i,j), \end{cases} \tag{6.3}$$

where $\bar{M}(i,j)$ is the average of M inside the window $W(i,j)$. So this operator uses the average (of the most significant bits) of the neighbouring pixels in $W(i,j)$ to predict the least significant bit at (i,j).

The aforementioned prediction operator works good in combination with a correctness estimate operator μ that is based on a combination of the local variance and a number that expresses to which extent pixels in $W(i,j)$ agree on their prediction of the least significant bit at (i,j). The local variance is defined as

$$v(M)(i,j) = \frac{1}{|W(i,j)|} \sum_{(i',j') \in W(i,j)} \left(M(i',j') - \bar{M}(i,j)\right)^2. \tag{6.4a}$$

The idea is that the quality of the prediction of the least significant bit is better at locations where the local variance is small. We define the *agreement number* as

$$a(M)(i,j) = \left| \sum_{(i',j') \in W(i,j)} \operatorname{sign}\left(M(i',j') - M(i,j)\right) \right|, \tag{6.4b}$$

where the sign operator is defined as is (4.6). The agreement number is highest if all neighbours of $M(i,j)$ agree on how to predict the least significant bit in the sense that they are all larger or all smaller than $M(i,j)$. The agreement number is smallest if half the neighbours are smaller and the other half are larger. The idea behind this definition is that the prediction is better if the agreement number is higher. In our experiment, an additive combination of the local variance and the agreement number performed well. More specifically, we used the measure μ defined by

$$\mu(M)(i,j) = 10a(M)(i,j) - v(M)(i,j). \tag{6.4c}$$

Table 3 shows how well the prediction operator in (6.3) π performs by listing the ratio of ones and zeros in D for each of the test images. Is also demonstrates the performance of the correctness measure μ by showing the ratio of the first 10% of the sorted bits in D.

For the LSB prediction method, we used the same adaptive arithmetic coder to compress the bitstream $\mathcal{D}$ as the one used in the implementation of Fridrich's method. To compensate for the changing characteristics in the sorted bitstream, we compress the bitstream blockwise using blocks of approximately 10.000 bits (4% of the total bitstream $\mathcal{D}$).

6.4 Implementation of improved Tian method

When testing our improvement of Tian's method, we used a number of different regularity measures μ to select pixel pairs for expansion. Although the different measures performed

	10%	100%
Barbara	0.3494	0.4488
F-16	0.2425	0.3997
Lena	0.3518	0.4349
Mandrill	0.4409	0.4795

Table 3: *The average number of ones after selecting 10% and 100% of the sorted bits using the LSB prediction method for each of the four test images.*

slightly different for a given image, there was no a single measure that outperformed all other measures for most images. We will therefore show only the results for one regularity measure, namely $\mu : C \to \mathbb{R}_+$ as defined in (5.1) with the cross-shaped window

$$W(i,j) = \{(i-1,j),(i,j),(i+1,j),(i,j-1),(i,j+1)\}.$$

This measure performed well for most images.

6.5 Capacity versus distortion behaviour

Fig. 7, 8, 9, and 10 show the overall picture of the capacity versus distortion performance of the discussed reversible data embedding methods for each image. Since the capacity-distortion curves of Fridrich's method and our LSB prediction method are located in a relatively small area, this area is enlarged in Fig. 11, 12, 13, and 14.

It is clear that Tian's method and our improvement of it can achieve much higher embedding capacities than the other two methods, namely up to 0.5 bits per pixel. When the embedding process is repeated for the vertical direction, the methods can even achieve capacities close to 1 bit per pixel. Except for the Barbara image, Tian's method and its improvement have identical maximal capacities. Moreover, they achieve these maximal capacities at virtually the same distortions. This is due to the fact that Tian's method achieves maximum capacity when all expandable pairs are selected for expansion. As is shown in Table 2, virtually all pairs are expandable so that the corresponding location map can be compressed very well. In this case, the overhead of Tian's original method compared with our modification becomes negligible.

The overhead of having to keep a location map of pairs that are used for expansion is quite well visible in Fig. 7, 8, 9, and 10. Whereas Tian's original method achieves a positive capacity when a large fraction of the pixel pairs are used for expansion, our improvement only needs less then 1% of the pairs. See Table 4 for the exact amounts for each test image. As a result, Tian's original method is not capable of embedding small payloads at low distortions, whereas our improvement clearly is. For the Mandrill image, the distortion difference at 0.05 bit per pixel is well over 10 dB.

The bad performance of Tian's original algorithm at low capacities makes Fridrich's method and our method of LSB prediction better alternatives. As can be seen in Fig. 11, 12, 13, and 14, these two methods typically outperform Tian's method below 0.02–0.05 bits per pixel. If the LSB prediction method uses no sorting, its capacity versus distortion behaviour is comparable to Fridrich's method with a flipping function of amplitude 1. When sorting is used, LSB prediction clearly outperforms Fridrich's method.

Although both Fridrich's method and our method of LSB prediction outperform Tian's original method at low capacities, our improvement of Tian's method has the best capacity-

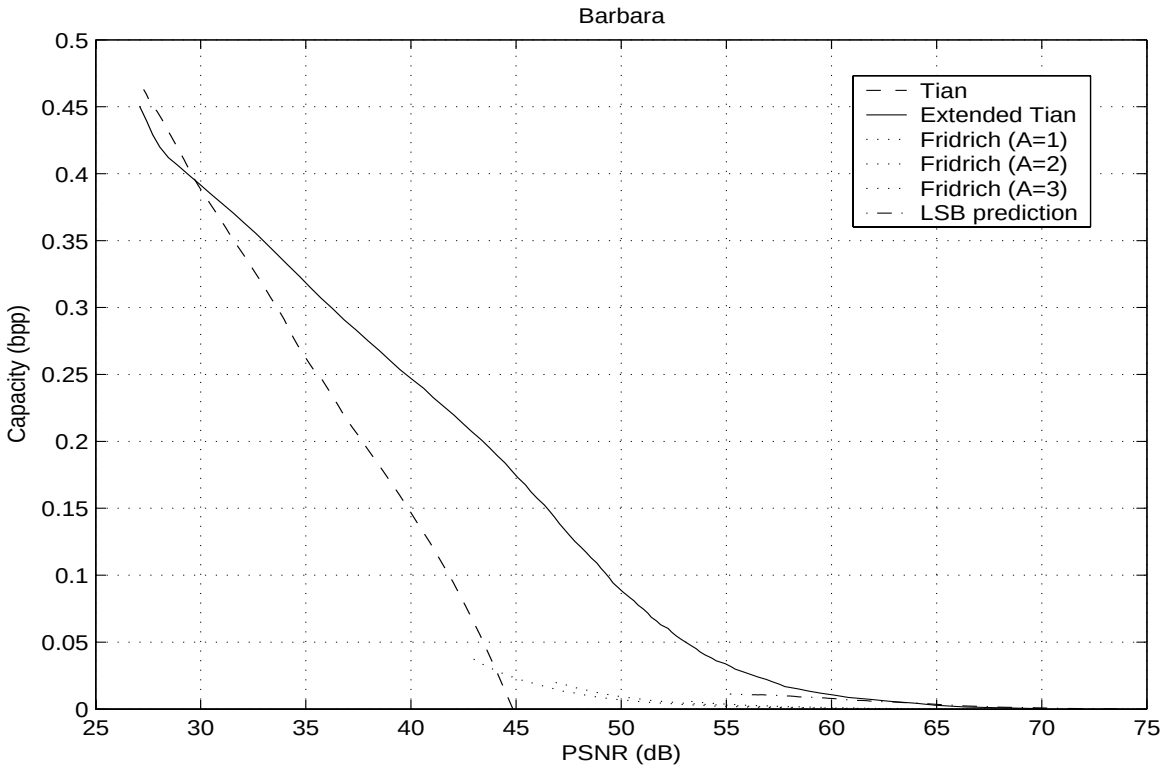

Figure 7: *Capacity versus distortion performance of the various methods for the Barbara image.*

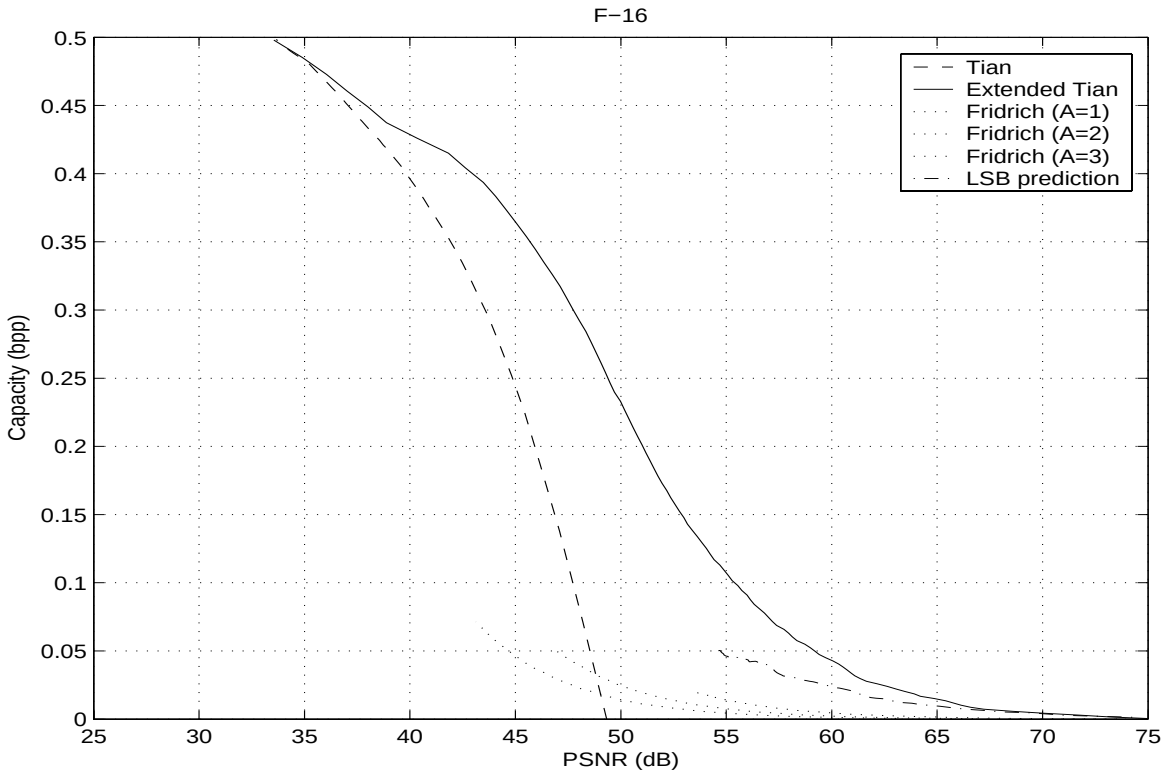

Figure 8: *Capacity versus distortion performance of the various methods for the F-16 image.*

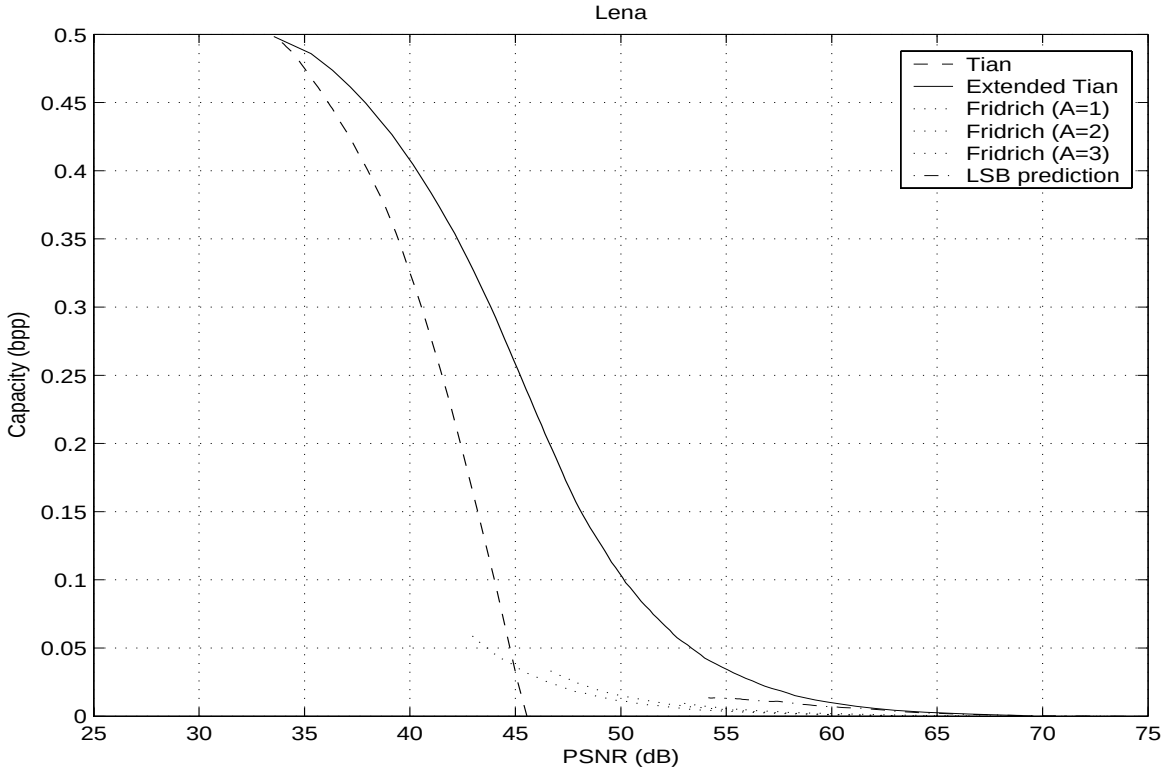

Figure 9: *Capacity versus distortion performance of the various methods for the Lena image.*

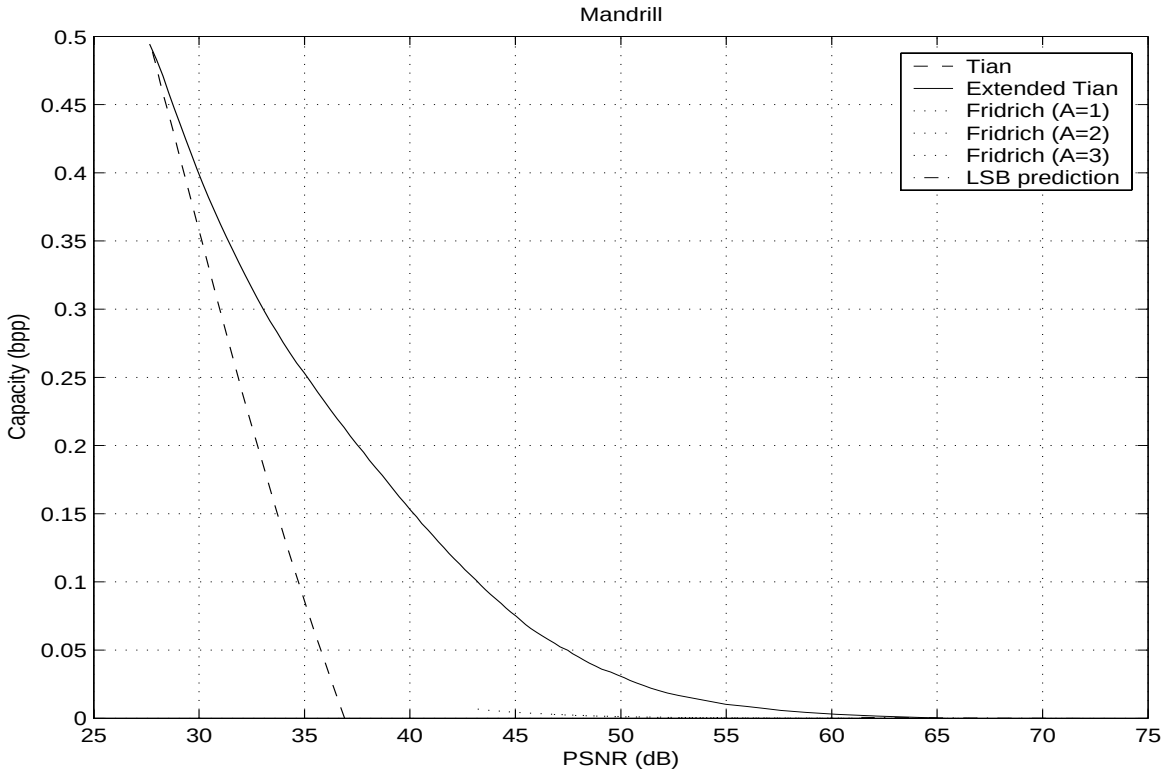

Figure 10: *Capacity versus distortion performance of the various methods for the Mandrill image.*

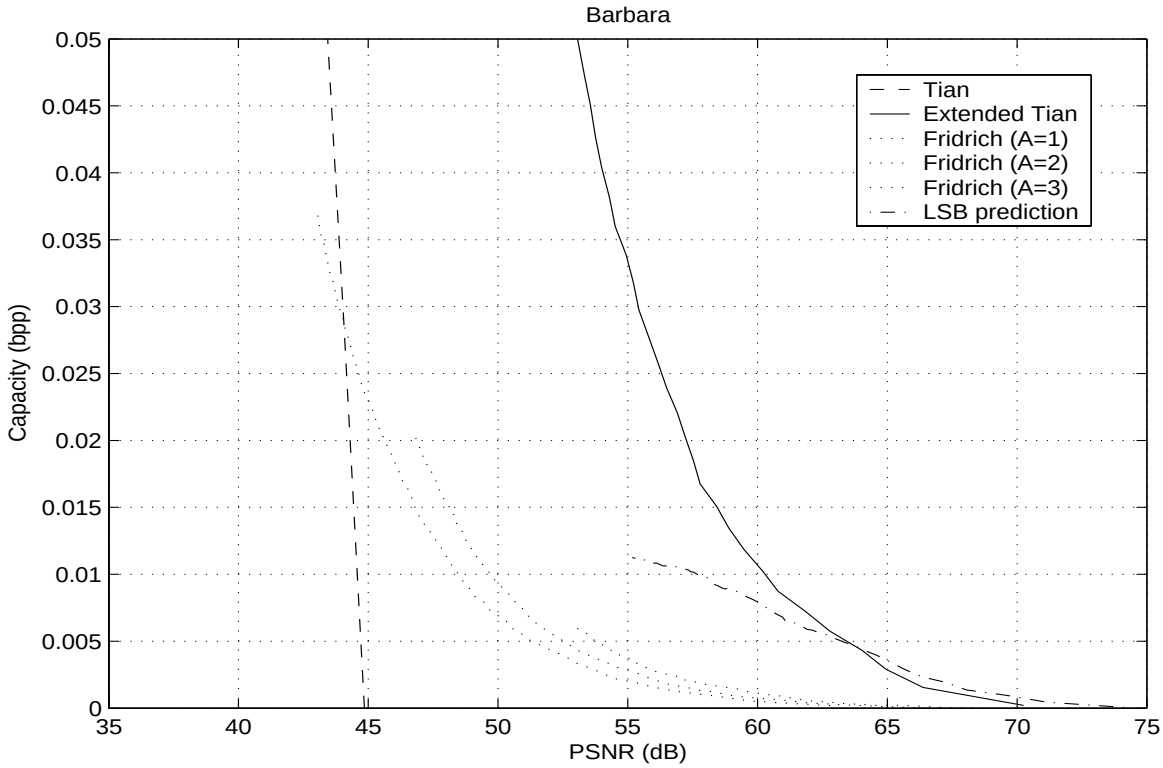

Figure 11: *Capacity versus distortion performance at low capacities of the various methods for the Barbara image.*

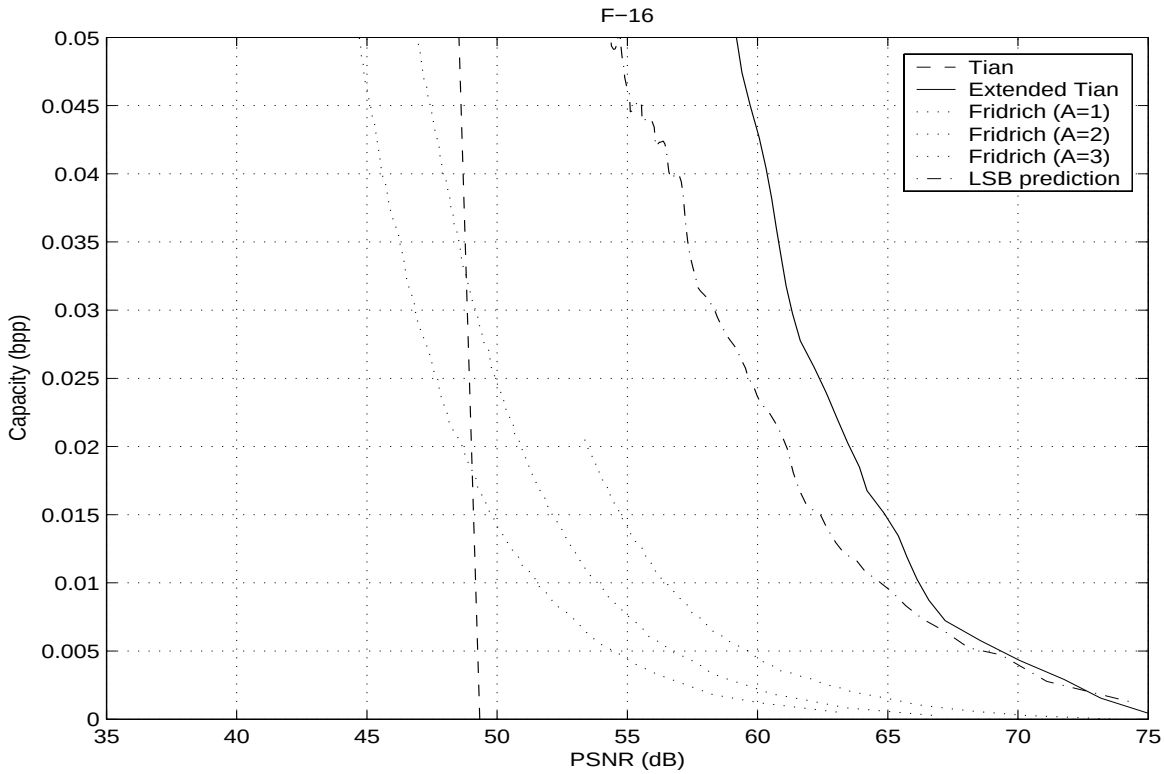

Figure 12: *Capacity versus distortion performance at low capacities of the various methods for the F-16 image.*

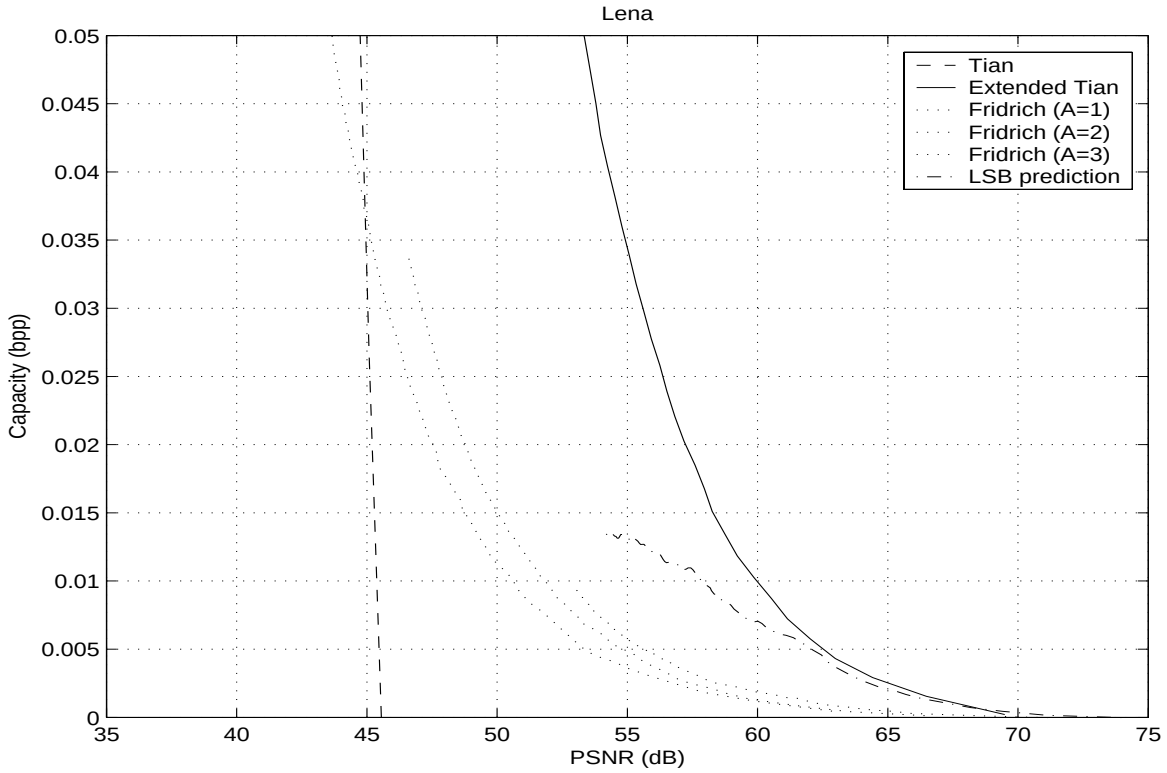

Figure 13: *Capacity versus distortion performance at low capacities of the various methods for the Lena image.*

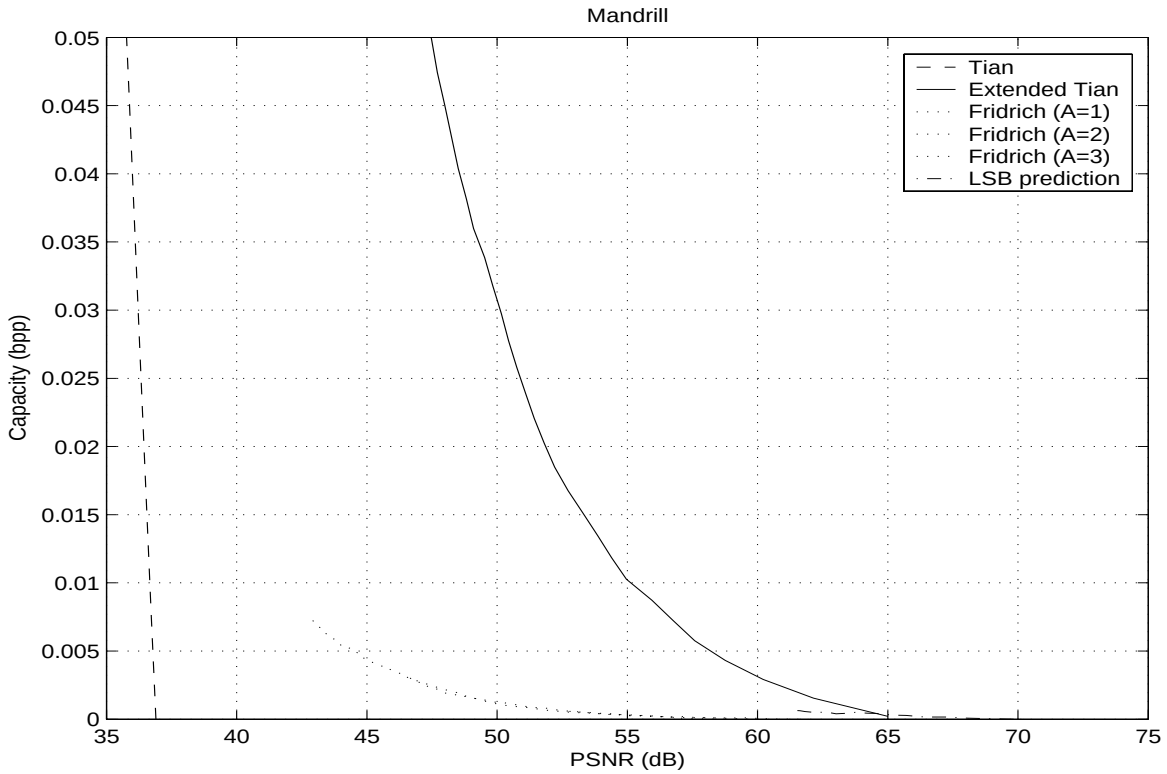

Figure 14: *Capacity versus distortion performance at low capacities of the various methods for the Mandrill image.*

	Tian	Improved Tian
Barbara	59.1%	0.2%
F-16	68.6%	0.2%
Lena	70.6%	0.2%
Mandrill	72.4%	0.2%

Table 4: *The minimum number of pixel pairs selected for expansion for which the capacity of Tian's original method and its improvement are positive for each of the four test images.*

distortion behaviour in virtually all cases. Only at very low capacities (less then 0.005 bits per pixel) does LSB prediction outperform the improved Tian method.

6.6 Capacity control

As mentioned before, capacity control is a problem for Tian's original method. Although the method can achieve large capacities, it cannot beforehand determine how many pixel pairs to select for expansion to achieve this capacity. This can only be done by making a table that, for each image, lists how many pixel pairs need to be selected to achieve certain capacities. Celik and co-workers [4] observed that their method suffers from the same problem, but we believe that progressive compression (like we used for Fridrich's method, LSB prediction and improved Tian) might solve this problem for their method.

The three other methods all use progressive compression which enables compression until the desired capacity is achieved. As a result these methods can control the capacity very well.

6.7 Embedding Time

During our experiments, we measured, for each of the methods, the amount of CPU time it takes to embed payloads of various sizes. We found that in all cases the compression module takes much more time than the other components. Furthermore, the time it takes to compress a bitstream is approximately proportional to the size of the output of the compression. Thus bitstreams that allow higher compression take less time. As a result, Tian's original method (at low capacities) and Fridrich's method are the slowest, with our method of LSB prediction a close third. All three methods have to compress bitstreams that are relatively hard to compress.

The location map that needs to be compressed in Tian's original method is harder to compress when less pairs are selected for expansion. As a consequence Tian's algorithm is faster when it embeds at a higher rate. At near maximum capacity Tian's algorithm performs as fast as our improvement.

With our improvement of Tian's algorithm the situation is very different. At low capacities, the location bitstream has very low entropy; moreover, it consists only of zeros in most cases. Thus, this bitstream allows strong compression. At high capacity, it becomes more likely that non-expandable pairs are selected, thus resulting in a small fraction of ones in the location bitstream, making this bitstream slightly harder to compress. On the whole, our improvement of Tian's algorithm is the fastest of the four methods we implemented: at low capacities it is a lot faster then the other three methods and at maximum capacity it is about as fast as Tian's original method.

7. Conclusions

After giving a review of three existing reversible data embedding methods[6] by Fridrich and co-authors [7], Tian [13], and Celik and co-workers [4], we introduced two new methods: least significant bit prediction and an improvement of Tian's method.

LSB prediction utilises a variant of Sweldens' wavelet lifting scheme [12] to predict the least significant bitplane by using the information contained in the most significant bitplanes. The capacity-distortion behaviour of the method is quite good, especially at low capacity: it easily outperforms the methods of Fridrich and Tian. The LSB prediction method sorts the predicted least significant bits according to an estimate of the prediction quality which greatly improves the performance of the method. It should be noted that such a sorting technique could just as well be used in combination with Fridrich's method. It is likely that this would also improve that embedding technique significantly.

The main advantage of Tian's original method is its high maximal capacity of up to 0.5 bit per pixel (more if the method is applied multiple times). Disadvantages are high distortions at low capacities and inability to control the capacity. We proposed an improvement of Tian's method that keeps it advantage while removing its shortcomings. Our improvement is able to keep the distortion low when embedding small messages and can automatically create just enough capacity to embed the desired payload thus keeping the distortion minimal. As an additional advantage, the improvement embeds much faster due to easy compression.

As it stands, Tian's method (and our improvement) uses the one-dimensional Haar wavelet transform to decorrelate the data and create "free space" in which to embed the payload. The advantage of this simple transform is that it is easy to describe which detail coefficients can be expanded without resulting in an overflow. It may be interesting to try and use more complicated wavelets as this could potentially decorrelate the data better and achieve higher capacities. The challenge here would be to devise a system that prevents overflow when the inverse wavelet transform is applied to the expanded detail coefficients.

On the whole, the two new methods performed substantially better in our experiments than the existing methods. Moreover our improvement of Tian's method was superior in almost all circumstances. Only at very low capacities (say < 0.005 bits per pixels) LSB prediction sometimes performed better.

[6] Reversible data embedding is a new and active research area. After this paper was finished, we became aware of two new embedding techniques: Alattar [1] introduced another extension of Tian's method and Van Leest and co-authors [14] described an entirely new method.

References

1. Alattar, A. Reversible watermark using difference expansion of triplets. In *Proceedings of the IEEE Conference on Image Processing* (Barcelona, Spain, 2003).
2. Bender, W., Butera, W., Gruhl, D., Hwang, R., Paiz, F. J., and Pogreb, S. Applications for data hiding. *IBM Systems Journal 39*, 3&4 (2000), 547–568.
3. Bender, W., Gruhl, D., Morimoto, N., and Lu, A. Techniques for data hiding. *IBM Systems Journal 35*, 3&4 (1996), 313–336.
4. Celik, M. U., Sharma, G., Tekalp, A. M., and Saber, E. Lossless generalized-LSB data embedding. submitted to IEEE Trans. Image Proc., 2003.
5. Cox, I. J., Miller, M., and Bloom, J. *Digital Watermarking*. Morgan Kaufmann Publishers, San Francisco, 2001.
6. De Vleeschouwer, C., Delaigle, J.-F., and Macq, B. Circular interpretation of bijective transformations in lossless watermarking for media asset management. *IEEE Transactions on Multimedia 5*, 1 (2003), 97–105.
7. Fridrich, J., Goljan, M., and Du, R. Lossless data embedding - New paradigm in digital watermarking. *EURASIP J. Appl. Signal Processing (Special Issue on Emerging Applications of Multimedia Data Hiding)*, 2 (2002), 185–196.
8. Kalker, A. A. C. M., and Willems, F. M. J. Capacity bounds and code constructions for reversible data-hiding. In *IS&T/SPIE's 15th Ann. Symp. Electronic Imaging* (Santa Clara, California, 2003).
9. Kalker, T., and Willems, F. Capacity bounds and constructions for reversible data-hiding. In *Proc. of the 14th International Conference on Digital Signal Processing* (July 2002), vol. 1, pp. 71–76.
10. Katzenbeisser, S., and F. A. P. Petitcolas (eds). *Information hiding techniques for steganography and digital watermarking*. Artech House Books, Norwood, 2000.
11. Swanson, M. D., Kobayashi, M., and Tewfik, A. H. Multimedia data-embedding and watermarking technologies. *Proceedings of the IEEE 86*, 6 (1998), 1064–1087.
12. Sweldens, W. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Applied and Computational Harmonic Analysis 3* (1996), 186–200.
13. Tian, J. Reversible data embedding using a difference expansion. *IEEE Transaction on Circuits and Systems for Video Technology 13*, 8 (August 2003), 890–896.
14. van Leest, A., van der Veen, M., and Bruekers, F. Reversible image watermarking. In *Proceedings of the IEEE Conference on Image Processing* (Barcelona, Spain, 2003).