2005

Unitary Embedding for Data Hiding with the SVD

Clifford Bergman
Iowa State University, cbergman@iastate.edu

Jennifer Davidson
Iowa State University

Unitary Embedding for Data Hiding with the SVD

Clifford Bergman and Jennifer Davidson

Department of Mathematics, Iowa State University, Ames Iowa, USA

ABSTRACT

Steganography is the study of data hiding for the purpose of covert communication. A secret message is inserted into a *cover file* so that the very existence of the message is not apparent. Most current steganography algorithms insert data in the spatial or transform domains; common transforms include the discrete cosine transform, the discrete Fourier transform, and discrete wavelet transform. In this paper, we present a data-hiding algorithm that exploits a decomposition representation of the data instead of a frequency-based transformation of the data. The decomposition transform used is the *singular value decomposition* (SVD). The SVD of a matrix A is a decomposition $A = USV^T$ in which S is a nonnegative diagonal matrix and U and V are orthogonal matrices. We show how to use the orthogonal matrices in the SVD as a vessel in which to embed information. Several challenges were presented in order to accomplish this, and we give effective solutions to these problems. Preliminary results show that information-hiding using the SVD can be just as effective as using transform-based techniques. Furthermore, different problems arise when using the SVD than using a transform-based technique. We have applied the SVD to image data, but the technique can be formulated for other data types such as audio and video.

Keywords: steganography, singular value decomposition, data hiding

1. INTRODUCTION

Information hiding techniques that are used today include cryptography, watermarking, and steganography. Each area has a different objective when hiding data. *Cryptography* is the study of hiding message content by encrypting or encoding the message bits in such a way that the message is unintelligible unless the key to decrypt it is known. In cryptography, it is clear that a message is being transmitted; the goal of encryption is make the unauthorized decryption of the message take unreasonable amounts of computer processing resources and time. *Watermarking* of digital data is concerned with protecting the digital data itself for ownership purposes, copy control, or other content protection purposes. In watermarking, a sequence of bits is inserted within the data. While it may be known that a watermark has been inserted for copy protection purposes, the goal of watermarking is to make removal of the inserted watermark bits impossible without additional information, such as a key. *Steganography* is a type of covert communication where a secret message is hidden in a carrier or cover message; the goal of steganography is to embed message bits so that the very existence of the message is not detectable by an observer.

Research in data hiding has blossomed during the past decade, with commercial interests, and more recently, government interests driving the field. Watermarking was initially perceived to be the answer to content protection by the music and motion picture industries to blatant and numerous violations of copyrighted material such as music and movies. The notion that embedding information directly into the data could help identify violators was very simple and attractive. It is very easy to copy and transmit music on CDs and, as computers and broadband connections became faster, movies on DVDs or movies recorded illegally using digital video cameras. Thus, companies pursued the development of watermarking techniques that could somehow identify the illegal actions that were taken on copyright material. However, to date, there have been no techniques developed that meet the expectations of watermarking as desired by music and movie industries.[1] In addition, with the advent of the Digital Millennium Copyright Act of 1998,[2] it became illegal not only to circumvent anti-piracy measures built into most commercial software and many hardware devices, but to manufacture, sell,

E-mail: {cbergman,davidson}@iastate.edu

or distribute code-cracking devices that can be used to illegally copy software. Consequently, the music and movie industries no longer rely on watermarking of copyrighted material to prove violation of the DMCA but have been relying on other approaches such as locating possible violators through their internet providers.

Because more freeware is available for embedding secret messages (see `http://stegoarchive.com`), and more bandwidth is available for sending image data efficiently, digital forensics personnel are interested in analyzing image data that may contain secret messages. This area of steganography is called steganalysis. Thus, one goal of designing a good embedding algorithm is to avoid detection by steganalytic techniques.

As a simple, but commonplace, example of steganography we describe the least significant bit embedding (LSB) algorithm. For an image A, represent A as a collection of bit planes in descending order of importance $\{A_n, A_{n-1}, \ldots, A_1, A_0\}$, where A_k is the k-th bit plane. Let $M = \{b_0, b_1, \ldots, b_{H-1}\}$ be the bit stream representation of the message. Assuming that H is less than the number of pixels of A, we visit each site in A_0, the least significant bit location of each pixel value in image A, and replace the image bit with a message bit. Since the LSB bit plane of an image closely resembles noise, manipulating the LSB location does not typically change the visual appearance of the image. Extraction of the message is performed by simply visiting the LSB locations in the image A and extracting the bits in the proper order, and reforming the message string M. For more security, the bit stream M is typically encrypted and hence its distribution resembles a random sequence of 0s and 1s.

Variations on the LSB method include using the lowest two, three, or even four bit planes to insert a message, assuming that n is sufficiently large. The main drawback to this data hiding scheme is that it is very *fragile,* that is, any distortion will change the message bits, and hence a third party can change the message bits without the receiver noticing. In addition, steganalysis techniques such as *pairs of values*[3] and the *RS method*[4] can identify not only the presence of a message embedded in this manner, but its location and length.

We focus on image data in this paper, although our method can be modified and applied to video and audio data. There are several reasons why images are used for steganography. First, because of the high degree of redundancy in image data, it is possible to embed a great deal of hidden information without visibly affecting the cover image. Second, innocuous-looking images are commonplace throughout the internet and arouse little suspicion. By contrast, under current bandwidth availability, video files posted on web sites take too long to transfer effectively. Also, audio and video data are prone to be examined for copyright infringement. Third, the sheer volume of image data available online makes it difficult to identify suspicious content. Thus, image data is commonly used for data hiding.

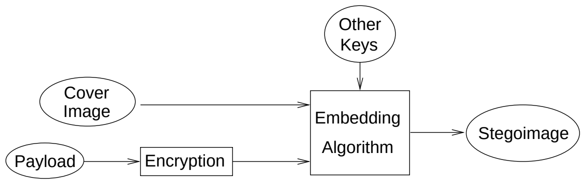
We organize the rest of the paper follows: Section 2 describes some fundamental concepts and terminology of data hiding; Section 3 reviews the singular value decomposition which is the main tool we use for our embedding algorithm; Section 4 describes the basic embedding algorithm; Section 5 describes enhancements to the algorithms and the extraction algorithm; Section 6 discusses experimental results applied to six image data; and Section 7 presents the conclusions and future direction of the research.

2. BASIC PROPERTIES OF DATA HIDING

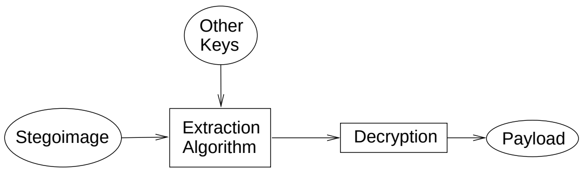
In this section, we define some concepts basic to data hiding. These properties apply to watermarking as well as steganography.

A simple scenario for image steganography can be described as follows. Alice embeds a *payload message* within a *cover image,* producing a *stegoimage,* and sends the stegoimage to Bob under the noses of an adversary who recognizes only the innocent cover. Bob *extracts* the payload from the stegoimage. Alice would like to ensure that her message gets to Bob safely, without the adversary either suspecting or destroying the payload. Figure 1 displays the overall procedure in diagram form.

There are several properties important in the creation and evaluation of an effective stego-algorithm. These include the following.



(a) embedding



(b) extraction

Figure 1. Basic steps in stegonagraphic embedding

Capacity measures the amount of payload that can be embedded in a fixed size cover file. It is measured in bits (of payload) per byte (of cover). For example, in the LSB embedding scheme, the sender chooses as a cover image a grayscale, bit-mapped image, and replaces the least significant bit of each pixel with one bit of payload. Assuming each pixel occupies 8 bits, this scheme has a capacity of 1 bit/byte.

Perceptibility describes the ability of a third party (not the intended recipient) to visually detect the presence of hidden information in the stegoimage (or audibly, in the case of audio data). Note that we do not require the third party to extract the information, just preceive its existence. We say that the steganography embedding algorithm is *imperceptible* when used on a particular image if an innocent third party, interested in the content of the cover image, is unaware of the existence of the payload. Essentially this requires that the embedding process not degrade the visual quality of the cover image.

Detectability describes the probability that a determined adversary, who suspects steganography, will be able to determine the existence of a payload, thus compromising the message's security. In other words, a stego algorithm provides low security if the payload is detectable in the stegoimage with a high probability. Undetectability is obviously a much more difficult requirement to meet than is imperceptibility, but as the use of steganography increases, so will the use of steganalysis. The concept of a *safe bit-rate* is related to detectability of a stegoalgorithm. The safe bit-rate (SBR) is the maximum capacity of a stego algorithm, when applied to a particular image, that is not detectable by steganalysis. The SBR is therefore dependent not only on the algorithm used to embed the data, but the data itself as well as the steganalysis techniques available to detect it. It is estimated by Fridrich and Goljan[4] for example, that the safe bit-rate of LSB embedding is 0.005 bits/pixel. It appears that the SBR for most stegoalgorithms will be much less than the current capacities available by many embedding algorithms.

Robustness characterizes the ability of the payload to survive the embedding and extraction process, even in the face of manipulations of the stegoimage such as filtering, cropping, rotating and compression.

Speed reflects the computational effort required to embed and extract the hidden data.

It is well-understood that there is always a tradeoff between capacity and visual imperceptibility, and capacity and detectability. As described above, the LSB method has a very high capacity of 1 bit/pixel, but as is shown by Westfeld and Pfitzmann,[3] is extremely vulnerable to any kind of filtering as well as detectable by statistical attacks. By contrast, there are no reliable methods for detecting the existence of payloads hidden using other techniques. Fridrich, Goljan and Du[5] have estimated that in order to attain a high degree of undetectability, stego systems based on least significant bit methods must have a capacity bounded above by 0.005 bits/byte. For this reason, we believe it is important to develop steganographic techniques that do not rely on least significant bit manipulations. Our algorithm attempts to attain that goal.

3. REVIEW OF THE SINGULAR VALUE DECOMPOSITION

The singular value decomposition (SVD) of a matrix with real or complex entries is one of the fundamental tools of numerical linear algebra. It has applications to regression analysis, data compression and numerical linear algebra among others. Because of its importance, there are a wealth of numerical tools available for computing the SVD. In this section we summarize the definitions and properties of the SVD that we shall need. A full development can be found in most linear algebra texts, for example Golub and Van Loan.[6]

Let A be an $n \times n$ matrix with real entries. Note that while we restrict our discussion to real-valued, square matrices, the SVD applies more generally to complex-valued rectangular matrices. The *singular value decomposition of* A is a representation

$$A = USV^T \tag{1}$$

in which U and V are $n \times n$ orthogonal matrices, and S is a diagonal matrix with nonnegative entries. Recall that a matrix U is *orthogonal* if $U^TU = I$. Put another way, U is orthogonal if its columns are pairwise orthogonal unit vectors. As usual, U^T denotes the transpose of the matrix U.

As a rule, it is the largest singular values that exert the most influence on the matrix A. In particular, when A contains the greyscale values for an image, perturbations of the smaller singular values and their corresponding singular vectors have no perceptible effect on the image. It is this mathematical principle that our embedding technique is designed to exploit.

The diagonal entries of S in equation (1) are of the form $\sqrt{\lambda}$, where λ is an eigenvalue of the symmetric matrix AA^T. The corresponding eigenvectors form the columns of U. The diagonal entries of S are called the *singular values of A,* while the columns of U and V are the left and right *singular vectors.* Note that the eigenvalues of AA^T are real and nonnegative. Consequently the singular values of A are also real and nonnegative.

The question of uniqueness in the decomposition (1) is of concern to us here. To that end we make the following definition.

Definition 3.1.

1. A vector $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ is *lexicographically positive* if its first nonzero component is positive.
2. A singular value decomposition $A = USV^T$ is *normal* if the columns of U are lexicographically positive and the diagonal entries of S are in non-increasing order.

Theorem 3.2. *A matrix has a unique normal singular value decomposition if its singular values are pairwise distinct and nonzero.*

Proof. Since the singular values of A are pairwise distinct, the eigenvalues of the matrix $B = AA^T$ are pairwise distinct as well. B is a symmetric matrix, hence the spectral theorem asserts that B has a basis of eigenvectors. Combining these two facts, each eigenspace of B is one-dimensional. Thus, if we require the eigenvectors to have unit length, there are exactly two choices for each eigenvector, $\mathbf{u}$ and $-\mathbf{u}$, only one of which is lexicographically positive. Since the eigenvectors of B form the columns of U, we have only one choice for each column of U.

The requirement that the diagonal entries of S be decreasing imposes uniqueness on S and also on the order of the columns of U. Hence U is unique. Since the singular values are nonzero, S is invertible. Finally, we obtain V uniquely as A^TUS^{-1}. $\square$

In our applications, the matrix A will have integer entries in the range $\{0, 1, \ldots, 255\}$ and dimension between 8 and 16. It seems quite difficult to determine the probability that such a matrix will fail to satisfy the hypothesis of Theorem 3.2, however our experience suggests that this probability is exceedingly small. Thus we will proceed under the assumption that all matrices have distinct, nonzero singular values and all singular value decompositions are normal. Of course in practice, this means our computations have to be sufficiently accurate that we can distinguish between distinct singular values.

3.1. Related work

We are not aware of other uses of the singular value decomposition in steganography. Liu and Tan[7] proposes a watermarking technique that utilizes the SVD.

1. Compute the normal singular value decomposition, USV^T, of A.
2. Transform U into U':
   (a) Set certain components $u'_{ij} = p_k \cdot |u_{ij}|$
   (b) Choose remaining components to ensure that U' is still orthogonal.
3. Compute $A' = U'SV^T$.
4. Clip and round the entries in A' to integers in the range $0 \ldots 255$. The resulting matrix, $\tilde{A}$ will be a block of the stegoimage.

Figure 2. The basic embedding algorithm

4. THE BASIC EMBEDDING ALGORITHM

In this section we describe the procedure for embedding the payload into the cover image. There are problems with this algorithm, most notably in the error rate incurred during recovery. In Section 5 we discuss enhancements to the basic procedure that reduce the error rate to an acceptable level.

4.1. Description of the algorithm

We denote by $p_1 p_2 p_3 \ldots$ the payload that we wish to embed. Each payload bit p_i is assumed to have a value in $\{+1, -1\}$. (A conventional $\{0, 1\}$-bit b can be converted to a $\{+1, -1\}$-bit using either of the transformations $b \mapsto (-1)^b$ or $b \mapsto 2b - 1$.) The cover image is assumed to be a rectangular array of pixel locations with values in the range $0 \ldots 255$. All of our experiments were done with greyscale images, however, our approach could be applied to color images either by utilizing the intensity byte in HSI images, or possibly by embedding bits into each of the three color components of a cover image in RGB format.

The cover image is divided into a series of $n \times n$ blocks in some standard order. If the number of rows or columns in the cover image is not a multiple of n, extra rows and columns can be ignored. Several bits of payload will be embedded into each block.

Let A be a typical $n \times n$ block. Payload bits are embedded into A by a simple four-step process, given in Figure 2. Step 2 requires further explanation. Recall that the visual quality of A is primarily determined by its largest singular values and singular vectors. By assumption, those are the left-most values of S and the left-most columns of U and V. Part of our strategy is to leave those columns untouched in order to achieve imperceptibility in our method. The precise number of columns that we protect in this way is a parameter that can be adjusted. We discuss the tradeoffs in this parameter below. We denote by m the number of columns of U that will be left unchanged.

For simplicity in this discussion, let us take $n = 8$ and $m = 2$. In other words, we will be manipulating the right-most 6 columns of each 8×8 block. At the end of this section we shall give formulas for general values of n and m. With $n = 8$, $m = 2$, we can embed 15 bits in each block of the cover image.

A schematic illustration of the matrix U is given in Figure 3. Note that it is divided into three regions. The shaded region consists of those entries that are to be left unchanged. This consists of the two columns we wish to protect as well as the first row. The first row is left unchanged because the definition of normal SVD requires the top row of U to be all nonnegative. The upper triangular region contains the entries that will hold the embedded bits. The bits are embedded according to the formula

$$u'_{ij} = p_k \cdot |u_{ij}|, \qquad i = 3 \ldots 8;\ j = 2, \ldots, 9 - i.$$

In this equation, $k = \frac{1}{2}(i-3)(14-i) + (j-1)$ simply counts from 1 to 15. The effect of this transformation is that the triangular entries in U' differ only from the corresponding entries in U in their sign. The sign of the entry in U' **is** the embedded bit.

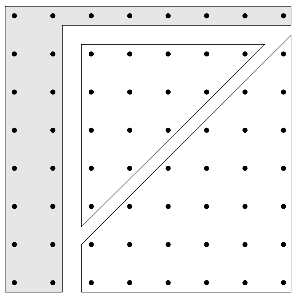


Figure 3. The matrix U

We now use the remaining entries in the matrix (those in the lower trapezoid) to ensure that U' will be orthogonal. Let U'_i denote the ith column of the matrix U'. Orthogonality requires that $U'_i \cdot U'_j = \delta_{ij}$. We proceed as follows. First choose u'_{73} and u'_{83} to satisfy the pair of equations

$$\begin{aligned} U'_1 \cdot U'_3 &= 0 \\ U'_2 \cdot U'_3 &= 0. \end{aligned}$$

This amounts to solving a homogeneous 2×2 system of linear equations. Then choose $u'_{64}, u'_{74}, u'_{84}$ to satisfy the system

$$\begin{aligned} U'_1 \cdot U'_4 &= 0 \\ U'_2 \cdot U'_4 &= 0 \\ U'_3 \cdot U'_4 &= 0. \end{aligned} \tag{2}$$

Continue this process to determine the lower entries in $U'_5, \ldots, U'_8$. Note that determining each column requires solving a simple system of linear equations. (In fact, these equations could be solved once symbolically, and then the solution evaluated for each block of the image.) Finally, we finish the process by dividing each column of U' by its Euclidean norm. That is, $U'_i \leftarrow U'_i/|U'_i|$, $i = 3, \ldots, 8$. The resulting matrix U' is orthogonal and its columns will be lexicographically positive.

Step 3 of the basic algorithm simply multiplies together the new matrix U' with the original matrices S and V^T. This produces a matrix A' from which the payload can be reliably recovered. From Theorem 3.2, A' has a unique normal SVD. Thus U' can be computed from A' and the payload bits extracted from the triangular region in U'.

Unfortunately, the entries in A' are not generally integers in the range $0 \ldots 255$, so additional processing is necessary before A' can be used as a block of the stegoimage. We have chosen the simplest processing scheme: each entry is rounded and clipped to the range 0. . . 255. That is, if an entry x satisfies $0 \leq x \leq 255$, it is rounded to the nearest integer. If $x < 0$, then x is replaced by 0, and if $x > 255$, it is replaced by 255.

This strategy does indeed result in a valid image and furthermore, the perturbation caused by the embedding process is imperceptible. However, we now a have a different problem: the extraction process gets many of the payload bits wrong. The signs of the entries in the singular vectors are not particularly robust to the process of rounding and clipping. It is this defect that the additional measures described in Section 5 are intended to remedy.

Figure 5 shows an example of the basic embedding technique. On the left is a 512×512 pixel cover image before embedding. On the right is the result of embedding 61,440 bits into the image. Any distortion caused by the embedding is completely invisible. 77% of the embedded bits were retrieved correctly.

1. Compute the SVD, $\tilde{U}\tilde{S}\tilde{V}^T$ of $\tilde{A}$.
2. Extract payload bits from the signs of the entries in the triangular portion of $\tilde{U}$: $p_k = \tilde{u}_{ij}/|\tilde{u}_{ij}|$.

Figure 4. The extraction algorithm

4.2. Further comments on the parameters

The details of the above discussion were based on the choice of $n = 8$ and $m = 2$. One can make other choices for these parameters but the basic strategy remains the same. The top row and the first m columns of U are left unchanged. Payload bits are embedded in an upper triangle consisting of $1 + 2 + \cdots + (n - m - 1) = \frac{1}{2}(n-m-1)(n-m)$ matrix entries. Then the values in the remaining lower trapezoid are determined to ensure that U' is orthogonal.

The choices of n and m affect the capacity, the imperceptibility and the robustness of this process. The capacity of the basic algorithm is easily seen to be

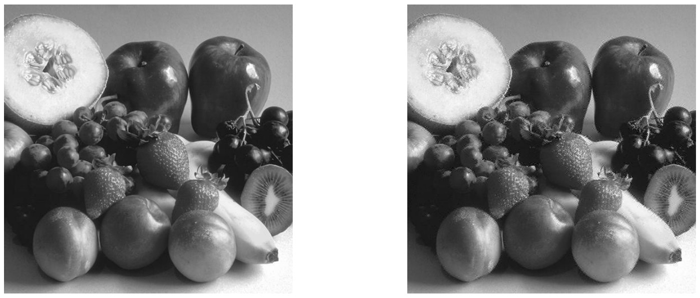
$$\frac{(n-m-1)(n-m)}{2n^2} \text{ bits/pixel.} \tag{3}$$

The imperceptibility is directly related to the value of m. A higher value of m preserves more of the singular values in the original image block, hence the stegoimage will very closely resemble the original cover image. Curiously, the error rate is also directly dependent on the value of m. In other words, increasing m results in an increased number of errors. At this time we do not have a satisfactory understanding of this phenomenon, but the probable explanation is that as m is increased, a larger proportion of the bits are embedded in the right-most columns of U, and these are the least robust to perturbation. A further discussion of these relationships together with our experimental results is given in Section 6.

Let us remark that it is possible for this algorithm to fail. One possibility is that the hypotheses of Theorem 3.2 will not hold. Another is that one of the systems of equations such as (2) may prove to be singular. Should this occur, we can take advantage of the high degree of redundancy in the image and simply apply a small random perturbation to the image block A. We have found that adding a small amount of white noise to each component of A fixes the problem.

4.3. Encryption of the payload

We assume the payload is encrypted before embedding begins. There are several reasons for this. The most obvious is that encryption provides additional protection for the content of the communication. However,


(a) Cover image before embedding

(b) Stegoimage

Figure 5. Effect of embedding on cover image

encryption is also necessary for undetectability. If unencrypted, the payload will generally have statistical characteristics that could be exploited by a suspicious adversary who is looking for steganography. For example, if the payload were ordinary ASCII characters, then every eighth bit would be a '0'. Since this is not a characteristic one would expect in a typical image, the adversary could easily conclude that steganography was in use.

Broadly speaking, ciphers fall into two categories: block ciphers and stream ciphers. In a block cipher, a block of plaintext (typically 64 or 128 bits) is encrypted all at once, producing a block of ciphertext of the same length. Familiar examples are DES and AES. A stream cipher, on the other hand, operates one bit at a time. Usually a long pseudo-random bit stream, called a *running key,* is generated. This bit stream is added, bit by bit, to the message to be encrypted. The receiver decrypts by subtracting the same pseudo-random bit stream from the ciphertext. RC4 and SEAL (see Schneier[8]) are examples of stream ciphers. Also, any block cipher operated in either output feedback mode or counter mode can be used as a stream cipher.

Although block ciphers are considered to be more secure than stream ciphers, a block cipher would be a poor choice for this application. In a block cipher, even a one-bit error results in the corruption of the entire block at decryption time. Since our scheme relies on error correcting codes to repair individual errors (see Section 5.3), it is necessary to utilize a stream cipher for encryption.

5. ENHANCEMENTS TO THE BASIC ALGORITHM

The basic embedding algorithm described in Section 4 is promising in most respects: it has high capacity and good imperceptibility properties. Unfortunately, it suffers from an unacceptably high rate of errors upon retrieval of the payload. In this section we attempt to analyze the causes of those errors and propose remedies.

5.1. Spacing the singular values

If an eigenvalue of a square matrix has multiplicity 1, the corresponding eigenvector is uniquely determined, except for its magnitude. This is the principle that was employed in Theorem 3.2. For eigenvalues of higher multiplicity however, eigenvectors are much less constrained. They can range anywhere inside a k-dimensional subspace, where k is the multiplicity of the eigenvalue.

It turns out that when two eigenvalues are distinct but close together, they behave approximately like a single eigenvalue of multiplicity 2. As a result, small perturbations in the matrix can cause the corresponding eigenvectors to vary wildly inside a 2-dimensional subspace. Or, as Golub and Van Loan put it,[6] "eigenvectors associated with nearby eigenvalues are 'wobbly'".

These considerations suggest that we will experience less volatility in the singular vectors, and consequently obtain fewer errors, if the singular values are well-spaced. Of course the largest few singular values determine the perceptual quality of the image, thus we wish to leave those values unchanged.

For this reason we add the following step to the basic embedding algorithm (Figure 2), between steps 1 and 2.

1a. Replace the last $n-k$ diagonal entries of S, $\sigma_{k+1}, \sigma_{k+2}, \ldots, \sigma_n$, with the values $\sigma_k - h, \sigma_k - 2h, \ldots, \sigma_k - (n-k)h$, where $h = (\sigma_k - \sigma_n)/(n-k)$.

This perturbation seems to have very little effect on the visual quality of the image, at least when n is relatively small.

After step 4 of the embedding algorithm, the singular values of $\tilde{A}$ will no longer be quite uniform. However they will probably be more uniform than in a typical image. It is possible that this characteristic could be utilized by a steganalyst scanning for hidden data.

5.2. Iteration of the basic embedding algorithm

Our second strategy for reducing errors is to simply iterate the basic embedding algorithm several times on each block, embedding the same bits each time. That is, we begin with a block A and payload bits $\mathbf{p} = p_1, \ldots, p_k$. We apply steps 1–4 (including step 1a) to A and $\mathbf{p}$ and obtain the output block $\tilde{A}$. We then apply the same steps to $\tilde{A}$ and $\mathbf{p}$ and obtain a new output block $\tilde{\tilde{A}}$, etc.

Although we have no proof that this method will "converge", empirical evidence indicates that iterating 5–10 times (especially in conjunction with our other techniques) results in a considerable reduction in the error-rate. Notice that the extraction algorithm remains unchanged. Thus, while embedding is obviously slowed by a factor equal to the number of iterations, extraction is unaffected.

Instead of iterating on each block a fixed number of times, one can use an adaptive approach. For each block, embed the payload using the basic algorithm, then immediately extract the bits and count the number of errors. If the errors exceed a set threshold, repeat the process until the threshold is met. One could, of course, set the threshold to 0, but this may result in an unacceptably long (if not infinite) running time. However, as we discuss in the next subsection, in conjunction with an error-correcting code, this strategy can be quite effective.

5.3. Use of an error-correcting code

The tools of coding theory are well-developed and well-suited to the task of preserving data-integrity under noisy conditions. This model fits our problem very well. Through the use of an error-correcting code we can reduce the received error-rate quite effectively.

For example, if we find that the basic embeddding algorithm results in an error rate of 0.10, we could employ a coding scheme that is capable of correcting 10% of its bits. In this way we should be able to all but eliminate errors upon extraction.

While it is not necessary to do so, there is some advantage in choosing a code with a word size that matches the number of bits that can be embedded in a single block of the cover. The reason is that in the adaptive iteration method discuseed in Section 5.2, the threshold can be chosen equal to the number of errors per word corrected by the code, which should result in errorless retrieval. This suggests choosing codes with a word length equal to one of the triangular numbers 15, 21, 28, etc. Good choices might be the BCH codes $(15, 7, 5)$, $(15, 5, 7)$ and $(21, 9, 8)$ all of which have reasonably good error-correcting capabilities and capacities. There are numerous references on coding theory, such as Huffman and Pless.[9]

For example, let us assume we have chosen to employ the basic embedding algorithm with a block size of 8 and 2 columns protected. This choice of paramters allows us to embed 15 bits in each block of the cover image. We decide to use the $(15, 7, 5)$ code, which holds 7 bits of payload in each word and is capable of correcting 2 errors in each 15 bit word.

We proceed as follows. The payload is first coded according to the BCH code, and then encrypted using a stream cipher. The resulting bit string is broken into words of length 15. Each word is then embedded into an 8×8 block of the cover image using the basic embedding algorithm, repeating the embedding until the number of errors in that block is at most 2. In this way, we can be confident that when the recipient extracts the string, decrypts and decodes it, the decoding process should correct any errors that remain.

Note that employing an error-correcting code will decrease the capacity of the embedding algorithm by a factor equal to the rate of the code. In our example above with $n = 8$ and $m = 2$, the original capacity was 0.23 and the code has a rate of 7/15. Thus we are able to embed .107 bits/pixel after error-correction.

It is important that a stream cipher be utilized, and that it be applied *after* the payload is error-coded. This is because the error correcting code leaves a footprint (in the form of a well-defined redundancy among bits) that would be easy for a steganalyst to detect. This footprint is concealed by the cipher.

A stream cipher will not interfere with the error correction for the following reason. Assume we are using the $(15, 7, 5)$ code. We begin with 7 bits of payload, $\mathbf{p} = p_1 p_2 \ldots p_7$. These seven bits are processed by the ECC, producing a 15 bit code word $\mathbf{c} = c_1 \ldots c_{15}$. Next, 15 bits of running key, $\mathbf{r} = r_1, r_2, \ldots, r_{15}$, are obtained from

	Columns Protected			
block size	1	2	3	4
6	0.28	0.17	0.08	0.03
8	0.33	0.23	0.16	0.09
10	0.36	0.28	0.21	0.15
12	0.38	0.31	0.25	0.19
14	0.40	0.34	0.28	0.23

Figure 6. Capacity (bits/pixel) as a function of block size and number of columns protected

the stream cipher. We multiply the two bit strings (don't forget that our bits are represented as ± 1), obtaining a 15-bit cipher-word $\mathbf{s} = \mathbf{c} \cdot \mathbf{r}$. It is $\mathbf{s}$ that is embedded into the cover image block.

Suppose that on extraction, we obtain a string $\hat{\mathbf{s}}$ in which two bits of $\mathbf{s}$ are corrupted. That is, $\hat{s}_i = -s_i$ and $\hat{s}_j = -s_j$, for two indices i and j. Then $\hat{\mathbf{s}}$ is decrypted to obtain $\hat{\mathbf{c}} = \hat{\mathbf{s}} \cdot \mathbf{r}$. Note that $\hat{c}_i = \hat{s}_i \cdot r_i = -s_i \cdot r_i = -c_i$ and similarly, $\hat{c}_j = -c_j$. All other bits of $\mathbf{c}$ are retrieved intact. Finally, since $\hat{\mathbf{c}}$ contains only two errors, the code is able to correctly compute the original payload $\mathbf{p}$.

6. EXPERIMENTAL RESULTS

As we have explained, our scheme involves several parameters that must be fine-tuned in order to optimize the combination of imperceptability, data capacity and robustness. In this section we describe some of the experiments we have done in this regard and make some recommendations.

In this discussion, we have used several abbreviations. 'Blocksize' (bs) denotes the number of rows and columns of the matrix used by the basic embedding algorithm. (This was denoted by the letter n in the development in Section 4.) The 'number of columns protected' (cp) is the number of columns in each block left unchanged by the embedding (m in Section 4). 'Singular values protected' (svp) denotes the number of singular values left unchanged by the uniformization technique in Section 5.1 (denoted k in that section).

The relationship of the parameters to capacity is the easiest to describe. A formula for data capacity is given in equation (3), where n denotes the block size and m the number of columns protected. Figure 6 shows some typical combinations of the parameters.

Asymptotially, the capacity is $\frac{1}{2}\left(\frac{n-m}{n}\right)^2$. Note that the quantity in parenthesis is the proportion of columns left unprotected (and consequently available to hold data). Unfortunately, our experiments indicate that the perceptability of the embedding process is also directly related to that same ratio.

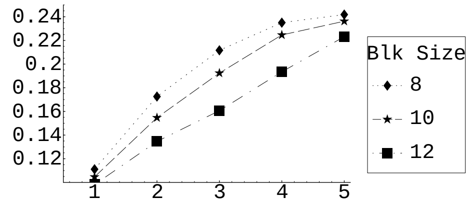
A second parameter that affects visual image quality is the number of singular values protected. The situation is similar to that of columns protected, but not as pronounced. (Increasing the number of singular values protected does not affect capacity. It does, however, cause an increase in the error rate, see Figure 8.) The number of singular values protected must be at least as large as the number of columns protected.

Based on our purely visual experiements we have settled on the following combinations as providing minimal acceptable levels of imperceptability.

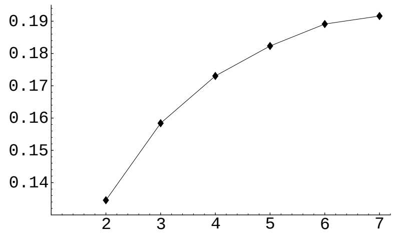
block size	8	8	10	10	12
cols. protected	2	3	4	5	5
sing. vals. protected	5	3	7	5	9
capacity	0.23	0.16	0.15	0.10	0.15

We did not consider larger block sizes for several reasons. Most importantly, the singular value decomposition becomes more expensive as the block size increases, and round-off error becomes more of a problem. Furthermore, our experience indicates that image quality degrades rather quickly with an increase in block size.

We now turn to the error rate. Our estimate of the error rate is computed by simply embedding a random bit string, extracting it and counting the number of incorrect bits. The error rate is defined to be the quotient



(a) columns protected



(b) singular values protected (bs=8, cp=2)

Figure 7. Error rate as a function of number of columns and number of singular values protected

of the number of bit errors by the total number of bits, and can be viewed as the probability that any given bit will be recovered incorrectly. Our experiments were done using six sample images that display a variety of typical image characteristics.

As Figure 7 indicates, the error rate increases as both the number of columns protected and number of singular values protected increases. Thus it is necessary to strike a compromise between imperceptability (which prefers more protected columns and singular values) and error rate (which prefers fewer).

It seems that an error rate of approximately 0.10 is unavoidable with the basic embedding strategy. For this reason we have developed several techniques for reducing the errors. The first is iterating the data-embedding step as explained in section 5.2. Figure 8 shows the effect of iteration on the error rate. Clearly, iteration can greatly decrease the number of errors, although it will not eliminate them entirely. Note that the time required to run the embedding algorithm is linear in the number of iterations performed.

The second strategy is the use of an error-correcting code as explained in Section 5.3. Figure 9 shows the result of first embedding and then extracting a valuable payload from an innocent cover image. In our experiments using a blocksize of 8 with 2 columns and 5 singular values protected, and using the BCH(15,7,5) code, we acheive an error-rate of approximately 0.0006. We believe this will be satisfactory for all practical purposes.

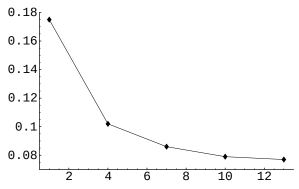


Figure 8. Error rate as a function of number of iterations

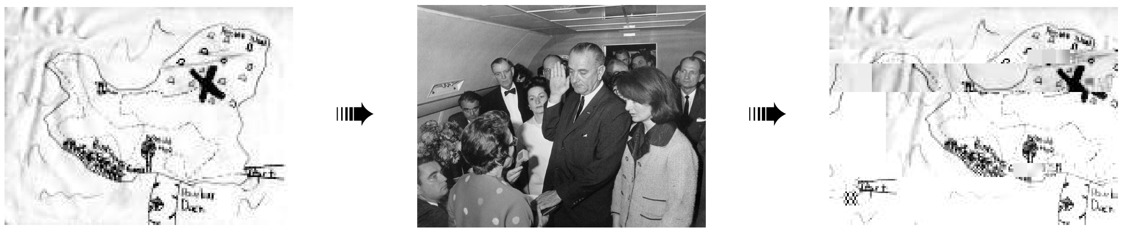


Figure 9. Example of embedding and extraction of payload

7. CONCLUSIONS

We have described a new method of hiding data in a graphic image file. By choosing parameters carefully, this embedding is impreceptible to the casual observer and, especially if used with error correction, allows extraction of the message with few errors. Even with error correction, the capacity of the method is approximately 0.1 bits/pixel, which compares very favorably to the safe bit rate of LSB methods as estimated by Fridrich and Goljan.

There are several avenues of future research involving this method. We have not addressed the crucial question of undetectability. We have argued that the use of a stream cipher will conceal the use of the error-correcting code. However, in many ways, this method does resemble an LSB-type technique in that individual message bits are inserted into the cover file. Is it possible to adapt either the PoV[3] or RS[4] attacks to detect messages embedded via the SVD?

There are also questions of robustness to consider. As it stands, this method is not at all robust against common image manipulations such as cropping, rotation or compression. Is it possible to modify our embedding technique so that the message survives these manipulations?

ACKNOWLEDGMENTS

This work was partially funded by the National Institute of Justice, through the Midwest Forensics Resource Center at Ames Laboratory under interagency agreement number 2002–LP–R–083. The Ames Laboratory is operated for the US Department of Energy by Iowa State University, under contract No. W–7405–Eng–82.

The treasure map in Figure 9 was provided by Ella (Peg-Leg) Bartlett. All other images used in this study were obtained from the Watermarking Evaluation Testbed (`http://www.datahiding.org`), courtesy of Dr. Edward Delp. Daniel Wengerhoff did some preliminary computations using our method. His results were influential in its final formulation.

REFERENCES

1. D. Schonberg and D. Kirovski, “Fingerprinting and forensic analysis of multimedia,” in *Proc. 12th Annual ACM Int’l Conf. on Multimedia*, pp. 788–795, Assoc. Comput. Machinery, 2004.
2. “Digital millennium copyright act.” `http://thomas.loc.gov.cgi-bin/query/z?c105:H.R.2281.ENR:`.
3. A. Westfeld and A. Pfitzmann, “Attacks on steganographic systems,” in *3rd Int’l Workshop on Information Hiding*, *Lecture Notes in Computer Science* **1768**, pp. 61–75, Springer-Verlag, (Berlin), 2000.
4. J. Fridrich and M. Goljan, “Practical steganalysis—state of the art,” in *Proceedings SPIE Photonics West*, **4675**, pp. 1–13, (San Jose, CA), January 2002.
5. J. Fridrich, M. Goljan, and R. Du, “Reliable detection of LSB steganography in grayscale and color images,” in *Proc. ACM Special Session on Multimedia Security and Watermarking*, pp. 27–30, Assoc. Comput. Machinery, (Ottawa, Canada), 2001.
6. G. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1983.
7. R. Liu and T. Tan, “An SVD-based watermarking scheme for protecting rightful ownership,” *IEEE Trans. Multimedia* **4**(1), pp. 121–128, 2002.
8. B. Schneier, *Applied Cryptography*, John Wiley and Sons, New York, second ed., 1996.
9. W. Huffman and V. Pless, *Fundamentals of Error-Correcting Codes*, Cambridge University Press, Cambridge, U.K., 2003.