ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
Katedra matematiky

MINIMALIZACE VLIVU VLOŽENÉ INFORMACE VE STEGANOGRAFII POMOCÍ ŘÍDKÝCH KÓDŮ

MINIMIZING EMBEDDING IMPACT IN STEGANOGRAPHY USING LOW DENSITY CODES

Diplomová práce

Thesis

Vypracoval: Tomáš Filler
Vedoucí práce: Ing. Jessica Fridrich, PhD
Školní rok: 2006/2007

Název práce:
Minimalizace vlivu vložené informace ve steganografii pomocí řídkých kódů

Autor: Tomáš Filler

Obor: Softwarové inženýrství

Druh práce: Diplomová práce

Vedoucí práce: Ing. Jessica Fridrich, PhD. Department of Electrical and Computer Engineering, SUNY Binghamton, USA.

Abstrakt: Steganografie je věda zabývající se ukrýváním zpráv, jejichž přítomnost není možné odhalit. Jako médium zde používáme digitální obrázky a popisujeme obecný postup pro minimalizaci vlivu vložené informace. V první části práce popisujeme relevantní výsledky ze steganografie a dále se zabýváme bezpečností neviditelné komunikace. Ukazujeme, že problém minimalizace vlivu vložené informace je ekvivalentní problému binární kvantizace. Pro řešení tohoto problému představujeme nový algoritmus založený na lineárních kódech s řídkou generující maticí (LDGM kód). Tímto algoritmem, který jsme nazvali Bias Propagation (BiP), jsme výrazně snížili složitost problému binární kvantizace pomocí LDGM kódů. V práci uvádíme teoretickou analýzu tohoto algoritmu a dokazujeme nutnou podmínku pro jeho konvergenci. Tato podmínka udává tvar LDGM kódu, který je možné použít s BiP algoritmem. Prezentovaný algoritmus je 10–100 krát rychlejší než jakýkoliv jiný dosud publikovaný. V závěru práce přikládáme dosažené výsledky, které naše tvrzení podporují.

Klíčová slova: steganografie, matrix embedding, binární kvantizace, LDGM kódy, Bias Propagation

Title:
Minimizing Embedding Impact in Steganography Using Low Density Codes

Author: Tomáš Filler

Abstract: Steganography is the science of hiding information such that its presence cannot be detected. We use digital images, and describe the general approach for constructing steganographic schemes that minimizes the statistical impact of embedding. In the first part of this work, we summarize some relevant results from image steganography and study the security of undetectable communication. We show that the problem of minimizing the statistical impact of embedding is equivalent to binary quantization. We propose a new algorithm for solving the binary quantization problem using Low Density Generator Matrix (LDGM) codes. Using this algorithm, which we call Bias Propagation (BiP), we drastically reduce the complexity of binary quantization using LDGM codes. We present theoretical analysis of this algorithm. We derive a necessary condition for the BiP algorithm to converge. This condition describes the form of LDGM codes that can be used with this algorithm. In comparison to the state of the art work, our algorithm is 10–100 times faster. Achieved results and comparisons are presented at the end of this work.

Keywords: steganography, matrix embedding, binary quantization, LDGM codes, Bias Propagation

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze 5. května 2007

Tomáš Filler

Poděkování
S dokončením této práce myslím na to, jak poděkovat těm, bez kterých by tato práce jen stěží vznikala.

První, komu bych zde moc rád poděkoval, je má školitelka Jessica Fridrich. Bez její trpělivosti a zájmu bych nestrávil svůj poslední rok na univerzitě v americkém Binghamtonu. Byla to ona, kdo během posledního roku ovlivňoval mé myšlenky. Jsem rád, že jsem se od ní mohl učit, jak zprvu složité problémy řešit pomocí jednoduchých otázek. Dále bych jí chtěl poděkovat za cenné připomínky, které ovlivnily podobu této práce.

Během posledního roku jsem si uvědomil, jak moc mě ovlivnila cvičení, která jsem v průběhu studia vedl. Rád bych proto poděkoval i těm, se kterými jsem v té době spolupracoval.

Velké poděkování patří i mým rodičům. Bez jejich podpory a pomoci bych nebyl schopen uskutečnit své sny. Také bych rád poděkoval své manželce Radce za její pochopení a lásku. Byla mi oporou během našeho pobytu v Binghamtonu.

Acknowledgments
Completing this thesis reminds me how much I should express thanks to people without which this work would never be finished.

The first person I would like to mention is my supervisor, Jessica Fridrich. Without her patience and interest, I would not spend my final year at Binghamton University. It was her, who influenced my ideas and was always able to discuss the problems I had this year. I am glad that I could learn from her how to approach the complex tasks by asking simple questions. I would also like to thank her for the thorough proofreading which helped me to stress the important ideas in this work.

Within the last year, I realized how much I was influenced by several teaching possibilities offered to me during my university studies. They helped me to understand the situation from another point of view and that is why I would like to thank to all I worked with in that time.

Many thanks belong to my parents and to my wife. Without the support and love, my parents gave me, I would not be able to realize the dreams I had. Finally, I would like to thank my wife Radka for her support and understanding when we were in Binghamton. Her love and friendship made my life in Binghamton much happier.

At the end, I should mention one quote that I heard several times from my supervisor. I think that it describes this work in the best way.

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein

Contents

Introduction 1

1 Basics of image steganography 2

1.1 General concepts and the prisoner's problem . . . 2

1.2 Examples of steganographic schemes . . . 4

1.2.1 Embedding schemes for bitmap images . . . 5

1.2.2 Embedding schemes for JPEG images . . . 7

1.3 Steganalysis . . . 9

1.3.1 Analysis of LSB embedding . . . 10

1.3.2 Sample pairs analysis . . . 11

1.4 Adaptive steganography . . . 13

1.4.1 Public-key steganography . . . 14

1.4.2 Steganography with non-shared side-information . . . 17

2 Minimizing embedding impact 18

2.1 Cachin's model of steganographic security . . . 19

2.1.1 Review of Hypothesis Testing . . . 19

2.1.2 Information-theoretical model of steganographic security . . . 20

2.2 Improving embedding efficiency . . . 21

2.2.1 Basics of coding and information theory . . . 22

2.3 General approach for minimizing embedding impact . . . 24

2.3.1 Problem formulation . . . 25

2.4 Binary case of proposed framework . . . 27

3 Bias Propagation, an algorithm for binary quantization 31

3.1 Introduction to binary quantization . . . 31

3.1.1 Review of recent work and algorithms . . . 32

3.2 Graph representation of a code . . . 33

3.3 Motivation for solving MAX-XOR-SAT problem . . . 36
3.4 Bias Propagation, intuitive approach . . . 39
3.5 Weighted binary quantization . . . 44
3.6 Bias Propagation, formal derivation . . . 45
3.6.1 Sum-product algorithm . . . 45
3.6.2 Finding bitwise MAP using sum-product algorithm . . . 48
3.6.3 Dealing with cycles in a factor graph . . . 50
3.7 Convergence analysis . . . 51
3.7.1 Evolution over rounds . . . 59
3.8 Survey Propagation based quantizer . . . 60
3.9 Bias Propagation as a special case of SP based quantizer . . . 63

4 Determining the coset member and calculating syndromes 65
4.1 Algorithm for partial triangularization . . . 66
4.2 Practical results of triangularization . . . 68

5 Implementation and results 69
5.1 Implementation details . . . 69
5.2 Degree Distributions . . . 69
5.3 SP based quantizer vs. BiP algorithm . . . 73
5.4 Damping and restarting . . . 75
5.5 Decimation strategy analysis . . . 75
5.6 Codeword quantization . . . 76
5.7 Weighted case of Bias Propagation . . . 81
5.8 Application of proposed framework and results comparison . . . 82

Conclusion 86

Introduction

This thesis is motivated by the problem of near-optimal information hiding. We address the problem of undetectable message embedding into digital images (image steganography). To increase the security of this communication scheme, we minimize the number of changes that have to be made to embed a message. This minimization problem is equivalent to binary quantization (binary lossy compression).

In this thesis, we present a new approach for solving the binary quantization problem using Low Density Generator Matrix (LDGM) codes along with its theoretical analysis.

This work is structured as follows.

- The first chapter contains introduction to image steganography. We use so-called prisoners' problem to describe the model of invisible communication. We describe some methods for hiding information in digital images.
- The second chapter is devoted to the security of invisible communication. Here, we derive theoretical bounds on the minimal distortion, while embedding a message. We formulate the problem of optimal communication as the binary quantization problem using linear codes.
- In the third chapter, we describe the new algorithm for binary quantization – Bias Propagation (BiP). We give two different views on the derivation of this algorithm. Using the tools from modern coding theory, we study the convergence of this algorithm and derive the necessary condition. Finally, we explain tight connections between BiP and the "SP based quantizer" proposed by Wainwright et al. [24].
- The fourth chapter addresses the problem of message extraction in our steganographic scheme. Here, we describe the triangularization algorithm for sparse matrices. This approach allows us to extract the message in linear time.
- The last chapter contains implementation details of the Bias Propagation algorithm and presents numerical results. Here, we present the comparison of our work to the current state of the art in binary quantization. As a generalization, we show the results for weighted binary quantization.

Chapters 1 and 2 are introductory and summarize some relevant results from steganography. Chapter 3 contains the main contribution of this work – Bias Propagation algorithm and convergence analysis. Our approach for calculating syndromes using sparse generator matrix is introduced in Chapter 4. Chapter 5 presents the results obtained from highly optimized implementation of the proposed algorithm.

The work on this thesis was supported by Air Force Research Laboratory, Air Force Material Command, USAF, under the research grant FA8750-04-1-0112 and AFOSR grant number FA9550-06-1-0046.

Chapter 1

Basics of image steganography

The idea of information hiding is nothing new in the history. As early as in ancient Greece there were attempts to hide a message in trusted media to deliver it across the enemy territory. The idea of using vinegar as an invisible ink to write the message on a paper is really old. This message is invisible to human eye until we heat the paper and the vinegar turns dark. In the modern era of digital communication, we can think about similar ideas and use for example digital images as a medium for hiding a message. The word "steganography" comes from two Greek words: *steganos* (covered) and *graphos* (writing) and often refers to secret writing or data hiding.

In this thesis, we will constrain ourselves to (digital) image steganography (further referred as steganography) only to be able to practically demonstrate our results. The approach and final results from this thesis can be easily used for steganography in other digital media, such as video, music etc.

In this chapter, we will describe the basics of image steganography and give some examples of algorithms how to hide data into an image. We will start by defining some basic concepts of information hiding in terms of the so-called prisoners' problem and giving the notation which will be used throughout the whole thesis. Furthemore, we will describe some techniques how to detect the presence of a message and hence how to break steganographic schemes. Finally, we will mention the problem of "public key steganography", where similarly to public key cryptography we want to use different key (public key) to embed the message and another key (private key) to extract the message.

1.1 General concepts and the prisoner's problem

Although steganography and cryptography are similar in that both are used for communicating a secret message, the concept of steganography is different. In the case of cryptography, the presence of the message in some data stream is obvious, but without the correct key the message is unintelligible. The goal of steganography is to communicate the secret message, which is hidden in some media, without being able to detect the presence of this message.

We can formalize the concept of steganography using the so-called *prisoners' problem* defined by Simmons [23]. The diagram of this problem is shown in Figure 1.1. Assume that two prisoners Alice and Bob want to create an escape plan. They are in separate prison cells, but they can communicate by sending messages. Each message is examined by warden Wendy. Wendy is a *passive warden* and her goal is to examine each message and deliver it when it

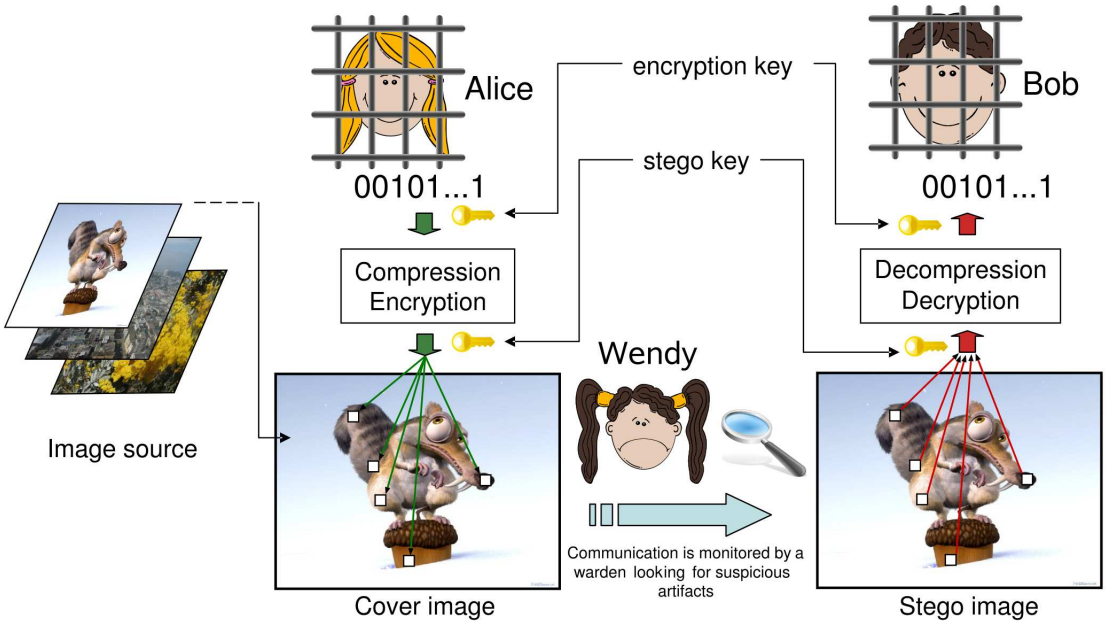


Figure 1.1: Prisoner's problem diagram.

does not seem suspicious to her. When the message seems suspicious, Wendy can investigate some resources to extract the hidden message and the whole escape plan will be revealed. Usually, we say Wendy cuts the communication channel.

In our model of image steganography, Alice wants to send message M to Bob and she does so by *embedding* it into an image. The image used for embedding the message is called the *cover image*, where the resulting image with embedded message is called the *stego image*. When Alice communicates with Bob, we assume that the algorithm used for embedding is known to Wendy, however the key used by this algorithm is shared and keep secret. This key can be represented as a password and used for initiating a pseudo-random number generator. Further, we can use the output of the pseudo-random number generator for selecting the pixels that will be modified. This assumption, that the algorithm used for embedding is known to warden, is known as Kerchoff's principle and is also used in cryptography. In this model, Wendy should not distinguish between cover and stego images.

In steganography, there are generally three ways how to embed a message into a cover image. We can do that by:

- *cover selection* where Alice selects an appropriate image that will communicate the desired message
- *cover synthesis* in this case the embedder has to create the image with embedded message
- *cover modification* this case is the most frequently used one, where Alice has some large source of images and she embeds the message into an arbitrarily picked image by modifying some pixels.

In this work, we focus on steganography by cover modification. Throughout the text, boldface symbols denote vectors or matrices. To formalize the process of modifying the cover image, we will introduce the concept of *steganographic scheme*. We represent the cover image as

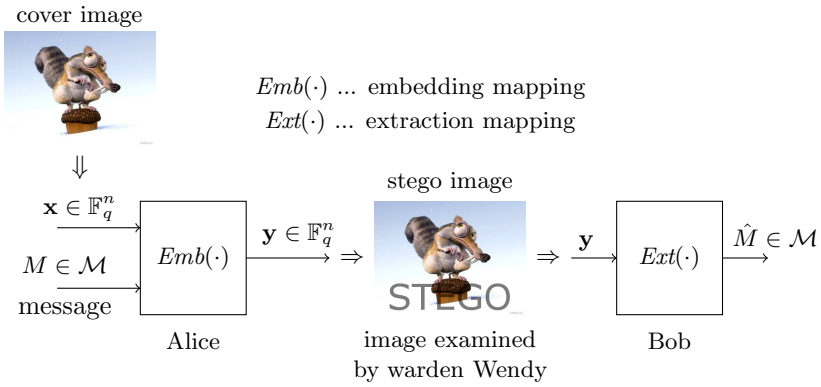


Figure 1.2: Model of steganographic scheme.

a sequence of n elements $\mathbf{g} = \{\mathbf{g}_1, \ldots, \mathbf{g}_n\} \in \mathcal{G}^n$, where $\mathcal{G} = \{0, 1, \ldots, 2^r - 1\}$ and r is the number of bits needed to describe each element. Most steganographic methods work with a finite-field representation of $\mathbf{g}$ obtained through some symbol-assignment function $symb : \mathcal{G} \to \mathbb{F}_q$. As an example we can mention $symb(\mathbf{g}_i) = \mathbf{g}_i \mod 2$ or $symb(\mathbf{g}_i) = \mathbf{g}_i \mod 3$, where the function assigns bit or ternary symbol to each cover element. Thus the cover image $\mathbf{g}$ can be represented as a vector $\mathbf{x} \in \mathbb{F}_q^n$.

A steganographic scheme is a pair of *embedding* and *extraction* mappings $Emb : \mathbb{F}_q^n \times \mathcal{M} \to \mathbb{F}_q^n$, $Ext : \mathbb{F}_q^n \to \mathcal{M}$ satisfying

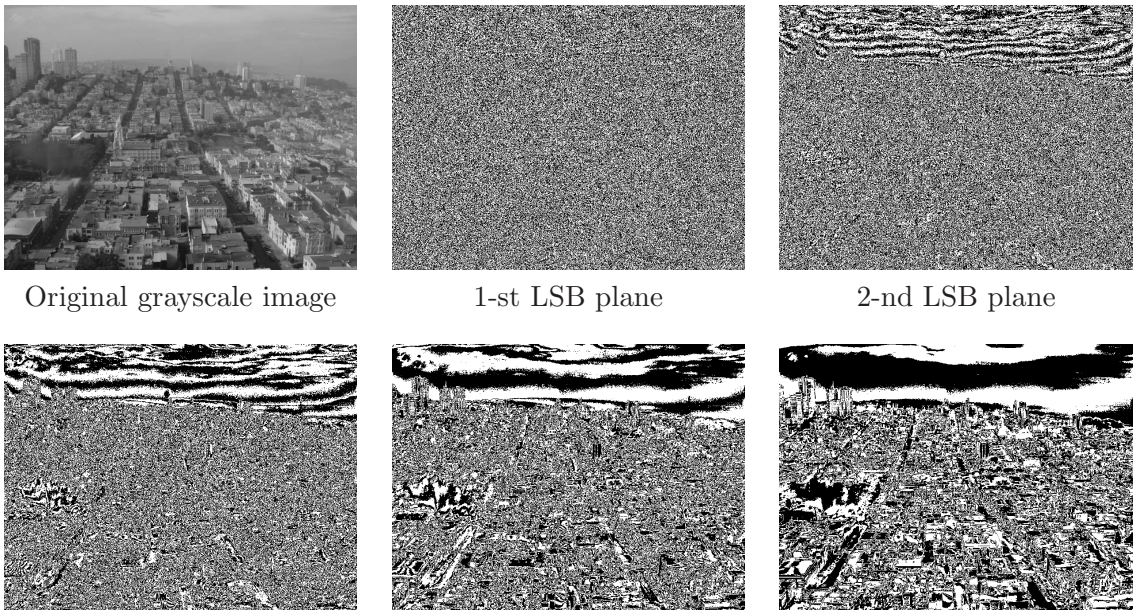
$$Ext(Emb(\mathbf{x}, M)) = M \qquad \forall \mathbf{x} \in \mathbb{F}_q^n, \ \ \forall M \in \mathcal{M}, \tag{1.1}$$

where $\mathcal{M}$ is the set of all messages M that can be communicated. We say that the embedding capacity of the scheme is $\log |\mathcal{M}|$ bits. In Section 1.2 we give some examples of steganographic schemes that can be used for embedding messages into bitmap and JPEG images that fulfill this definition.

At this point, we can see that steganographic schemes should be designed for each specific image format, because each format offers different possibilities for information hiding. In fact, in steganography we are exploiting the unused "space" which we can obtain from each image, and we are filling this space with our message. Here, we try to unify the approach by defining the embedding scheme and hence handle different formats and embedding methods using one notation.

1.2 Examples of steganographic schemes

In this section, we mention some well known steganographic schemes that can be applied to bitmap images. These schemes were mostly developed by students to demonstrate the simplicity and power of information hiding. Further, we will see that although these methods do not produce visible artifacts on common images, they are not secure in a steganographic sense. At the end, we will discuss some schemes designed for usage with JPEG images. JPEG format is interesting for steganography, because most common images use this graphical format for its compression and therefore it will not be suspicious to send JPEG images.



Original grayscale image | 1-st LSB plane | 2-nd LSB plane

3-rd LSB plane | 4-th LSB plane | 5-th LSB plane

Figure 1.3: Five least significant bitplanes of a bitmap image.

1.2.1 Embedding schemes for bitmap images

To develop the first embedding scheme, we can use the fact that human eye is not sensitive to small changes of each pixel. To demonstrate this fact, assume we have an 8-bit grayscale image that was never compressed (e.g., obtained using a scanner). Let us have a look at the least-significant bit of each pixel. The image formed from these bits will most likely look like a random noise because these pixels do not carry the main image structure. Generally, we can obtain the so-called i-th least-significant bit (LSB) plane by taking the i-th bit from the binary expansion of each pixel's color. In Figure 1.3, we can see a grayscale image and its first to fifth LSB planes. Using this observation, we can construct the so-called "LSB embedding" scheme in which the message $M \in \mathcal{M}$ is represented using a binary sequence $\mathbf{m} \in \{0,1\}^m$, $m < n$, that replaces the first m bits in the LSB plane (first LSB plane). This operation slightly modifies the cover image, but the LSB plane will contain our message. To obtain this message back, Bob has to extract the first m bits from the LSB plane.

Up to now, Alice did not need any key for embedding a message. We can improve this method by using a shared key to generate a pseudo-random permutation of n elements and change the first m pixels given by this permutation. Due to the same key, Bob can construct the same permutation and extract the bits from the correct pixels in the correct order. To express the relative length of embedded message, we define the *relative message length* as $\alpha = \frac{m}{n}$. This ratio can be either pre-agreed, or a small, key-dependent portion of the cover can be reserved to communicate a suitably quantized α encoded using a few bits. Therefore, we can assume that Bob knows this value.

In Figure 1.4, we show a complete Matlab implementation of embedding and extraction mapping. Function `lsb_embed` takes the cover image (matrix `C`), the message (vector `M`), and the key and outputs the stego image `S`. The message can be extracted using `lsb_extract` function, which takes the stego image (matrix `S`), relative message length, and the secret key. This implementation uses the key to generate a pseudo-random permutation and embeds the message along this path. Finally, we use the following compact notation of embedding

```
function S = lsb_embed( C, M, key )
  old_seed = rand('seed');
  rand('seed', key);

  S = C;          %initialize stego image
  n = numel(C);   % # pixels in cover image
  m = numel(M);   % # bits in message
  path = randperm(n);
  path = path(1:m);
  S(path) = (C(path)-mod(C(path),2))+M(1:m);

  rand('seed', old_seed);
end
```

```
function M = lsb_extract(S, q, key)
  old_seed = rand('seed');
  rand('seed', key);

  n = numel(S);   % # pixels in stego image
  m = floor(q*n); % # bits in message
  path = randperm(n);
  path = path(1:m);
  M = mod(S(path), 2); % extract the message

  rand('seed', old_seed);
end
```

Figure 1.4: Matlab implementation of LSB embedding algorithm.

operation (Matlab notation):

$$\texttt{S(i)} = \texttt{C(i)} - \texttt{mod(C(i),2)} + \texttt{M(i)}. \tag{1.2}$$

Although it is not clear now, this approach of hiding message into the LSB plane is due to this embedding operation (replacing the least-significant bit) highly detectable. In section 1.3, we will show an algorithm that will be able to estimate the relative message length. Therefore, Wendy can reject to forward the message to Bob based on a given threshold.

According to (1.2), when we want to embed $\mathbf{m}_i = 0$ into a pixel with an odd color, we always subtract one and, on the contrary, when we want to embed $\mathbf{m}_i = 1$ into a pixel with an even color, we always add one. This behavior is due to the replacement of the least-significant bit by our message bit. There is another way how we can do the change when the least-significant bit does not match. We can randomly subtract or add one. If we omit the extreme cases, this operation can change more bits in the binary representation, however, the absolute value of the color changes will be one.

Based on the described idea, we can come up with another steganographic scheme called "plus minus one embedding" (± 1 embedding). We describe the embedding operation using the Matlab code from Figure 1.5, where the extraction mapping is the same as in LSB embedding. In this algorithm, we use the key in the same way to generate the pseudo-random path for embedding each bit. The embedding operation is realized by adding ± 1 randomly to each

```
function [ S ] = pm1_embed( C, M, key )
  old_seed = rand('seed');
  rand('seed', key);

  S = double(C);      % stego image initialization
  n = numel(C);       % # pixels in cover image
  m = numel(M);       % # bits in message
  path = randperm(n);
  path = path(1:m);
  E = (-1).^round(rand(size(M)));
  E(C(path)==0) = +1;
  E(C(path)==255) = -1;
  E(mod(C(path),2)==M) = 0;
  S(path) = double(C(path)) + E;

  rand('seed', old_seed);
end
```

```
function M = pm1_extract(S, q, key)
  % extraction mapping is the same
  % as for LSB embedding
  M = lsb_extract(S, q, key);
end
```

Figure 1.5: Matlab implementation of ± 1 embedding algorithm.

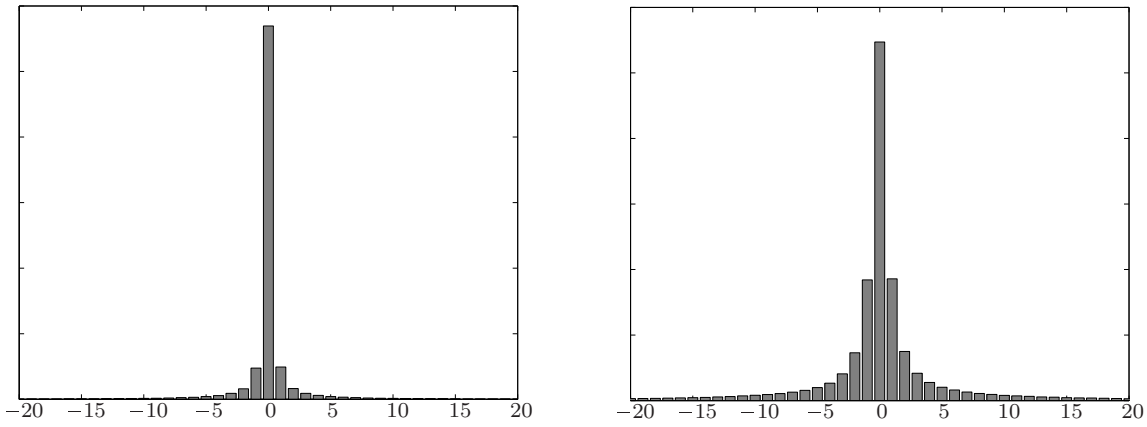


Figure 1.6: Histograms of DCT coefficients for JPEG compressed image from Figure 1.3 for two different quality factors.

pixel whenever the least-significant bit does not match our message bit. Finally, we redefine the embedding operation on boundaries, where for 8-bit images we always add one, when we want to change pixel with color 0 and subtract one, when we want to change pixel with color 255. We have to do this because changing 0 to 255 will be visible in the stego image.

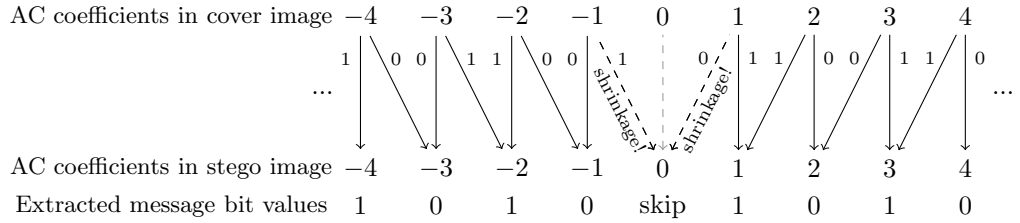
1.2.2 Embedding schemes for JPEG images

Now we know how to hide information into bitmap images, however for practical scenarios where most images are stored in the JPEG format we should have some algorithms for hiding messages into this type of images. The JPEG format uses the Discrete Cosine Transformation (DCT) to transform the original image divided into 8×8 pixel blocks to the frequency domain and perform lossy compression by quantizing each coefficient with a given factor. Finally, the sequence of quantized coefficients is losslessly compressed using run-length encoding and Huffman coding. In this section, we will only need to work with quantized coefficients. Due to the DCT transformation and quantization, the histogram of DCT coefficients has a characteristic shape that can be seen in Figure 1.6. Any steganographic scheme should not change this shape too much during embedding because Wendy can use the difference to distinguish between cover and stego images.

The first steganographic scheme for hiding messages into JPEG images is the simple LSB embedding of DCT coefficients. In this case, we have to be carefull because we cannot change all coefficients without introducing some visible artifacts. First, we have to omit DC coefficients, because these coefficients contain information about the mean color of each 8×8 block and their changes would be visible in the stego image. Next, we have to omit all zero coefficients, because there are too many of them in the image. We are omitting all coefficients equal to one, because values 0 and 1 are coupled by the embedding operation (form so-called *LSB pair*). The remaining coefficients are free to change. We call this algorithm *J-steg*.

In section 1.3, we will see that this algorithm can be reliably detected due to the nature of the embedding changes. The idea is that the type of embedding change should simulate some natural process that can happen to the image, therefore Wendy cannot recognize the stego image. Next, we describe the so-called F5 embedding algorithm published by Westfeld [25]. This algorithm tries to mask the embedding by lowering the absolute value of DCT

(a) Embedding changes in JPEG AC coefficients using F5 algorithm



(b) Example of embedding message "01110" using F5 algorithm

Figure 1.7: Example of embedding using the F5 algorithm.

coefficients and hence preserves the shape of the histogram. After embedding, the stego image looks like the original image compressed with lower quality factor, which is a natural process (Figure 1.6).

In this algorithm, we use the same argument for omitting changes in DC coefficients and changes in all zero AC coefficients. We use the rest of DCT coefficients to hide our message. The message is hidden in LSBs of allowed coefficients along some pseudo-random path by decreasing the absolute value of each coefficient if the LSB does not match our message bit. Because we are not using zero coefficients for embedding, the receiver (Bob) will extract the message from all non-zero AC coefficients as their LSBs. Using our embedding operation it can happen that we want to embed 0 into the 1 or -1 coefficient and we should replace the coefficient with a zero. This situation, called "shrinkage", causes problems, because Bob will not read the message bit from this zero coefficient. Therefore, we have to re-embed this message bit again. Due to shrinkage, we are losing capacity the number of possible coefficients that can be used for embedding. Another problem is that shrinkage occurs only when we are embedding 0s, therefore due to re-embedding, odd coefficients are more likely to change than even coefficients. This fact further causes "staircase" artifacts in histogram, which are not natural. The F5 algorithm solves this problem by redefining the LSB of each DCT coefficient as $LSB(x) = x \mod 2$ for $x > 0$ and $LSB(x) = 1 - (x \mod 2)$ for $x < 0$. Using this definition shrinkage occurs when we want to embed 0 into $x = 1$ or 1 into $x = -1$ which is equally likely and does not cause any artifacts in histogram. To summarize the embedding process, Figure 1.7 contains a complete example of the embedding using F5 algorithm. Finally, when Bob wants to extract the message he skips all DC and all zero AC coefficients and obtains the message by applying the modified LSB function along the same pseudo-random path as was used for embedding.

The type of the embedding change used in F5 was not the only one new idea that was

introduced. F5 came with an idea how to embed more bits using fewer changes in the cover image. Up to now, when we want to embed a message with relative length $\alpha < 1$, we simply skip $1-\alpha$ portion of pixels (non-zero AC coefficients) and use the rest for embedding. When we assume that the message is random unbiased bitstream, then we had to change every other pixel (coefficient) in this part. In fact, we can use the skipped part for embedding too and achieve a smaller number of changes. We introduce the approach using a simple example of how to embed 3 bits into 7 pixels making at most 1 change (relative message length is $\alpha = \frac{3}{7}$). Assume we have 7 AC coefficients transformed using LSB function into the sequence $\mathbf{x} = (1,0,1,1,1,1,1)^T$ and we want to embed the message $\mathbf{m} = (0,0,1)^T$. We construct the matrix

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

and do the embedding in the following way: (1) calculate $\mathbf{h} = \mathbf{Hx} + \mathbf{m}$ in binary arithmetic, (2) change the i-th non-zero AC coefficient, where i is the number of column in matrix $\mathbf{H}$ which equals to $\mathbf{h}$. In our example, we have $\mathbf{h} = (0,1,0)^T$, therefore we change the 6-th coefficient and output the modified sequence as a stego image. The receiver will extract the message as $\mathbf{m} = \mathbf{Hy}$, where $\mathbf{y}$ is a sequence of modified LSBs of non-zero AC coefficients. Step (2) is always possible to perform because matrix $\mathbf{H}$ contains all combinations of 1s and 0s as its columns. From this example, we can see that we needed only one change to embed 3 bits, instead of 1.5 change (on average) per embedded bit. This idea was called "Matrix embedding" and we will study this approach in more detail in Section 2.2.

1.3 Steganalysis

In the previous section, we described some steganographic techniques that can be used for hiding information in digital images. We constructed these schemes in such a way that no visible artifacts were introduced by embedding a message. This is obvious because Wendy could use these artifacts to distinguish between cover and stego image which we do not want to. Wendy is not constrained only to visible artifacts, however she can use an arbitrary algorithm to do the detection. Here, we assume that Wendy knows the embedding algorithm that Alice and Bob are using and hence she can focus on some other artifacts made to the stego image. Using this approach, Wendy is interested in science called *steganalysis*. Steganalysis is a counterpart to steganography and it studies methods how to reliably detect presence of message in stego images. When we are able to reliably distinguish between cover image and stego image then we say that the method used for embedding a message is *detectable* and hence not secure.

In this section, we will show the basic tool used for reliable estimation of the relative message length when we use the LSB embedding algorithm which we have discussed before. This method uses analysis of the embedding operation (LSB flipping) and quantifies the number of artifacts made during the embedding and hence estimates the number of bits embedded in the image. In the rest of this section, we will go through the analysis of LSB embedding algorithm and finally we will describe so-called "Sample pairs analysis" [7], [17] which can be used for estimating the relative message length.

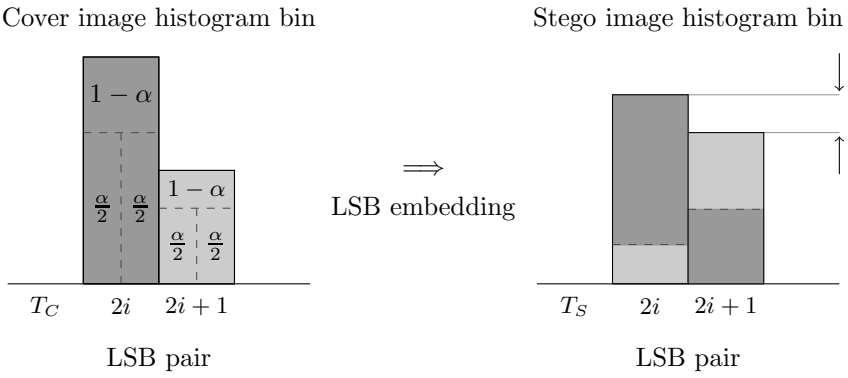


Figure 1.8: Impact of LSB embedding on pair of histogram bins.

1.3.1 Analysis of LSB embedding

LSB embedding uses LSB flipping as the core embedding operation. This operation flips the LSB bit when this bit does not match our message bit. Due to this definition, we obtain so-called *LSB pairs* of pixel values as the set $\{(0\ 1), (2\ 3), (4\ 5), \ldots\}$. The embedding operation has the property that each pixel value stays in its LSB pair after embedding. When we want to embed a bit to pixel with color 2, then the pixel's value will always stay in the same LSB pair (2 3) after embedding and similarly with other values. In further description, we will assume that we have an 8-bit grayscale images and denote $T_C[i]$ the number of pixels in cover image with color i and similarly $T_S[i]$ for stego image. Here, we call T_C *type of image* C. Type is unnormalized histogram. Using this notation and the property of LSB pairs, we can write

$$T_C[2i] + T_C[2i+1] = T_S[2i] + T_S[2i+1] \qquad \forall i = 0 \ldots 127. \tag{1.3}$$

It means that the number of pixels in each LSB pair is the same for cover and stego image, because no pixel can be removed from this pair during embedding. This sum is preserved for arbitrary relative message length α.

To show the first aspect of this embedding scheme, we consider the message $\mathbf{m}$ to be an unbiased random sequence of $\{0, 1\}$. Suppose that we are embedding the message with relative message length α. Then, the average histogram T_S of the stego image can be expressed in the following way:

$$T_S[2i] = (1-\alpha)T_C[2i] + \frac{\alpha}{2}T_C[2i] + \frac{\alpha}{2}T_C[2i+1] \tag{1.4}$$

$$T_S[2i+1] = (1-\alpha)T_C[2i+1] + \frac{\alpha}{2}T_C[2i] + \frac{\alpha}{2}T_C[2i+1] \tag{1.5}$$

for each histogram bin pair $i = 0, \ldots, 127$. We obtain these equation using the following idea (see Figure 1.8). When we are embedding a message with relative message length α, we do not change the ratio $1-\alpha$ of all pixels with color $2i$, but we process α portion of that. Through this processing, we have equal probability (message is unbiased bitstream) of seeing pixel with correct LSB ($\frac{\alpha}{2}T_C[i]$) which we do not change and with incorrect LSB which we flip from $2i$ to $2i+1$. Similarly, for the case with color $2i+1$. A special case of these equations occurs for a fully embedded image, where $\alpha = 1$. In this case, the LSB embedding algorithm completely evens out the bins of the type (histogram) of stego image. The change of type bins can be seen from Figure 1.9, where we have the type of the original

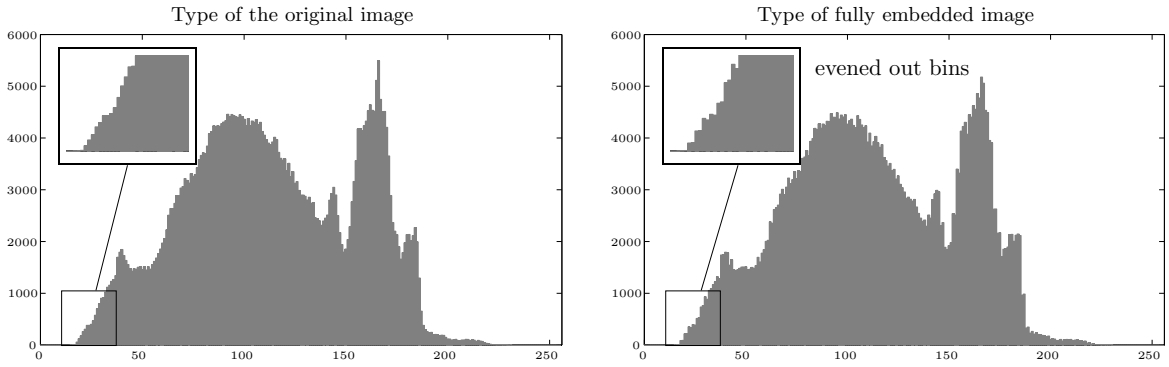


Figure 1.9: Type comparison of cover and fully embedded stego image.

image (from Figure 1.3) and type of a fully embedded stego image.

The process of evening out the histogram bins for each LSB pair is not natural for digital images and hence we find the first artifact that is introduced by LSB embedding. We can use this artifact and somehow measure the gap between each bin, but it will not be reliable method. Nevertheless we can use this artifact to derive an upper bound for relative message length α. Suppose that $T_S[2i] > T_S[2i+1]$ and calculate $T_S[2i] - T_S[2i+1]$ using equations (1.4) and (1.5). We obtain

$$T_S[2i] - T_S[2i+1] = (1-\alpha)\Big(T_C[2i] - T_C[2i+1]\Big) \leqslant (1-\alpha)\Big(T_C[2i] + T_C[2i+1]\Big)$$

thus using (1.3) ,we obtain

$$\alpha \leqslant \frac{2T_S[2i+1]}{T_S[2i] + T_S[2i+1]}.$$

We can do the same for case $T_S[2i] < T_S[2i+1]$, finally we get the following upper bound

$$\alpha \leqslant 2\frac{\min\{T_S[2i], T_S[2i+1]\}}{T_S[2i] + T_S[2i+1]}.$$

In next section, we will come with a better approach how to estimate the relative message length.

1.3.2 Sample pairs analysis

This section is devoted to better analysis of the LSB flipping operation done by Wu [7] and further improved by Lu [17]. The idea of this method is to develop a measure that will be preserved by natural images and that will be disturbed by stego images produced by the LSB algorithm. This is done by analysing values on pairs of pixels. This idea was further generalized to so-called triples analysis by Ker [15].

We start by dividing the image into pairs of neighboring pixels and denote this pair by $(u, v) \in \mathcal{P}$, where $\mathcal{P}$ is the set of all pixel pairs in the image. The size of the set $\mathcal{P}$ is $\frac{n}{2}$, where n is the number of pixels. Next, we divide the set $\mathcal{P}$ into three disjoint subsets $\mathcal{P} = \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$

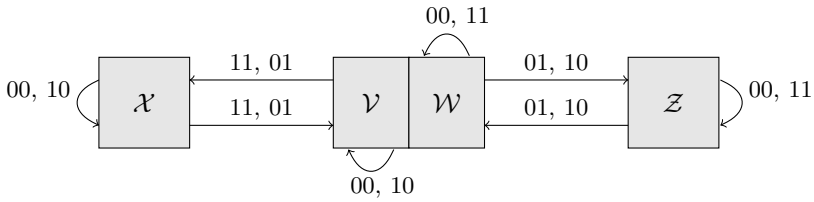


Figure 1.10: Pixel pair transition diagram for Sample Pairs analysis.

based on the following definition

$$\begin{aligned}(u,v) \in \mathcal{X} \quad &\Leftrightarrow \quad \Big(u < v \text{ and } v \text{ is even}\Big) \text{ or } \Big(u > v \text{ and } v \text{ is odd}\Big) \\ (u,v) \in \mathcal{Y} \quad &\Leftrightarrow \quad \Big(u < v \text{ and } v \text{ is odd}\Big) \text{ or } \Big(u > v \text{ and } v \text{ is even}\Big) \\ (u,v) \in \mathcal{Z} \quad &\Leftrightarrow \quad u = v\end{aligned} \tag{1.6}$$

and further divide set $\mathcal{Y} = \mathcal{V} \cup \mathcal{W}$, where

$$\begin{aligned}(u,v) \in \mathcal{W} \quad &\Leftrightarrow \quad (u,v) = (2k, 2k+1) \text{ or } (u,v) = (2k+1, 2k) \\ (u,v) \in \mathcal{V} \quad &\Leftrightarrow \quad (u,v) \notin \mathcal{W}.\end{aligned} \tag{1.7}$$

Although the definition looks complicated, these sets have important properties which we demonstrate using Figure 1.10. In this figure, we have the transition diagram which can be interpreted in the following way. The pixel pair $(u, v) \in \mathcal{X}$ can change the set, if the modification pattern for changing LSB is 11, or 01 (both pixels or only the second pixel are changed). Assume the modification pattern 11 from the definition of set $\mathcal{X}$ we can see that either v is odd or v is even. When v is odd ($u > v$), then by flipping LSB of odd number we get smaller even number and by flipping LSB of u, we cannot obtain $v + 1$, hence the inequality $u > v$ still holds and the modified (u, v) belongs to $\mathcal{W}$. Using a similar approach, we can prove the complete transition diagram. Another important aspect which we can see from the diagram is, that sets $\mathcal{X} \cup \mathcal{V}$ and $\mathcal{W} \cup \mathcal{Z}$ are closed to arbitrary LSB embedding.

To express the relative message length α only using a stego image we use $\mathcal{X}, \mathcal{Y}, \mathcal{V}, \mathcal{W}, \mathcal{Z}$ to denote the previously defined sets calculated from the cover image and $\mathcal{X}', \mathcal{Y}', \mathcal{V}', \mathcal{W}', \mathcal{Z}'$ to denote same sets calculated from the stego image. Our goal is to express α in terms

```
function alpha = sp(S)       % S is input matrix - grayscale image
  L = double(S(:,1:2:end));  % set of left values in each pair
  P = double(S(:,2:2:end));  % set of right values in each pair
  p = numel(L);              % total number of pairs
  % size of each sample set
  z = sum(sum(L==P));        % size of sample set Z
  x = sum(sum( bitor(bitand(L<P, mod(P,2)==0), bitand(L>P, mod(P,2)==1)) )); % size of sample set X
  % size of sample set W
  w = sum(sum(bitor(bitand(bitand(L<P,mod(P,2)==1),L-P==-1),bitand(bitand(L>P,mod(P,2)==0),L-P==1))));
  y = p-z-x;          % size of sample set Y
  % solve quadratic equation
  q = [-2*x+p+sqrt((2*x-p)^2-2*(w+z)*(y-x)),-2*x+p-sqrt((2*x-p)^2-2*(w+z)*(y-x))]/(w+z);
  alpha = max([0, min([real(q(1)) real(q(2))])]);  % return correct rel. msg. length
end
```

Figure 1.11: Matlab implementation of Sample Pairs analysis.

of prime sets, because these sets can be calculated by Wendy (warden does not have the original cover image). When we are embedding unbiased message, every other visited pixel is changed during the embedding, hence the probability of seeing the flipping patterns 11 and 00 in the stego image is $\left(\frac{\alpha}{2}\right)^2$, $\left(1-\frac{\alpha}{2}\right)^2$, respectively, and similarly for other patterns. Using this result and the transition diagram from Figure 1.10, we can write the expected size of sets $\mathcal{X}', \mathcal{Y}', \mathcal{W}'$

$$|\mathcal{X}'| = |\mathcal{X}|\left(1-\frac{\alpha}{2}\right)+|\mathcal{Y}|\frac{\alpha}{2} \tag{1.8}$$

$$|\mathcal{Y}'| = |\mathcal{Y}|\left(1-\frac{\alpha}{2}\right)+|\mathcal{X}|\frac{\alpha}{2} \tag{1.9}$$

$$|\mathcal{W}'| = |\mathcal{W}|\left(1-\alpha+\frac{\alpha^2}{2}\right)+|\mathcal{Z}|\alpha\left(1-\frac{\alpha}{2}\right). \tag{1.10}$$

For natural images, there is no reason why the size of sets $\mathcal{X}$ and $\mathcal{Y}$ should differ. Hence, we have

$$|\mathcal{X}| = |\mathcal{Y}| \quad \Rightarrow \quad |\mathcal{X}| = |\mathcal{Y}| + |\mathcal{W}|. \tag{1.11}$$

Subtracting equations (1.8) and (1.9) we obtain

$$|\mathcal{X}'| - |\mathcal{Y}'| = \big(|\mathcal{X}| - |\mathcal{Y}|\big)(1-\alpha). \tag{1.12}$$

When we substitute from (1.11) we can rewrite the last equation in the following form

$$|\mathcal{X}'| - |\mathcal{Y}'| = |\mathcal{W}|(1-\alpha). \tag{1.13}$$

Here, we have to find another equation for $|\mathcal{W}|$. Using (1.10) we can write

$$\begin{aligned}|\mathcal{W}'| &= |\mathcal{W}|\left(1-\alpha+\frac{\alpha^2}{2}\right)+|\mathcal{Z}|\alpha\left(1-\frac{\alpha}{2}\right) = |\mathcal{W}|\left(1-\alpha+\frac{\alpha^2}{2}\right)+\big(\gamma-|\mathcal{W}|\big)\alpha\left(1-\frac{\alpha}{2}\right) = \\ &= |\mathcal{W}|(1-\alpha)^2+\gamma\alpha\left(1-\frac{\alpha}{2}\right),\end{aligned}$$

where $\gamma = |\mathcal{W}|+|\mathcal{Z}| = |\mathcal{W}'|+|\mathcal{Z}'|$, which is a known value. Finally, by substituting (1.13) into the last equation, we obtain the following cubic equation for the unknown relative message length α

$$\frac{1}{2}\gamma\alpha^2 + \big(2|\mathcal{X}'| - |P|\big)\alpha + |\mathcal{Y}'| - |\mathcal{X}'| = 0, \tag{1.14}$$

where all coefficients can be calculated from the stego image. To obtain the correct estimation of the parameter α, we have to take the smaller real part from both roots.

In Figure 1.11, we present the complete Matlab code of the above procedure. To show the reliability of this method, Figure 1.12 presents the result from embedding a message with different α into 995 RAW images. According to [15] the smallest message which can be reliably detected using the Sample Pairs or Triples analysis is roughly $\alpha = 0.05$ for RAW images and even smaller for other image sources. This result suggests that using LSB algorithm for embedding messages is highly detectable and we should rather avoid using this approach.

1.4 Adaptive steganography

Up to now, we assumed that in steganography the embedding process was non-adaptive. It means that we did not consider the position, where each embedding change was done.

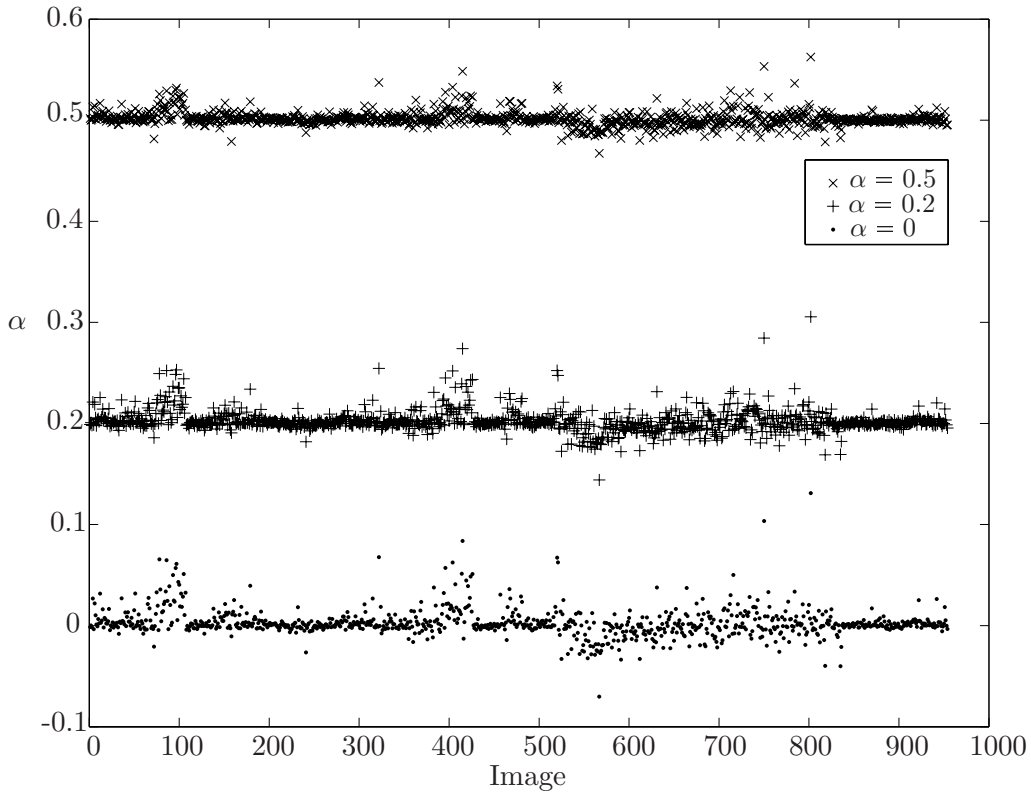


Figure 1.12: Detection accuracy of Sample Pairs analysis for grayscale images.

The problem with adaptivity is that when Alice embeds information into some pixels, these pixels are selected according to some side information which does not need to be known to Bob, and hence Bob does not know from which pixels he should extract the message. In this section, we describe two basic scenarios, where we need the adaptivity of embedding changes and how to approach this problem.

1.4.1 Public-key steganography

First, we describe the public-key steganography. In prisoners' problem (Figure 1.1), we had the assumption that both Alice and Bob are sharing same key (it could be some password) which was used for selecting pixels for message embedding and extraction. This communication model is similar to private-key cryptography, where both parties have the same key, but we can imagine the public-key communication model in steganography too. Assume that Bob has two keys, which we call *public* and *private* key and he shares his public key with everyone (warden can obtain this key as well). According to public-key cryptography, we have the following communication model: when Alice (as well as other prisoners) wants to send a message to Bob, she takes Bob's public key and use this key for embedding a message and produces stego image. Next the stego image is delivered by warden to Bob. The goal of this scheme is to be able to invisibly communicate in a way that only Bob can extract the message from stego image using his private key. The goal of the warden Wendy is to be able to distinguish between cover and stego images, where she knows the embedding algorithm and Bob's public key. This communication model is know as *public-key steganography*

and is an interesting case of invisible communication. The original communication model as introduced in Section 1.1 will be called *private-key steganography*.

As in private-key steganography, we will use cryptography and data compression for encrypting the original message (see Figure 1.1). When embedding a message using Bob's public key, we will use the public-key version of cryptography to encrypt the original message. By compressing and encrypting the message, we can assume that the resulting bitstream is random (all good encrypting schemes have this property). The key problem here is how to embed random bitstream message, such that the warden cannot detect anything using steganalysis, but the warden knows exactly how Alice did the embedding. We cannot use the same approach and simply embed the message along some pseudo-random path, because this path has to be generated from the public key which is known by the warden. If we still use this approach, the warden can select pixels from this path and use this path as an input to some steganalytic algorithm. By restricting to this path, we simulate the embedding using relative message length $\alpha = 1$, because each pixel was used for embedding and thus the warden can easily detect the presence of some message in this path. This scheme will be easily detectable and thus we have to redefine the way how to choose pixels for embedding. We will show the whole approach using LSB flipping as an embedding operation, but it can be generalized and easily used with other types of changes (± 1).

We want to embed a binary message $\mathbf{m} \in \{0,1\}^m$ into the cover image represented as $\mathbf{x} \in \{0,1\}^n$. First, we divide the cover image into m disjoint blocks of approximately the same size which we denote by $\mathcal{B}_j$, $\{1,\dots,n\} = \bigcup_{j=1}^{m} \mathcal{B}_j$. This division is deterministic and is part of the whole scheme. For example, $\mathcal{B}_j$ is a block of $k \times k$ neighboring pixels. For each block, we define $LSB(\mathcal{B}_j)$ as XOR of all pixel LSB values, $LSB(\mathcal{B}_j) = \left(\sum_{i \in \mathcal{B}_j} \mathbf{x}_i\right) \mod 2$. Finally, we embed one message bit into each block sequentially in such a way that $LSB(\mathcal{B}_j) = \mathbf{m}_j$. For some $j \in \{1,\dots,m\}$, when the LSB of block j does not match the message bit, we flip the LSB of an *arbitrary* pixel in this block and thus $LSB(\mathcal{B}_j) = \mathbf{m}_j$, otherwise we do not change any pixel in the block.

To study the detectability of this scheme, we switch the role and assume image that we do not know whether it is cover or stego generated by this method. If it is a stego image, then we cannot separate the set of visited and unvisited pixel as was in the previous case, because when the LSB of some block did not match the message bit, Alice changed an arbitrary pixel from this block. We still can apply steganalysis on the whole image as in the case of private-key steganography, but this is all we can do. Possibly, we have another statistics we can use to distinguish between cover and stego and it is the statistics of $LSB(\mathcal{B}_j)$, because by embedding we are changing these bits and we can expect that this statistics will be changed by embedding. Unfortunately, even for relatively small blocks, the set of $LSB(\mathcal{B}_j)$ is random bitstream and by embedding random message we obtain statistically same bitstream and hence it is almost impossible to use this information for detection. To show this, consider the following theorem.

Theorem 1.1 (Power of parity) *Assume we have a biased bitstream* $\mathbf{x}$*, where we can see* 0 *with probability* P_0 *and* 1 *with* $P_1 = 1 - P_0$*. Let* $\mathbf{y}$ *be another bitstream constructed from* $\mathbf{x}$ *as a blockwise parity for some block size* k

$$\mathbf{y}_i = \left(\sum_{j=(i-1)*k+1}^{i*k} \mathbf{x}_j \right) \mod 2,$$

then the resulting bitstream is randomized exponentially fast with respect to block size k*.*

$$|Pr\{\mathbf{y}_i = 0\} - Pr\{\mathbf{y}_i = 1\}| = |1 - 2P_1|^k \tag{1.15}$$

Proof: First we derive the expression for $\Pr\{\mathbf{y}_i = 1\}$.

$$\begin{aligned}
Pr\{\mathbf{y}_i = 1\} &= Pr\left\{ \sum_{j=(i-1)*k+1}^{i*k} \mathbf{x}_j \text{ is odd} \right\} = Pr\left\{ \sum_{j=1}^{k} \mathbf{x}_j \text{ is odd} \right\} = \\
&= \sum_{\substack{j=0 \\ j \text{ is odd}}}^{k} \binom{k}{j} P_1^j (1-P_1)^{k-j} = \\
&= \frac{1}{2}\left[\sum_{j=0}^{k} \binom{k}{j} P_1^j (1-P_1)^{k-j} - \sum_{j=0}^{k} \binom{k}{j} (-P_1)^j (1-P_1)^{k-j} \right] = \\
&= \frac{1}{2}\Big[(P_1 + 1 - P_1)^k - (-P_1 + 1 - P_1)^k \Big] = \frac{1}{2}\Big[1 - (1-2P_1)^k \Big] \\
Pr\{\mathbf{y}_i = 0\} &= 1 - Pr\{\mathbf{y}_i = 1\} = \frac{1}{2}\Big[1 + (1-2P_1)^k \Big]
\end{aligned}$$

Finally we obtain

$$\Big|Pr\{\mathbf{y}_i = 0\} - Pr\{\mathbf{y}_i = 1\}\Big| = \left| \frac{1}{2}\Big[1 + (1-2P_1)^k \Big] - \frac{1}{2}\Big[1 - (1-2P_1)^k \Big] \right| = |1 - 2P_1|^k.$$

□

From this theorem, we can see that by using blockwise parity we can create random bitstream from a biased bitstream (image) even for small blocklengths. Hence, the stream of $LSB(\mathcal{B}_j)$ is almost random. From the analysis of LSB embedding (equation (1.4) and (1.5)), we can see that there will be no difference between message extracted from cover image and from stego image (both are random bitstreams). Hence, the power of parity saves us in the sence, that Wendy cannot distinguish between cover and stego images based on extracting the message.

Finally, we should note that in the case when the warden is sure that she has the stego image with embedded message, she can extract the encrypted message, but she cannot obtain the original message, because we used the public-key cipher and she does not have the correct key for decryption. We should stress that the goal of public-key steganography is to enable invisible communication, which, when it is combined with public-key cryptography, forms the whole communication scheme. However, while the scheme we just described is working, it has one disadvantage low capacity. By dividing the image into blocks, we are losing a lot from the original number of bits, we can communicate using this scheme. Further in this thesis, we show that we can construct better public-key steganography schemes without loss of capacity.

1.4.2 Steganography with non-shared side-information

Another case where we need adaptivity is the following one. From practical experiments of testing steganalytic algorithms on various image sources can be seen that the detection accuracy depends on the level of noise that is present in the image. For example, the accuracy is lower for scanned images than for images obtained from a digital camera. This idea suggests to embed more bits in textured areas than in smooth areas in the given cover image. This is a good idea which should decrease the detectability, because it masks the changes into noisy areas. We use a similar approach as in public-key steganography to develop the adaptive scheme which embeds message bits only into textured areas.

In a similar way, we divide the cover image into small blocks ($k \times k$ neighboring pixels) and use the block variance as a measure of adaptivity. The block variance $var(\mathcal{B}_j)$ (variance of pixels in a specific block) gives us information which we use for embedding in the following way. We define threshold T and embed one message bit into each block with $var(\mathcal{B}_j) > T$ and we skip all blocks with $var(\mathcal{B}_j) \leqslant T$. The threshold T should be shared between Alice and Bob or it should be communicated (suitably quantized and stored using a few bits). The embedding of one message bit is done similarly as in public-key steganography. There is one problem we should solve. When Bob wants to extract the message, he calculates variance of each block and extracts the message as $LSB(\mathcal{B}_j)$ from $var(\mathcal{B}_j) > T$. The problem occurs when we change the block variance by embedding the message bit such that the resulting block variance will be lower than the threshold T. Bob will not use this block for extracting the message and hence he will extract the wrong message. This problem can be solved by either changing another pixel in the block $\mathcal{B}_j$ such that this problem does not occur, or we have to skip this block with embedded message bit and re-embed the same bit again (similarly as with shrinkage in F5). This lowers the capacity, but is necessary for correct communication.

Chapter 2

Minimizing embedding impact

In the previous chapter, we described steganography and steganalysis as two complementary disciplines, where the goal of these disciplines is to introduce and detect invisible communication in some general media. We showed some basic algorithms how to perform this communication using digital images and, as an example, we described an algorithm for reliable detection of this communication. We used the prisoners' model for describing the communication scenario. In this chapter, we will use the prisoners' problem and study the security of information embedding. The aim of this chapter is to use a known model of steganographic security to introduce a general approach for improving the security of known steganographic schemes. This will be done by minimizing the impact of embedding a message into the cover image.

We start by giving some definitions which are intuitive and which will be used further in this text. When we were embedding message M using some staganographic scheme, we took the cover image represented using the symbol assignment function as a sequence $\mathbf{x} \in \mathbb{F}_q^n$ and we changed this sequence to the stego image represented as a sequence $\mathbf{y} \in \mathbb{F}_q^n$. Further, we assume that our message M can be represented as a sequence $\mathbf{m} \in \mathbb{F}_q^m$. In our previous examples, we used $q = 2$, but we can think of another values. We will discuss the extension later. In further text, we assume that the message is a random stream of elements from $\mathbb{F}_q$.

To measure the similarity between the cover and stego images, we use the *embedding distortion* defined as

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |\mathbf{x}_i - \mathbf{y}_i|. \tag{2.1}$$

For the special case where $q = 2$, $q = 3$, we are counting the number of modified elements. We can connect this definition with previous algorithms and calculate embedding distortion based on relative message length α. For the case of LSB embedding and ± 1 embedding, we obtain $d(\mathbf{x}, \mathbf{y}) = \frac{\alpha}{2} n = \frac{m}{2}$. The embedding distortion is tightly connected with another function called *embedding efficiency* defined as

$$e(\mathbf{x}, \mathbf{y}) = \frac{m}{d(\mathbf{x}, \mathbf{y})}. \tag{2.2}$$

This function determines the average number of symbols (bits, ternary symbols) embedded using only one change in cover image. For the case of the previously mentioned algorithms, we obtain $e(\mathbf{x}, \mathbf{y}) = 2$. This number seems to be constant, but later we will show that 2 is the lower bound and that we can achieve higher embedding efficiency. We will use d and e

to denote *average embedding distortion* and *average embedding efficiency*, where the average is calculated over all possible messages and cover images for a given steganographic scheme.

Before we introduce a formal model for steganographic security, we can intuitively define the *capacity of a steganographic scheme* as the number of bits that can be embedded using this scheme without introducing statistically detectable artifacts. This definition is based on current knowledge of steganalysis, where we can detect some artifacts and use them to brake the scheme. As an example, using the results from Sample Pairs analysis, we can say that the capacity of the LSB embedding scheme is very low.

Although LSB embedding is highly detectable even for small α, the ± 1 embedding is much less detectable. Intuitively, it is not hard to see that the security of a steganographic scheme depends on relative message length, because for small values of α we need a small number of changes to embed a message and hence introduce less statistically detectable artifacts. We can conclude this intuitive discussion by the statement, that we can increase the steganographic security by minimizing the number of changed elements in the stego image. By minimizing the number of changed elements, we increase the embedding efficiency and decrease distortion.

2.1 Cachin's model of steganographic security

To formally describe the steganographic security, we use the model introduced by Cachin [4]. This model describes steganography with passive warden in terms of hypothesis testing and introduces the information-theoretical view of security. To introduce this approach, we first mention some basics from hypothesis testing and after that describe the security of information embedding.

2.1.1 Review of Hypothesis Testing

Assume we have two probability distributions P_{A_0} and P_{A_1} defined over some set $\mathcal{A}$ and we obtained a measurement $a \in \mathcal{A}$. The hypothesis testing problem is the task of deciding whether the measurement a was obtained from the distribution P_{A_0} (hypothesis H_0) or P_{A_1} (hypothesis H_1). In this case of binary hypothesis testing problem, the decision rule is a binary partition function defined over the set $\mathcal{A}$ that assigns one of the two hypotheses to each measurement $a \in \mathcal{A}$. In this hypothesis testing problem, we can introduce 2 types of errors. We say that we are making *error of the first kind* (false positive), when we accept hypothesis H_1, when H_0 is actually true. Conversely, we say that we are making *error of the second kind* (false negative), when accepting H_0, while H_1 is true. We use *probability of false alarms*, denoted P_{FA}, defined as the probability of making the error of the first kind and P_{MD}, called *probability of missed detection*, to express the probability of making an error of the second kind.

The optimal solution for the binary hypothesis testing is given by the Neyman-Pearson theorem. This theorem states, that for every P_{FA} there exists a constant threshold T, so that the optimal decision function for rejecting hypothesis H_0 (and accepting H_1) is the *log-likelihood ratio* function

$$L(a) = \frac{P_{A_0}(a)}{P_{A_1}(b)} \leqslant T. \tag{2.3}$$

This decision function is optimal, because it minimizes the probability P_{MD} given fixed P_{FA}. The threshold T can be calculated from the constraint on P_{FA} and satisfies

$$Pr\{L(a) \leqslant T|H_0\} = P_{FA}. \tag{2.4}$$

The information, we obtain from hypothesis testing using two probability distributions P_{A_0} and P_{A_1}, is measured by *relative entropy* (Kullback-Leibler divergence) between these two distributions, defined as

$$D(P_{A_0} \parallel P_{A_1}) = \sum_{a \in \mathcal{A}} P_{A_0}(a) \log \frac{P_{A_0}(a)}{P_{A_1}(a)}. \tag{2.5}$$

Using relative entropy, we are measuring the "distance" between two probability distributions. Although the relative entropy does not fulfill the definition of the true distance measure, it can be useful to think of it as a distance. The useful property is that $D(P_{A_0} \parallel P_{A_1}) = 0$ if and only if $P_{A_0} = P_{A_1}$.

Another characteristic of the relative entropy is that deterministic processing cannot increase the value of relative entropy. It means that when we have a deterministic mapping $f : \mathcal{A} \to \mathcal{B}$ and transform $B_0 = f(A_0)$, $B_1 = f(A_1)$ then

$$D(P_{B_0} \parallel P_{B_1}) \leqslant D(P_{A_0} \parallel P_{A_1}). \tag{2.6}$$

2.1.2 Information-theoretical model of steganographic security

The warden's task in steganography with passive warden can be seen as a hypothesis testing problem. We define the set $\mathcal{A}$ as a set of all possible images and define two probability distributions P_C and P_S, where P_C is the distribution of cover images (natural images) and P_S is the distribution of stego images. Here, we assume that warden has the P_C distribution and from Kerchoff's principle we assume that warden knows the algorithm we are using for embedding and hence she knows P_S. The warden's objective is to perform binary hypothesis testing and accept hypothesis H_0 if the received image is a cover image (natural image), otherwise accept hypothesis H_1 the received image is suspicious (warden receives a stego image) and she should not deliver this image to Bob. The optimal approach is to use the Neyman-Pearson theorem, where we set the probability of false alarms P_{FA} to some small value.

It is easy to see that if the $P_C = P_S$, then the hypothesis testing problem reduces to pure random guessing, therefore the following definitions are plain enough. We say that the steganographic scheme is ϵ-*secure against passive adversaries* if

$$D(P_C \parallel P_S) \leqslant \epsilon.$$

We say that the scheme is *perfectly secure* if $\epsilon = 0$ ($P_C = P_S$).

Using the assumption that the steganographic scheme is ϵ-secure, we can derive the lower bound on the probability of missed detection P_{MD}. To do this, we use the fact that the optimal decision function for binary hypothesis problem is a deterministic mapping (2.3). Using this mapping (denote it f), we can transform the set $\mathcal{A}$ into a binary set $\{0, 1\}$, where the function f outputs 0, when it receives cover image and 1 when it receives stego image.

Using this mapping, we obtain the following probability distributions

$$P_{f(C)}(0) = 1 - P_{FA} \qquad P_{f(C)}(1) = P_{FA}$$
$$P_{f(S)}(0) = P_{MD} \qquad P_{f(S)}(1) = 1 - P_{MD}.$$

Now it is straightforward that

$$\epsilon > D(P_C \parallel P_S) \geqslant D(P_{f(C)} \parallel P_{f(S)}) = (1 - P_{FA}) \log \frac{1 - P_{FA}}{P_{MD}} + P_{FA} \log \frac{P_{FA}}{1 - P_{MD}} = \\ = D(P_{FA}, P_{MD}), \tag{2.7}$$

where $D(P_{FA}, P_{MD})$ is the *binary relative entropy*.

In practical application of steganalysis, we prefer to have small P_{FA} rather then small P_{MD}. This is because we rather miss some stego image then create false alarms. The reason is because when we claim that the received image contains some hidden message we use our resources to extract the message and therefore it is better to be sure that we are not making mistake in detection. Due to this reason, it is meaningful to derive the lower bound on P_{MD} for special case $P_{FA} = 0$. In this case, we have

$$\epsilon > -\log P_{MD} \quad \Rightarrow \quad P_{MD} > 2^{-\epsilon}, \tag{2.8}$$

where for $\epsilon \to 0$ we obtain useless detector because $P_{MD} \to 1$.

Although in practice we cannot obtain the distributions P_C and P_S, we can use this model to describe the security of steganographic scheme in a theoretical way. Based on the terms we used in this section, we can say that by minimizing the Kullback-Leibler divergence of cover and stego image distributions we are increasing the security of a whole scheme. This statement is more general then just minimizing the number of changes.

2.2 Improving embedding efficiency

Before we introduce a general approach for minimizing Kullback-Leibler divergence of cover and stego image distributions, we will focus on a simple approach for increasing embedding efficiency. We already mentioned the idea when we were describing the F5 embedding algorithm for JPEG images. This approach, introduced by Crandall in [6], was called Matrix Embedding and it enables us to embed more bits using fewer changes.

In section 1.2.2 we showed that for special case $q = 2$ when both Alice and Bob share the following binary matrix

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \tag{2.9}$$

then they can communicate 3 bits by making at most 1 change in 7-bit cover. To use this in real communication scheme, Alice and Bob pre-agree the relative message length α and both generate the matrix $\mathbf{H}$. The following communication will be done by dividing cover image and message into blocks and embedding each block separately.

The embedding of a 3-bit message $\mathbf{m}$ into a 7-bit cover $\mathbf{x}$ is done by calculating $\mathbf{h} = \mathbf{H}\mathbf{x} + \mathbf{m}$. If $\mathbf{h} = 0$, then we are done and $\mathbf{y} = \mathbf{x}$, because Bob will always extract the message as $\mathbf{m} = \mathbf{H}\mathbf{y}$. When $\mathbf{h} \neq 0$, we can find the vector $\mathbf{h}$ as a column (say i-th column) in matrix $\mathbf{H}$

p	α	e
1	1.000	2.000
2	0.667	2.667
3	0.429	3.429
4	0.267	4.267
5	0.161	5.161

p	α	e
6	0.093	6.093
7	0.055	7.055
8	0.031	8.031
9	0.018	9.018
p	$\frac{p}{2^p-1}$	$\frac{p}{1-2^{-p}}$

Table 2.1: Relative message length α and average embedding efficiency e for Matrix Embedding using binary $(2^p-1, 2^p-1-p)$ Hamming codes.

and hence it is sufficient to change i-th bit in cover $\mathbf{x}$ and output this vector as stego $\mathbf{y}$. By changing the i-th bit we add the corresponding column to $\mathbf{h}$ and hence $\mathbf{h} = 0$ and Bob will extract the correct message. We can always find the non-zero vector $\mathbf{h}$ as a column in $\mathbf{H}$, because we constructed this matrix so that it contains all possible non-zero vectors of length 3.

By using this simple trick, we can achieve higher embedding efficiency then by using normal embedding, where we visit only those bits in cover $\mathbf{x}$ that should contain our message. The average embedding efficiency when using this approach is $e = \frac{3}{1-2^{-3}} = 3.429$, because we embed 3 bits and make distortion 1 with probability $1-2^{-3}$ and distortion 0 with probability 2^{-3} ($\mathbf{h}$ matches our message $\mathbf{m}$).

The matrix $\mathbf{H}$ from (2.9) is only a special case and we can construct larger or smaller matrices containing p rows and all non-zero vectors of length p as its columns. Using this approach, we obtain a set of matrices that can be used by communicating messages with different relative length α. In Table 2.1, we show the relative message length α and the average embedding efficiency e that we can obtain for different values of the parameter p. We can use this approach even for $\alpha = 1$, but we will not obtain any gain. The purpose of this idea is in creating fewer changes (smaller distortion) or we can embed larger message making the same distortion. For example, when $p = 4$ we are embedding a roughly 25% message and making the same distortion as embedding 12.5% message without using Matrix Embedding.

Matrix Embedding as was introduced by Crandall is a special case of a general approach which we will study further in this chapter. This approach is based on coding theory, where we use so-called syndrome codes for communicating messages. Matrix $\mathbf{H}$ in (2.9) is an example of a parity check matrix of linear $(7, 4)$ Hamming code. Next, we give some basic definitions and results from coding and information theory that we will need.

2.2.1 Basics of coding and information theory

Definition 2.1 (q-ary linear (n, k) code) *Linear subspace $\mathcal{C} \subset \mathbb{F}_q^n$, defined as $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_q^n | \mathbf{Hc} = 0\}$, where $\mathbf{H} \in \mathbb{F}_q^{n-k \times n}$ and all operations are in $GF(q)$ is called* q-ary linear (n, k) code. *The matrix $\mathbf{H}$ is called a* parity check matrix *and the vector $\mathbf{c}$ is called* codeword. *We say that the code $\mathcal{C}$ has dimension k (equal to the dimension of the linear subspace) and codimension $n-k$. Matrix $\mathbf{H}$ contains all basis vectors from the null space as its rows. The matrix formed from all basis vectors is called a generator matrix $\mathbf{G} \in \mathbb{F}_q^{n \times k}$ and we can obtain the code as $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_q^n | \mathbf{w} \in \mathbb{F}_q^k, \mathbf{c} = \mathbf{Gw}\}$. We say that the code has rate $R = \frac{k}{n}$.*

Definition 2.2 (Coset, syndrome) *Let $\mathcal{C}$ be q-ary linear (n,k) code with generator matrix $\mathbf{G}$ and parity check matrix $\mathbf{H}$. For any $\mathbf{x} \in \mathbb{F}_q^n$ we define the syndrome of vector $\mathbf{x}$ as $\mathbf{m} = \mathbf{H}\mathbf{x} \in \mathbb{F}_q^{n-k}$. For any syndrome vector, we define the* coset *associated to syndrome $\mathbf{m}$ as $\mathcal{C}(\mathbf{m}) = \{\mathbf{x} \in \mathbb{F}_q^n | \mathbf{H}\mathbf{x} = \mathbf{m}\}$. For $\mathcal{C}(0)$ we obtain $\mathcal{C}(0) = \mathcal{C}$.*

From elementary algebra, we obtain that cosets associated with different syndromes are disjoint, hence we have 2^{n-k} disjoint sets. We can express each coset $\mathcal{C}(\mathbf{m})$ as $\mathcal{C}(\mathbf{m}) = \mathbf{x} + \mathcal{C}$, where $\mathbf{x} \in \mathcal{C}(\mathbf{m})$ is an arbitrary vector from the coset.

Here, we emphasize the main problems that are solved using coding theory. The first and major problem is reliable communication over a noisy channel. This part is sometimes called "channel coding" and was introduced by Shannon when he gave mathematical description of transmitting information over noisy channels. He introduced basic models for noisy channels that are used in todays communication. The goal of coding theory is to design codes that enable reliable communication, which means that we are able to "repair" the initial sequence corrupted by the channel. In this scenario, Shannon introduced the capacity of each channel as a maximal rate where the reliable communication is still possible. For practical applications, it was not clear how to achieve this capacity and reliably transmit the maximum information over the channel. This part of coding theory is well developed, but does not cover our problem in steganography, therefore we will not mention here.

The problem which is of our interest, is the problem of lossy data compression. This part of coding theory called "source coding" tries to compress the input sequence by encoding it using a smaller number of elements. The goal here is to introduce as small distortion as possible while performing large compression. It is easy to see that we cannot introduce small distortion, while using large compression. The bound for minimum possible distortion we can achieve by lossy data compression was given by Shannon and is known as the "rate-distortion bound". For the case when we want to compress binary sequences we have the following bound.

Theorem 2.1 (Rate-distortion bound) *For binary variable $x \in \{0,1\}$, we define probability distribution Bernoulli(p) as $Pr\{x = 1\} = p$, and $Pr\{x = 0\} = 1 - p$. Let $R = \frac{k}{n} \in [0,1]$ be the compression ratio for compressing i.i.d. realizations $\mathbf{y} \in \{0,1\}^n$ of Bernoulli(1/2) into a shorter sequence $\mathbf{x} \in \{0,1\}^k$. Denoting $\hat{\mathbf{y}} \in \{0,1\}^n$ the decompressed sequence and $d_H(\mathbf{y}, \hat{\mathbf{y}}) = |\mathbf{y} - \hat{\mathbf{y}}|$ the Hamming distance of $\mathbf{y}$ and $\hat{\mathbf{y}}$, then every pair of compression $(f : \{0,1\}^n \to \{0,1\}^k)$ and decompression $(f^{-1} : \{0,1\}^k \to \{0,1\}^n)$ mapping fulfill the following rate-distortion bound*

$$R \geqslant 1 - H(D), \tag{2.10}$$

where D is the average distortion per bit defined as

$$D = \mathbb{E}_{\mathbf{y} \in \{0,1\}^n} \left[\frac{1}{n} d_H \left(\mathbf{y}, f^{-1}(f(\mathbf{y})) \right) \right], \tag{2.11}$$

and $H(x) = -x \log_2 x - (1-x) \log_2 (1-x)$ is the binary entropy function.

In Figure 2.1, we have the graph showing the rate-distortion bound. It shows that according to the theorem, all compression schemes have to operate above the rate distortion bound and hence it is not possible to construct any mappings that would perform better than this theoretical bound.

To perform the compression described in the theorem, we can think of the most trivial algorithm which we can always use. The compression mapping f outputs first k bits from

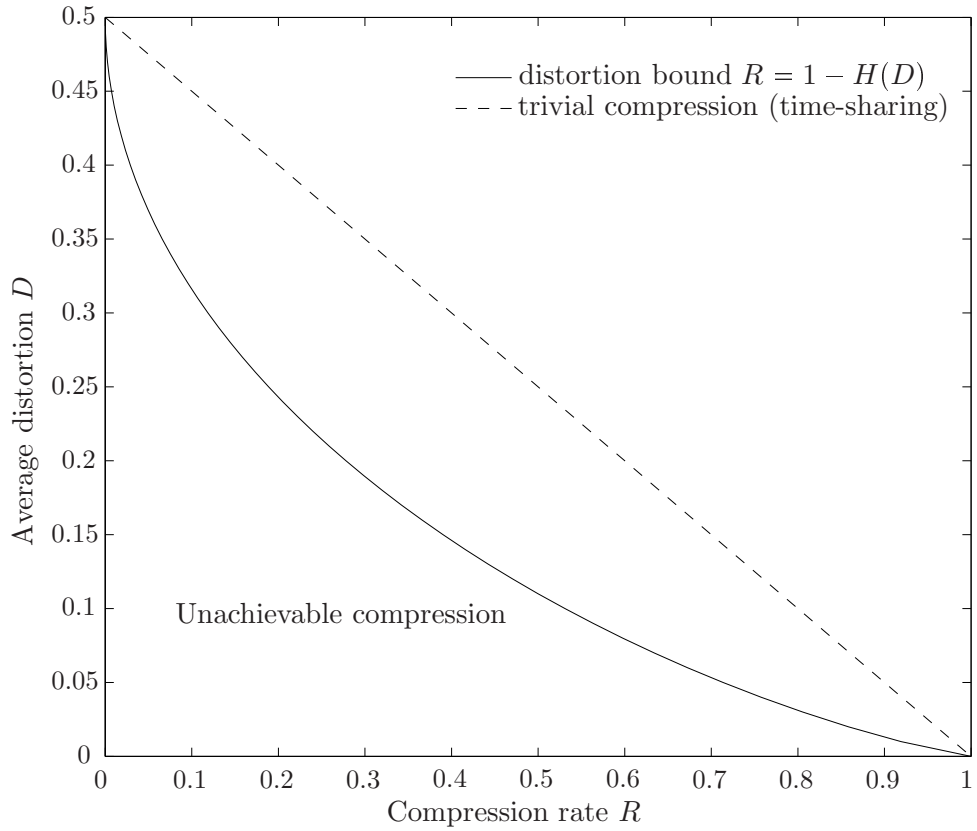


Figure 2.1: Rate-distortion bound for lossy compression of Bernoulli(1/2) source.

sequence $\mathbf{y}$ and the decompression is done by padding the end of the sequence with arbitrary values. When we assume the compression ratio $R = \frac{k}{n}$, then the average distortion per bit is $D = \frac{n-k}{2n}$. This algorithm is called *time-sharing*.

2.3 General approach for minimizing embedding impact

In Section 2.1, we saw that the security of a steganographic scheme can be measured as the Kullback-Leibler divergence of cover and stego image distributions. This result is true, but in practice we never have these distributions to be able to find the best steganographic scheme. In practice, we can approximate these distributions and try to design some scheme based on this approximation. This approach was done several times in steganography, but it did not take a long time to find some leak and come with some steganalytic algorithm to detect this (previously undetectable) scheme.

From this reason, we try to unify the approach and define the *embedding impact*, which we will further minimize. For this approach, we assume that the impact of embedding change in i-th pixel can be measured by a non-negative number $\varrho_i \in [0, 1]$ and hence we define the (total) embedding impact as

$$D(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_D = \sum_{i=1}^{n} \varrho_i |\mathbf{x}_i - \mathbf{y}_i|. \tag{2.12}$$

This detectability measure should be designed to correlate with the statistical detectability of embedding changes. In practice, ϱ_i is usually proposed using heuristic principles. For example, for a non-negative parameter ν and weight factors $\omega_i \geqslant 0$

$$\varrho_i = \omega_i |\mathbf{g}_i - \mathbf{g'}_i|^\nu, \tag{2.13}$$

where $\mathbf{g}_i$, and $\mathbf{g'}_i$ are colors of the i-th pixel in the cover and stego image, respectively. If the embedding change is probabilistic, we understand (2.13) as the expected value.

As an example, we have $\omega_i = 1$, $\forall i = 1, \ldots, n$. In this special case, we will minimize the number of changes as in Matrix Embedding. Furthermore, we can model the so-called wet paper coding [11] by setting $\omega_i = 1$ for $i \in Dry$ and $\omega_i = 0$ otherwise, for some index set $Dry \subset \{1, \ldots, n\}$. In general, the weighting factors may depend on the local texture to reflect the fact that embedding changes in textured (or noisy) areas are more difficult to detect than changes in smooth segments of the cover image.

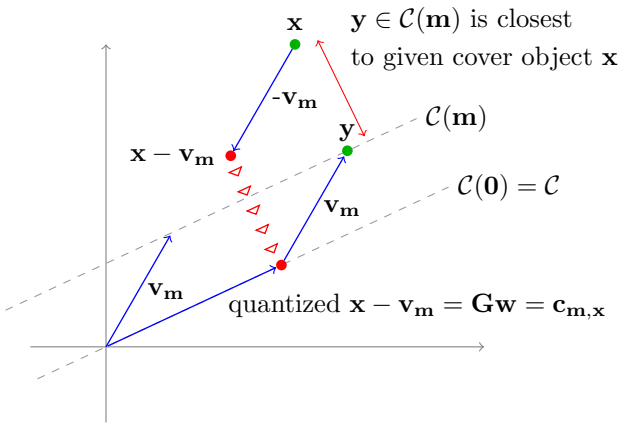
The impact ϱ_i may also be determined from some side-information available to the sender as in Perturbed Quantization steganography (PQ) [10]. For example, let us assume that the cover is a TIFF image sampled at 16 bits per channel. The sender wishes to embed a message while decreasing the color depth to a true-color 8-bit per channel image while minimizing the combined quantization and embedding distortion. Let z_i be the 16-bit color value and let $Q = 2^8$ be the quantization step for the color depth reduction. The quantization error is $e_i = Q|z_i/Q - [z_i/Q]|$, $0 \leqslant e_i \leqslant Q/2$, and the error when rounding z_i to the opposite direction is $Q - e_i$ leading to embedding distortion as the difference between both errors $\varrho_i = Q - 2e_i$. In PQ, the coefficients are selected for which $e_i \approx Q/2$ because for such coefficients, the embedding distortion is the smallest. Also note that in this case, since the quantization error is approximately uniform on $[-Q/2, Q/2]$, when sorting ϱ_i by their values the resulting profile will be well modeled with a straight line.

We point out that the (2.12) implicitly assumes that the embedding impact is additive because it is defined as a sum of detectability measures at individual pixels. In general, however, the embedding modifications could be interacting among themselves, reflecting the fact that making two changes to adjacent pixels might be more detectable than making the same changes to two pixels far apart from each other. A detectability measure that takes interaction among pixels into account would not be additive. If the density of embedding changes is low, however, the additivity assumption is plausible, because the distances between modified pixels will generally be large and the embedding changes will not interfere much.

2.3.1 Problem formulation

The central problem investigated in this thesis is construction of near-optimal steganographic schemes by minimizing the embedding impact. Generally, we can define the framework for an arbitrary finite field $\mathbb{F}_q$ used for cover and stego image representation, however we will further study the binary case. According to work of Fridrich [8], in steganography we should be interested in the binary and ternary cases, because $q \geqslant 4$ does not give us any further gain. Further in this section, we will use the term *element* referring either to bit (when $q = 2$) or trit (trinary digit) when $q = 3$. All operations are done in $GF(q)$.

Let us assume that the receiver knows the relative message length $\alpha = \frac{m}{n}$ and thus the number of secret message elements m. Let $\mathcal{C}$ be a q-ary linear $[n, n-m]$ code with an $n \times (n-m)$ generator matrix $\mathbf{G}$ and an $m \times n$ parity check matrix $\mathbf{H}$. Both matrices are shared between the sender and the recipient. Let $\mathcal{C}(\mathbf{m}) = \{\mathbf{u} \in \mathbb{F}_q^n | \mathbf{Hu} = \mathbf{m}\}$ be the coset



Embedding process:

1. shift cover $\mathbf{x}$ using arbitrary coset member $\mathbf{v_m}$
2. quantize $\mathbf{x}-\mathbf{v_m}$ into the code $\mathcal{C}$ (find nearest codeword to $\mathbf{x}-\mathbf{v_m}$)
3. shift nearest codeword back into coset $\mathcal{C}(\mathbf{m})$
4. output $\mathbf{y}$ as a shifted vector

Figure 2.2: Geometrical interpretation of embedding process.

corresponding to syndrome $\mathbf{m} \in \mathbb{F}_q^m$ ($\mathbf{m}$ is the secret message). The following embedding scheme communicates m elements in an n-element cover vector $\mathbf{x}$

$$\begin{aligned} \mathbf{y} = Emb(\mathbf{x},\mathbf{m}) &\triangleq \arg\min_{\mathbf{u}\in\mathcal{C}(\mathbf{m})} \|\mathbf{x}-\mathbf{u}\|_D \\ Ext(\mathbf{y}) &= \mathbf{H}\mathbf{y} = \mathbf{m}. \end{aligned} \tag{2.14}$$

Here, $\mathbf{y}$ are the elements assigned to the stego image. In other words, in an attempt to minimize the embedding impact, the sender selects such a member $\mathbf{y}$ of the coset $\mathcal{C}(\mathbf{m})$ that is closest to $\mathbf{x}$ (closest in metric $\|.\|_D$).

Let $\mathbf{v_m} \in \mathcal{C}(\mathbf{m})$ arbitrary. Then,

$$\min_{\mathbf{u}\in\mathcal{C}(\mathbf{m})} \|\mathbf{x}-\mathbf{u}\|_D = \min_{\mathbf{c}\in\mathcal{C}} \|\mathbf{x}-(\mathbf{v_m}+\mathbf{c})\|_D = \min_{\mathbf{w}\in\mathbb{F}_q^{n-m}} \|\mathbf{x}-\mathbf{v_m}-\mathbf{G}\mathbf{w}\|_D\,. \tag{2.15}$$

From (2.15), we see that embedding is a q-ary quantization problem. See Figure 2.2 for a geometrical interpretation of embedding algorithm. The sender needs to find $\mathbf{w} \in \mathbb{F}_q^{n-m}$ such that $\mathbf{G}\mathbf{w}$ is closest to $\mathbf{x}-\mathbf{v_m}$. Alternatively, we can say that the sender is compressing the *source sequence* $\mathbf{s} = \mathbf{x}-\mathbf{v_m}$ to $n-m$ *information* elements $\mathbf{w}$ so that the reconstructed vector $\mathbf{G}\mathbf{w}$ is as close to the source sequence as possible. Let us denote the closest codeword $\mathbf{G}\mathbf{w}$ as $\mathbf{c_{m,x}}$.

Assuming there exists an efficient algorithm for finding both $\mathbf{v_m}$ and $\mathbf{c_{m,x}}$, the stego object $\mathbf{y}$ is

$$\mathbf{y} = \mathbf{x} + \mathbf{c_{m,x}} - \mathbf{s} = \mathbf{c_{m,x}} + \mathbf{v_m}. \tag{2.16}$$

Four things need to be supplied to make the description of this embedding scheme complete. We need to describe the process by which we generate the code, the algorithm for finding $\mathbf{v_m}$, and the algorithm for q-ary quantization. We also need to explain why the distortion of this embedding scheme is near-optimal. The most difficult step in the proposed scheme is the quantization process.

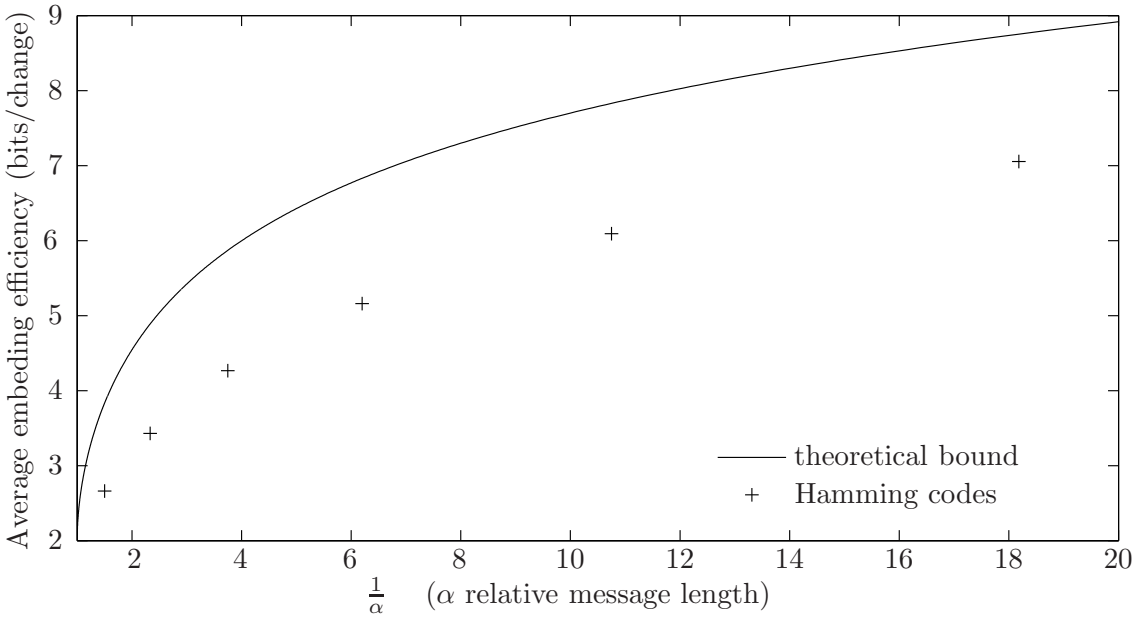


Figure 2.3: Upper bound on embedding efficiency and performance of Hamming codes.

2.4 Binary case of proposed framework

As a special case of proposed framework, we will further study the binary case. All results and derivations from the previous section hold and all operations are in $\mathbb{F}_2 = GF(2)$. In this section, we derive upper bound for achievable embedding efficiency and generally the lower bound for minimal embedding impact of arbitrary additive distortion measure. These bounds represent the smallest embedding impact we can ever achieve.

The simplest case of minimizing the embedding impact is the case for uniform profile ($\varrho_i = 1$), where we want to minimize the number of changes made by embedding. By minimizing the number of changes, we are increasing the embedding efficiency hence we will further work with this value. From equation (2.15), we obtain that the minimization problem is equivalent to binary quantization, therefore we can obtain the upper bound on embedding efficiency from rate distortion bound as described in Section 2.2.1. Here, we should note that a binary linear code $\mathcal{C}$ used for embedding has rate $R = \frac{n-m}{n} = 1 - \alpha$, hence we can write the rate-distortion bound using the average embedding distortion d (2.1) as

$$\alpha = 1 - R \leqslant H(d/n). \tag{2.17}$$

Finally from the definition of the average embedding efficiency we obtain the upper bound

$$e \leqslant \frac{\alpha}{H^{-1}(\alpha)}. \tag{2.18}$$

Figure 2.3 contains the graph of currently described upper bound with embedding efficiency of known Hamming codes we described in Section 2.2. On x axis, we plot the inverse of the relative message length α. When we do not use any kind of Matrix Embedding, we obtain the average embedding efficiency $e = 2$. From the graph we can see, that by using simple Hamming codes, we significantly increase the embedding efficiency, however Hamming codes do not saturate the bound, hence we can come up with some better linear codes. Another

problem is that for practical steganography we only have a few Hamming codes for special relative message lengths that we can use. In practice, we can solve this problem by using more than one Hamming code for embedding, for example embed half of our message using code with $p = 6$ and the rest of the message with code with $p = 7$. This approach is known as *direct sum of codes* and it allows us to obtain codes for other relative message lengths, however using this approach we do not obtain high embedding efficiency.

Next, we move to the more interesting case and study the minimization of embedding impact for a general profile ϱ_i. In this case, we will calculate the lower bound on the minimal embedding impact for a general but fixed profile ϱ_i. Further, we will plot this bound with respect to the relative message length $\alpha = \frac{m}{n}$.

Here, we do the derivation for the more general case when the embedding impact is an arbitrary (i.e., not necessarily additive) function of the detectability measure ϱ. For $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$, we define the modification pattern $\mathbf{z} \in \{0,1\}^n$ as $\mathbf{z}_i = \delta(\mathbf{x}_i, \mathbf{y}_i)$, where $\delta(a,b) = 1$ when $a = b$ and $\delta(a,b) = 0$, otherwise. Furthermore, we define $D(\mathbf{z}) = D(\mathbf{x}, \mathbf{y})$ as the embedding impact of making embedding changes at pixels with $\mathbf{z}_i = 1$. Let us assume that the recipient also knows the cover $\mathbf{x}$. By the Gelfand-Pinsker theorem [14], the conclusions reached here do not depend on this assumption. The sender then basically communicates the modification pattern $\mathbf{z}$. Assuming the sender selects each pattern $\mathbf{z}$ with probability $p(\mathbf{z})$, the amount of information that can be communicated is the entropy of $p(\mathbf{z})$

$$H(p) = -\sum_{\mathbf{z}} p(\mathbf{z}) \log_2 p(\mathbf{z}).$$

Our problem is now reduced to finding the probability distribution $p(\mathbf{z})$ on the space of all possible flipping patterns $\mathbf{z}$ that minimizes the expected value of the embedding impact

$$\sum_{\mathbf{z}} D(\mathbf{z}) p(\mathbf{z})$$

subject to the constraints

$$H(p) = \sum_{\mathbf{z}} p(\mathbf{z}) \log_2 p(\mathbf{z}) = m \qquad\qquad \sum_{\mathbf{z}} p(\mathbf{z}) = 1,$$

This problem can be solved using Lagrange multipliers. Let

$$F\big(p(\mathbf{z})\big) = \sum_{\mathbf{z}} p(\mathbf{z}) D(\mathbf{z}) + \mu_1 \left(m - \sum_{\mathbf{z}} p(\mathbf{z}) \log_2 p(\mathbf{z}) \right) + \mu_2 \left(\sum_{\mathbf{z}} p(\mathbf{z}) - 1 \right).$$

Then,

$$\frac{\partial F}{\partial p(\mathbf{z})} = D(\mathbf{z}) - \mu_1 (\log_2 p(\mathbf{z}) + 1/\ln(2)) + \mu_2 = 0$$

if and only if $p(\mathbf{z}) = A e^{-\zeta D(\mathbf{z})}$, where $A^{-1} = \sum_{\mathbf{z}} e^{-\zeta D(\mathbf{z})}$ and ζ is determined from

$$-\sum_{\mathbf{z}} p(\mathbf{z}) \log_2 p(\mathbf{z}) = m.$$

Thus, the probabilities $p(\mathbf{z})$ follow an exponential distribution with respect to the embedding impact $D(\mathbf{z})$.

If the embedding impact of the pattern $\mathbf{z}$ is an additive function of "singleton" patterns (patterns for which only one pixel is modified), then $D(\mathbf{z}) = \mathbf{z}_1 \varrho_1 + \cdots + \mathbf{z}_n \varrho_n$ and $p(\mathbf{z})$

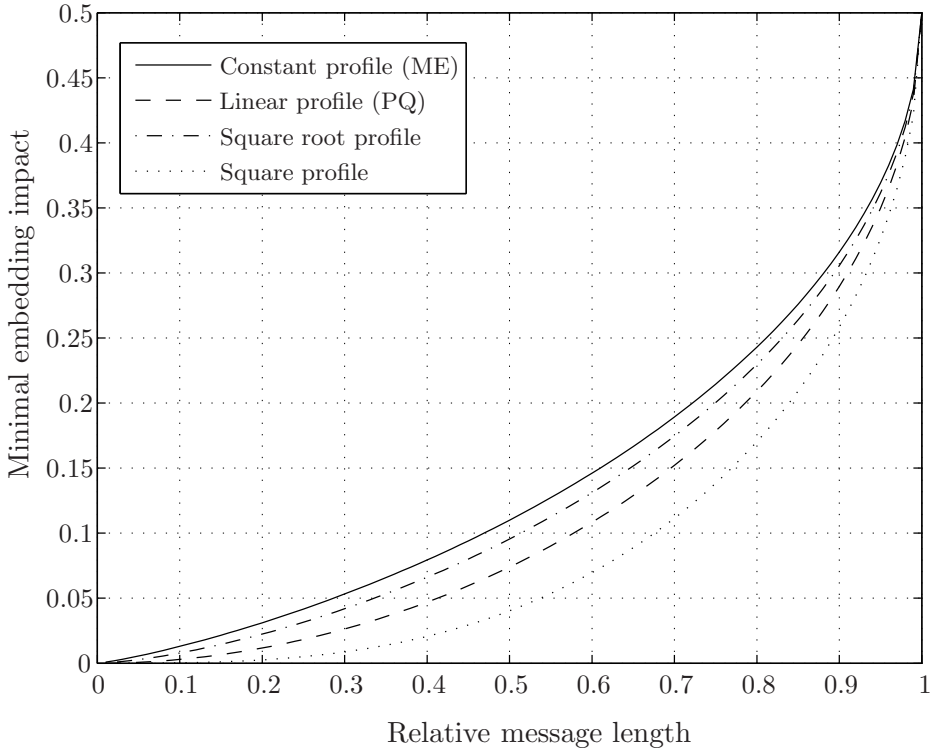


Figure 2.4: Minimal embedding impact vs. relative message length for four detectability profiles ϱ.

accepts the form

$$p(\mathbf{z}) = Ae^{-\zeta\sum_{i=1}^{n}\mathbf{z}_i\varrho_i} = A\prod_{i=1}^{n}e^{-\zeta\mathbf{z}_i\varrho_i},$$
$$A^{-1} = \sum_{\mathbf{z}}\prod_{i=1}^{n}e^{-\zeta\mathbf{z}_i\varrho_i} = \prod_{i=1}^{n}\left(1+e^{-\zeta\varrho_i}\right),$$

which further implies

$$p(\mathbf{z}) = \prod_{i=1}^{n}p_i(\mathbf{z}_i),$$

where $p_i(1)$ and $p_i(0)$ are the probabilities that the i-th pixel is (is not) modified during embedding

$$p_i(0) = \frac{1}{1+e^{-\zeta\varrho_i}} \qquad p_i(1) = \frac{e^{-\zeta\varrho_i}}{1+e^{-\zeta\varrho_i}}. \tag{2.19}$$

Of course, this further implies that the joint probability distribution $p(\mathbf{z})$ can be factorized and thus we only need to know the marginal probabilities p_i that the i-th pixel is modified.

It also enables us to write for the entropy

$$H(p) = \sum_{i=1}^{n} H(p_i),$$

where in the sum the function H applied to a scalar is the binary entropy function.

Note that when $\varrho_i = 1, \forall i$, we obtain

$$m = \sum_{i=1}^{n} H(p_i) = \sum_{i=1}^{n} H\left(\frac{e^{-\zeta}}{1+e^{-\zeta}}\right) \tag{2.20}$$

$$\mathbb{E}\left(\sum_{i=1}^{n} p_i \rho_i\right) = \frac{n e^{-\zeta}}{1+e^{-\zeta}}.$$

Thus, in agreement with the result we derived above (and also in [1]) we obtain the following relationship between the embedding impact per pixel d/n and the relative message length $\alpha = \frac{m}{n}$

$$\frac{d}{n} = H^{-1}\left(\frac{m}{n}\right).$$

Let us sort ϱ_i from the smallest to the largest and normalize so that $\sum_i \varrho_i = 1$. Let ϱ be a Riemann-integrable non-decreasing function on $[0,1]$ such that $\varrho(i/n) = \varrho_i$. Then for $n \to \infty$, the average distortion per element

$$d = \frac{1}{n}\sum_{i=1}^{n} p_i \varrho_i \to \int_0^1 p(x)\varrho(x)dx,$$

where $p(x) = \frac{e^{-\zeta\varrho(x)}}{1+e^{-\zeta\varrho(x)}}$. By the same token,

$$\alpha = \frac{m}{n} = \frac{1}{n}\sum_{i=1}^{n} H(p_i) \to \int_0^1 H(p(x))dx.$$

By direct calculation

$$\begin{aligned}\ln 2 \times \int_0^1 H(p(x))dx &= \zeta \int_0^1 \frac{\varrho(x)e^{-\zeta\varrho(x)}}{1+e^{-\zeta\varrho(x)}}dx + \int_0^1 \ln(1+e^{-\zeta\varrho(x)})dx = \\ &= \zeta \int_0^1 \frac{(\varrho(x) + x\varrho'(x))e^{-\zeta\varrho(x)}}{1+e^{-\zeta\varrho(x)}}dx + \ln(1+e^{-\zeta\varrho(1)}).\end{aligned}$$

The second equality is obtained by integrating the second integral by parts. Thus, we can obtain the embedding capacity-distortion relationship in a parametric form

$$\begin{aligned}d(\zeta) &= G_\varrho(\zeta) \\ \alpha(\zeta) &= \frac{1}{\ln 2}\left(\zeta F_\varrho(\zeta) + \ln(1+e^{-\zeta\varrho(1)})\right),\end{aligned}$$

where ζ is a non-negative parameter and

$$G_\varrho(\zeta) = \int_0^1 \frac{\varrho(x)e^{-\zeta\varrho(x)}}{1+e^{-\zeta\varrho(x)}}dx \qquad F_\varrho(\zeta) = \int_0^1 \frac{(\varrho(x)+x\varrho'(x))e^{-\zeta\varrho(x)}}{1+e^{-\zeta\varrho(x)}}dx.$$

Chapter 3

Bias Propagation, an algorithm for binary quantization

In this chapter, we will introduce the first component we need for constructing near-optimal steganographic schemes, the binary quantization algorithm. In general we are interested in *weighted binary quantization problem*. For some linear code $\mathcal{C}$, weighted norm $||\cdot||_D$ and given source sequence $\mathbf{s}$, find

$$\min_{\mathbf{c}\in\mathcal{C}} ||\mathbf{c}-\mathbf{s}||_D.$$

It turns out that the simpler problem defined using the Hamming distance is NP-hard. We are interested in some good and low complexity approximation of this problem. The main result and contribution of this thesis is presented in this chapter and contains a new algorithm that meets the desired requirements. This algorithm, called *Bias Propagation* (BiP), uses sparse linear codes and achieves near-optimal distortion. We provide the description along with theoretical analysis of this algorithm.

This chapter is structured as follows. We introduce the problem and review some recent work on binary quantization in Section 1. Section 2 is devoted to the description of a graphical model we will use for solving the binary quantization problem. Sections 3 and 4 contain the motivation and intuitive derivation of Bias Propagation algorithm. Here, we use an algebraic approach of solving equations to create an iterative algorithm. BiP algorithm can be easily generalized to solve the weighted quantization problem. This is discussed in Section 5. In Section 6, we present another view on this algorithm through marginalization over some probability distribution. This approach gives us the formal derivation and, in Section 7, it allows us to study the convergence of the BiP algorithm. Finally, in Section 8 and 9 we describe an algorithm for binary quantization introduced by Wainwright et al. [24] and discuss the connections between BiP and their work.

3.1 Introduction to binary quantization

The binary quantization problem is a problem of compressing a random n bit binary vector $\mathbf{s}$ drawn from Bernoulli($\frac{1}{2}$) distribution into $n-m$ binary vector by some deterministic mapping. Our goal is to minimize the average Hamming distortion measured as $D = \mathbb{E}\big[d_H(\mathbf{s},\hat{\mathbf{s}})\big] = \mathbb{E}\big[\frac{1}{n}\sum_{i=1}^{n}|\mathbf{s}_i-\hat{\mathbf{s}}_i|\big]$, where $\hat{\mathbf{s}}$ is the reconstructed vector. Hereafter, we will call the vector $\mathbf{s}$ the *source sequence*. This lossy compression is done with rate $R=\frac{n-m}{n}$.

The first assumption we make to solve the binary quantization problem is that we use linearity as tool for constructing the decompression mapping. We generate the reconstructed sequence as $\hat{\mathbf{s}} = \mathbf{G}\mathbf{w}$ for some generator matrix $\mathbf{G} \in \{0,1\}^{n\times(n-m)}$, where all operations are in $GF(2)$. This assumption helps us in the decompression, but it does not give any constrain in the ability to achieve the rate-distortion bound (see Section 2.2.1). It was showed [13] that the rate-distortion bound is saturated by almost all linear codes when $n \to \infty$. This is an encouraging result, but the key problem is in existence of an efficient algorithm for large n. From the complexity view, it is easy to solve the system of linear equations in $GF(2)$, but the problem of minimizing the number of unsatisfied equations in an overconstrained problem is a known NP-hard problem called MAX-XOR-SAT.

Due to the computational complexity, we accept another assumption. We will not use arbitrary linear codes, however we will use linear codes defined by sparse generator matrices - so called Low Density Generator Matrix codes (LDGM). By our definition code is sparse when the number of ones in a $\mathbf{G}$ matrix is $\mathcal{O}(n)$. This is a reasonable assumption, because every algorithm with linear complexity in the number of edges will stay linear in code length n. From the work of Martinian and Wainwright [19], we can still achieve the rate-distortion bound using LDGM codes.

3.1.1 Review of recent work and algorithms

Before turning our attention to the Bias Propagation algorithm, we review some recent results which are important for solving the MAX-XOR-SAT problem.

In the last decade, there was a rapidly growing interest in using statistical mechanics in computer science. This interconnection brought many interesting results. The key result is the ability to analyse NP-hard problems using statistical mechanics and design new algorithms based on the knowledge from this analysis. The best known and analysed problem is the problem of boolean formula satisfiability (SAT). This classical NP-hard problem was traditionaly solved using algorithms based on local search (e.g., simulated annealing). It was known that these simple algorithms work for some "easy" instances of the SAT problem, however there were satisfiable instances of this problem that were hard to solve. Using methods from statistical physics, it was possible to analyse the structure of the solution space and it was proved that solutions in the satisfiable instance form clusters (solutions are close to each other in a Hamming distance). For easy instances there is one large cluster of solutions which we can easily walk through using small local changes. As the problems becomes "harder" to solve, the number of clusters increases, while the size (number of solutions forming a cluster) is decreasing. The process looks like a complexity phase transition, because the size of a cluster decreases exponentialy and this forms a problem for any local-search algorithm.

The analysis of clustering in the solution space (in a SAT problem) leads to the design of a new algorithm called Survey Propagation (SP) [3]. This algorithm exploits the clustering phenomenon and works as a 'cluster finder' when looking for satisfiable solution. When the correct cluster of solutions is found, we can enumerate and select a satisfiable solution using some local-search algorithm.

The SP algorithm is an example of so called *message-passing algorithm* which takes advantage of the underlying graphical representation of the problem. There are many other examples of algorithms which represent the problem as a (bipartite) graph and solve the problem by sending messages (real numbers or vectors) along each edge in the graph. This is usually an iterative process where the nodes in the graph transform incoming messages and forward new

message back along each adjacent edge. This idea usually leads to very efficient algorithms with linear or log-linear complexity in the number of edges.

The current work done by other research groups to solve the binary quantization problem is not sufficient to say that this problem is completely solved. Here, we give a description of two approaches that represent the current state of the art.

The first idea for solving the binary quantization problem is based on the work of Ciliberti, Mezard, and Zecchina [5]. Their work is inspired by the successful use of linearity in GF(2) in channel coding, where the Low Density Parity Check matrix codes (LDPC) were used. In fact, the binary quantization problem is dual to this problem, therefore they propose to use the LDGM codes (duals to LDPC codes) for solving the binary quantization. They showed that using a sparse system of linear equations, where each equation contains exactly k variables, it is possible to achieve the rate distortion bound for increasing parameter k. The gap between the theoretical performance of this system and the rate-distortion bound decreased exponentialy k. Therefore, $k < 10$ should be enough for our steganographic scheme construction. The problem still lays in an implementation of the practical algorithm. Because of this constrain, they had to depart from the linearity in each equation and they proposed a compression scheme based on a set of non-linear equations. They showed that the set of non-linear equations has similar theoretical properties and that the construction of a practical algorithm is possible using the Survey Propagation algorithm. According to the paper, the compression speed was on the order of hours for $n = 1000$ ($k = 6$), which seems impractical for our steganographic applications.

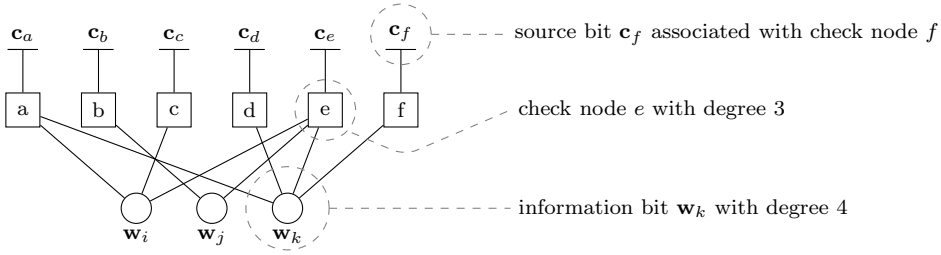
The second approach is based on the work of Wainwright and Maneva [24]. They proposed a compression scheme based on irregular LDGM codes (set of linear equations over $GF(2)$ with varying number of variables used in each equation). They used their previous analysis of Survey Propagation algorithm for SAT problem by Maneva, Wainwright, and Mossel [18] and derived the equations for the message-passing algorithm for binary quantization. They showed that the compression scheme based on this algorithm is able to produce results with near-optimal distortion. According to our analysis [9], we achieved speed of approx. 1000 bit/s, while obtaining the distortion very close to the bound. Although this approach gives us promising results, there are still many issues that need to be solved for practical applications. Due to the complexity of message update equations and the way how the whole scheme was derived, we do not see the solution in a straightforward manner. We will describe this algorithm later in this chapter in more detail, to compare it with Bias Propagation. For later reference, we will call this algorithm 'SP based quantizer'.

To derive the Bias Propagation algorithm, we will start with the underlying graphical representation of the MAX-XOR-SAT problem. This graphical representation will be used to describe the message-passing algorithm further in this chapter. In our derivation, we still keep the assumptions made earlier, namely the linearity of the reconstruction mapping and the sparsity of the generator matrix $\mathbf{G}$.

3.2 Graph representation of a code

The MAX-XOR-SAT problem consists of solving the following minimization problem:

$$\min_{\mathbf{c}\in\mathcal{C}} d_H(\mathbf{s},\mathbf{c}) = \min_{\mathbf{w}\in\{0,1\}^{n-m}} \frac{1}{n}\sum_{i=1}^{n} |\mathbf{s}_i - (\mathbf{G}\mathbf{w})_i|,$$



(a)

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

(b)

$$\begin{aligned} \mathbf{c}_a + \mathbf{w}_i + \mathbf{w}_k &= 0 \\ \mathbf{c}_b + \mathbf{w}_j &= 0 \\ \mathbf{c}_c + \mathbf{w}_i &= 0 \\ \mathbf{c}_d + \mathbf{w}_k &= 0 \\ \mathbf{c}_e + \mathbf{w}_i + \mathbf{w}_j + \mathbf{w}_k &= 0 \\ \mathbf{c}_f + \mathbf{w}_k &= 0 \end{aligned}$$

(c)

Figure 3.1: Factor graph representation of a linear code with generator matrix $\mathbf{G}$.

for arbitrary but constant source sequence $\mathbf{s} \in \{0,1\}^n$. We represent the code $\mathcal{C}$ by its generator matrix $\mathbf{G}$. For construction of the message-passing algorithm, we will represent the equation $\mathbf{c} = \mathbf{G}\mathbf{w}$ by a graph called the *factor graph*. In Figure 3.1 (a), we can see the factor graph representation of the matrix from Figure 3.1 (b). Each factor graph is given by the generator matrix of the code and contains three types of nodes: *information bits (info bits)*, *check nodes*, and their associated *source bits*. The factor graph is constructed from the set of equations we obtain by subtracting the vector $\mathbf{c}$ from the original equations for $\mathbf{c} = \mathbf{G}\mathbf{w}$. Due to the symmetry of minus operation in GF(2), we can represent each equation by the check node and connect each information bit that is present in this equation with an edge. Each check node performs XOR over the set of all connected info bits and its associated source bit. We say that the *check node is satisfied* if the result of the addition is zero, otherwise the *check is not satisfied*. An example of these equations used for constructing the factor graph from Figure 3.1 (a) is shown in Figure 3.1 (c). Note that there is one-to-one correspondence between a check node and its source bit.

We now describe the notation we will use later in this chapter. We define the set $V = \{1, \dots, n-m\}$ as a set of all info bits and we use variables $i, j, k \in V$ as an index variables to refer to these information bits. Similarly, we define the set $C = \{1, \dots, n\}$ as the set of all check nodes in the graph and use variables $a, b, c \in C$ to refer to these nodes or their associated source bits. To describe adjacent nodes in a graph, we define the set $C(i) = \{a \in C | \mathbf{G}_{a,i} = 1\}$ as a set of all checks connected to information bit i. Conversely, we define the set $V(a) = \{i \in V | \mathbf{G}_{a,i} = 1\}$ as a set of all information bits adjacent to the check a. To refer to the set of all bits - both information and source connected to check a, we use the set $\overline{V}(a) = V(a) \cup \{a\}$. We will say that *information bit has degree d* if this information bit has d adjacent check nodes and *check node has degree d* if it is connected to d information bits (we do not count the connected source bit).

To formalize the process of creating codes with a given rate R and length n, we will use the following terminology. We say that the code generated by matrix $\mathbf{G}$ has *degree distribution*

$$\rho^N(x) = 0.44x + 0.3x^2 + 0.17x^6 + 0.06x^{14} + 0.03x^{39}$$
$$\lambda^N(x) = x^9$$

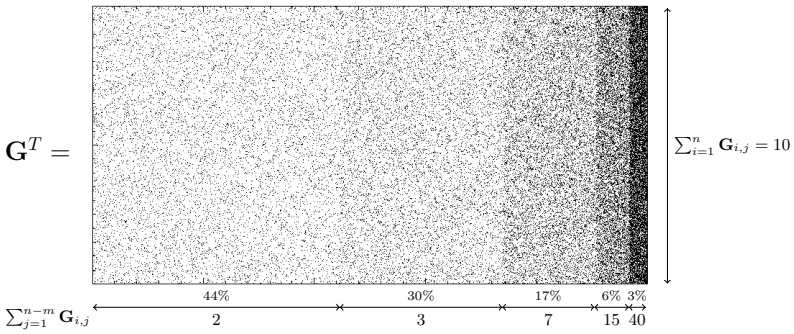


Figure 3.2: Example of a matrix $\mathbf{G}$ obtained using degree distribution (ρ^N, λ^N).

(from edge perspective) (ρ, λ), defined as:

$$\rho(x) = \sum_{i=1}^{d_R} \rho_i x^{i-1}, \qquad \lambda(x) = \sum_{i=1}^{d_L} \lambda_i x^{i-1}, \tag{3.1}$$

when the factor graph associated with this code has ρ_i and λ_i portion of all edges connected to check nodes and info bits with degree i, respectively. We denoted d_R, d_L the maximum check and information bit degree, respectively. From this construction, it follows that both polynomials $\rho(x)$ and $\lambda(x)$ have to have non-negative coeficients and fulfil $\rho(1) = 1$ and $\lambda(1) = 1$. In this thesis, we will mostly work with degree distributions defined from the edge perspective. There is another representation, sometimes used for the generator matrix $\mathbf{G}$ in practice, called *degree distribution from the node perspective*. This representation has the same form defined in (3.1), however the meaning of each coefficient is different. We denote the degree distribution from the node perspective as (ρ^N, λ^N). The coefficient ρ_i^N expresses the ratio of check nodes with degree i. The interpretation of λ_i^N for info bits is similar. Each degree distribution can be converted between both representations using the following equations:

$$\rho_i = \frac{i\rho_i^N}{\sum_{j=1}^{d_R} j\rho_j^N}, \qquad \rho_i^N = \frac{\rho_i/i}{\sum_{j=1}^{d_R} \rho_j/j}, \tag{3.2}$$

and similarly for λ. Using this notation, we can express the rate of the code as:

$$R = \frac{\sum_{i=1}^{d_R} i\rho_i^N}{\sum_{j=1}^{d_L} j\lambda_j^N} = \frac{\overline{\rho^N}}{\overline{\lambda^N}},$$

where $\overline{\rho^N}$ denotes the average check degree and $\overline{\lambda^N}$ average info bit degree from the node perspective.

In Figure 3.2, we can see an example of a randomly constructed matrix that has the degree distribution from the node perspective defined by (ρ^N, λ^N) shown in the same figure. To show the structure of the matrix, we sort the rows by their degrees.

3.3 Motivation for solving MAX-XOR-SAT problem

In this section, we formulate an approach for solving the MAX-XOR-SAT problem. This approach will be realized in the next section by a message-passing algorithm. We will start the formulation by assuming that we know some part of the optimal solution to the MAX-XOR-SAT problem (e.g., we know some fraction of bits from the optimal solution $\mathbf{w}^*$). Next, we use our approach to "recover" the unknown fraction of bits from the optimal solution $\mathbf{w}^*$. In the end, this assumption will be reformulated, however the whole methodology will be the same. While in practice it is not possible to fulfil the assumption about partial knowledge of the optimal solution, we can still assume the existence of such a solution. In this section, we assume we have an instance of MAX-XOR-SAT problem defined by a sparse matrix $\mathbf{G} \in \{0,1\}^{n\times(n-m)}$ and a source sequence $\mathbf{s}$.

First, assume that we know the optimal solution $\mathbf{w}^*$ and that $\mathbf{w} = \mathbf{w}^*$ up to one bit $\mathbf{w}_i$ that is unknown (we know the index i). In this case, we can easily find the missing bit $\mathbf{w}_i$ by choosing the bit, that will minimize the distortion $d_H(\mathbf{s}, \mathbf{Gw})$. Using the way how we obtain $\mathbf{w}$, we can say that our choice of $\mathbf{w}_i$ is the best one. Using this simple example, we add some notation we will use in this section. We divide the set of all check nodes $C(i)$ for each information bit i into two disjoint subsets $C(i) = C_0(i) \cup C_1(i)$ as follows:

$$C_0(i) = \left\{a \in C(i) \,\middle|\, \textstyle\sum_{j\in\overline{V}(a)\setminus\{i\}} \mathbf{w}_j = 0 \mod 2\right\}$$
$$C_1(i) = \left\{a \in C(i) \,\middle|\, \textstyle\sum_{j\in\overline{V}(a)\setminus\{i\}} \mathbf{w}_j = 1 \mod 2\right\}.$$

When we know the value of all information bits, we can interpret the set $C_0(i)$ as the set of all check nodes connected to information bit i that will be satisfied when $\mathbf{w}_i = 0$. The set $C_1(i)$ has the same interpretation for $\mathbf{w}_i = 1$. We say that *check a is forcing information bit i to be* 0 when $a \in C_0(i)$ and similarly for $a \in C_1(i)$. Using this notation, an unknown bit $\mathbf{w}_i$ should be set to $\mathbf{w}_i = 0$ if $|C_1(i)| < |C_0(i)|$ and to $\mathbf{w}_i = 1$ otherwise, because this strategy will violate fewer check nodes and generate less distortion.

Now we move to the more complicated case and still use the same problem and notation. Suppose that we have more bits unknown from the optimal solution, we have a set $U \subset V$ of unknown bits, where $|U| \ll |V|$. Again, we copy the known bits to vector $\mathbf{w}$ and we will try to recover the bits from U. To mark the unknown information bits, we extend the set of all possible values that each bit can have to $\{0, 1, *\}$. We set $\mathbf{w}_i = *$ for all unknown bits and $\mathbf{w}_i = \mathbf{w}_i^*$ otherwise.

In this case, generaly it is no longer possible to simply use the known bits to directly calculate the forcing value of all check nodes. This problem arises due to the fact that there can be dependencies between unknown bits. These dependencies are more frequent as the size of the set U increases. To solve this problem, we cannot go through all bit assignments and evaluate their distortion as we did before, because this strategy will give us an algorithm with exponential complexity.

We will approach this problem in an iterative fashion: we fix one information bit per round based on the relative amount of check nodes that will be unsatisfied when this bit is fixed. The idea is that by fixing an unknown information bit that will create the smallest number of contradictions regarding to the number of all check nodes in each round, we will obtain a greedy algorithm. This algorithm can find good assignments to our unknown information bits. In each round, we evaluate the *bias of information bit i* for each unknown information bit as:

$$B(i) = \frac{|C_0(i)| - |C_1(i)|}{|C_0(i)| + |C_1(i)|}. \tag{3.3}$$

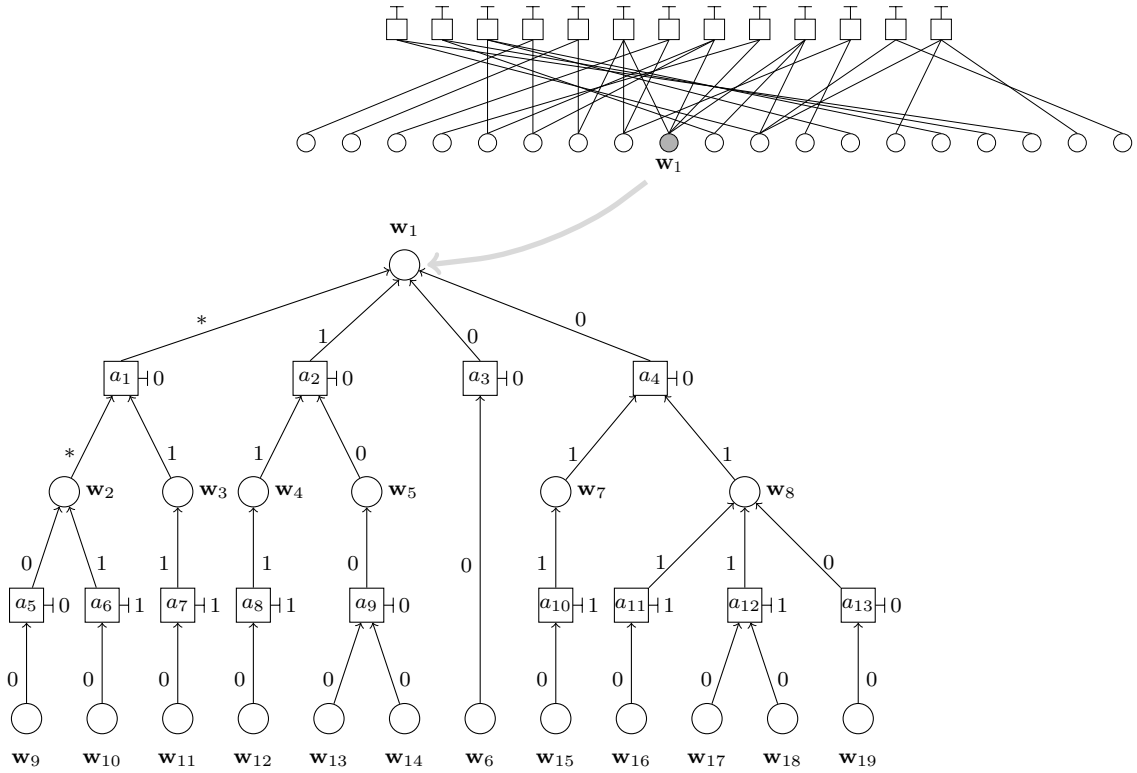


Figure 3.3: Calculating the bias of info bit $\mathbf{w}_1$ in a tree graph.

We fix info bit $j \in U$ that has the maximal $|B(j)|$ to value $\mathbf{w}_j = 0$ if $|C_1(j)| < |C_0(j)|$ and to $\mathbf{w}_j = 1$ otherwise. We will repeat this step and by fixing the most biased (in absolute value) unknown information bit in each round we "recover" all unknown bits in $|U|$ rounds.

To realize this idea, we have to describe how to evaluate the sets $C_0(i)$ and $C_1(i)$, when we have dependencies between unknown variables. Simply we would try to eliminate the unknown variables by trying to satisfy all other check nodes. To give an example of this process, we calculate the bias of the unknown variable w_1, $1 \in U$ from Figure 3.3. Without lost of generality, we consider the case when the last layer is all zeros. We can represent the dependencies from factor graph by reordering the nodes in a layered fashion as in Figure 3.3. We put information bit $\mathbf{w}_1$ on the top and layer the rest of the graph according to the shortest path of each node to $\mathbf{w}_1$. Here we make an assumption, that the graph obtained from this layered construction is a tree. This fact is not true in general and we will discuss the validity of this assumption at the end of this section. In this figure, each leaf in the tree represents a known information bit and the rest of circles represent unknown information bits.

To obtain the value of $\mathbf{w}_1$, we start by substituting known values from leafs into their adjacent check nodes. Due to the known values of substituted information bits, we can always construct the sets $C_0(i)$ and $C_1(i)$ for each information bit i which is in the upper layer from leaves. For example, from Figure 3.3 we can see that the sets of forcing checks for information bit $\mathbf{w}_8$ are the following: $C_0(\mathbf{w}_8) = \{a_{13}\}$ and $C_1(\mathbf{w}_8) = \{a_{11}, a_{12}\}$. To obtain $a_{12} \in C_1(\mathbf{w}_8)$ we use

$$\mathbf{w}_{17} + \mathbf{w}_{18} + \mathbf{w}_8 + a_{12} = 0,$$

where we substitute $\mathbf{w}_{17} = \mathbf{w}_{18} = 0$ and source sequence observation $a_{12} = 1$, and hence $\mathbf{w}_8 = 1$.

In the next step, we will obtain the values of information bits $\mathbf{w}_2, \ldots, \mathbf{w}_5, \mathbf{w}_7, \mathbf{w}_8$ by combining the sugestions that come from check nodes from lower layers of the graph. We can evaluate biases for these unknown information bits and do the following decision: if the bias is nonzero then fix this bit to the value that will violate less adjacent check nodes (in lower layer), otherwise set this bit to $*$. The $*$ value means that we still cannot determine the solution of this bit, because there were the same number of checks forcing this bit to 0 and to 1. This situation is demonstrated in Figure 3.3 by information bit $\mathbf{w}_2$. Check a_5 is forcing $\mathbf{w}_2$ to be 0, while check a_6 is forcing $\mathbf{w}_2$ to be 1. Because there is no other check that could change this zero bias value, the $\mathbf{w}_2$ gets $*$. On the other side of the graph, we set the information bit $\mathbf{w}_8$ to 1, because the set of check nodes forcing $\mathbf{w}_8$ to 1 is larger.

To induce the value of $\mathbf{w}_1$, we will continue with this process along the path from leaf to root nodes in the graph. To complete this process, we should describe how to handle $*$ values when we want to calculate whether the check will be forcing to 0 or 1. This process will be described using check a_1 in Figure 3.3. To give the description, we have to extend our notation and introduce new check nodes partitioning in the following way. For each information bit, define $C(i) = C_0(i) \cup C_1(i) \cup C_*(i)$, where

$$
\begin{aligned}
C_0(i) &= \left\{ a \in C(i) \,\middle|\, \left(\forall j \in \overline{V}(a)\backslash\{i\},\ \mathbf{w}_j \neq *\right) \wedge \left(\textstyle\sum_{j \in \overline{V}(a)\backslash\{i\}} \mathbf{w}_j = 0 \mod 2\right) \right\} \\
C_1(i) &= \left\{ a \in C(i) \,\middle|\, \left(\forall j \in \overline{V}(a)\backslash\{i\},\ \mathbf{w}_j \neq *\right) \wedge \left(\textstyle\sum_{j \in \overline{V}(a)\backslash\{i\}} \mathbf{w}_j = 1 \mod 2\right) \right\} \\
C_*(i) &= \left\{ a \in C(i) \,\middle|\, \exists j \in \overline{V}(a)\backslash\{i\},\ \mathbf{w}_j = * \right\}.
\end{aligned}
$$

The interpretation of each set $C_0(i)$ and $C_1(i)$ is similar with our previous definition. The sumations are defined only for check nodes $a \in C$ that do not receive any $*$ value from information bits in $\overline{V}(a)\backslash\{i\}$. Only these check nodes can force the value of information bit i. We cannot induce any value from check node that receive at least one $*$ value, because some information bits are uncertain. The result of this operation is that this check node will belong to the set $C_*(i)$. Next, we will use the same equation (3.3) to calculate the bias of each information bit. This equation has the following meaning: the information used for fixing the bit's value is obtained only from the check nodes that are forcing this bit. We do not count the check nodes that received $*$ to influence the bit's value. In Figure 3.3, the check node a_1 receives the $*$ value from the information bit $\mathbf{w}_2$. Therefore, the check node a_1 does not influence the bias $B(\mathbf{w}_1)$.

Using this approach, we calculate the bias of one unknown information bit $\mathbf{w}_1$. We can use the same strategy to calculate the bias of each unknown bit and select the most biased one, which we fix using the strategy described above. The complexity of calculating the bias for one unknown bit depends on the number of edges we have in the graph. For our sparse codes, we obtain linear complexity in code length. Although this strategy is only sub-optimal, we can expect that it can give us good results in the problem when we know which information bits we need to "recover". The sub-optimality is the price we pay for low complexity of the algorithm.

To remove our assumption about partial knowledge of optimal solution $\mathbf{w}^*$, we use the algorithm for the case $U = V$ and use arbitrary (but constant) vector $\mathbf{w}^0$ as an initial solution. In this case, the algorithm tries to recover the optimal solution and uses the initial solution to be able to induce the correct values. Here, we should note that the information used for fixing each bit is obtained from the whole graph and is global. Each bit is examined

whether it is the best candidate to fix its value according to the current solution. Based on this idea, we formulate the following scheme for solving the MAX-XOR-SAT problem.

1. Initialize the set of unknown information bits as $U = V$.
2. Start with initial solution $\mathbf{w} = \mathbf{w}^0$, where $\mathbf{w}^0 \in \{0,1\}^{n-m}$ is chosen arbitrarily.
3. Calculate bias $B(i)$ for each unknown information bit $i \in U$. To do this, subsitute bits from $\mathbf{w}$ to leaf nodes in a tree graph obtained by layered construction (see Figure 3.3).
4. Change the most biased ($j = \arg\max_{i \in U} |B(i)|$) unknown information bit $j \in U$ to $\mathbf{w}_j = 0$ if $|C_1(j)| \leqslant |C_0(j)|$ otherwise $\mathbf{w}_j = 1$.
5. Remove bit j from U.
6. If $U \neq \varnothing$, go to Step 3, otherwise declare vector $\mathbf{Gw}$ as the solution.

To complete our discussion, we should discuss the validity of the tree assumption we made when we want to induce unknown variables from the initial solution. We assumed that the graph obtained from the layered construction was a tree. We need this assumption and we cannot simply remove it, because we need the fact that each node in this graph has exactly one parent node and therefore we can induce exactly one variable from it. On the one hand, the problem is that we cannot construct a random graph according to a predefined degree distribution and assume that it will be a tree. Trees have a small number of edges and we cannot use them for data compression. On the other hand, due to the sparsness of the graph, we can expect that the number of cycles in this graph will be small and hence we can approximate the original graph with cycles using the tree graph. In general, the presence of short cycles usually causes a problem with convergence of message-passing algorithms. Although we know this behavior, the result is still usefull and we will make some modification of the original message-passing algorithm to avoid the influence of short cycles.

3.4 Bias Propagation, intuitive approach

In this section, we will use the motivation for solving the MAX-XOR-SAT problem as explained above to develop an exact message-passing algorithm. We will generalize the process of inducing unknown information bits from some initial solution and show that it can be realized using a message-passing algorithm sending only one real number message in each direction. We will give exact interpretation of each message and due to this interpretation, we call the algorithm *Bias Propagation*.

According to our motivation, we will start with some arbitrary initial solution $\mathbf{w} = \mathbf{w}^*$ and declare each information bit as unknown ($U = V$). Next, we should construct an algorithm that will find the most biased unknown information bit, fix it and declare this bit as known. These three steps should be repeated until the set of unknown bits is empty. To further describe the algorithm, we will use the term *one round* for the group of these steps. To find the most biased bit, we should calculate the bias for each bit in U. To do this, we will use another iterative approach which will consist of sending messages along edges in the graph representation of our code. Messages will be sent from information bits to check nodes, recalculated and forwarded back to information bits. We will call the sequence of these two message updates *one iteration*.

To realize the whole algorithm, we start with the description of the message-passing iteration process. This sequence of iterations should follow the idea presented in the previous section. Simultaneously, it should be able to simulate the process of inducing the information bit (the

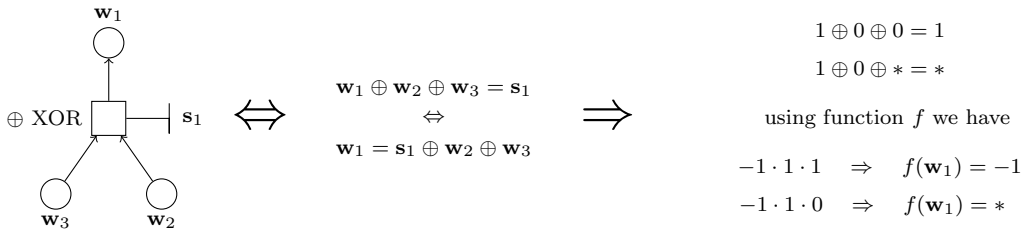


Figure 3.4: Example of check node calculation.

root node in the tree graph) from known values of other information bits (leaf nodes). To represent the value of each information bit, we use the following invertible mapping:

$$\mathbf{b}_i = f(\mathbf{w}_i) = (-1)^{\mathbf{w}_i}, \tag{3.4}$$

where $\mathbf{w}_i \in \{0, 1\}$ is the value of each information bit. This mapping is important to us, because of this property:

$$\mathbf{w}_1 \oplus \mathbf{w}_2 = f^{-1}\big(f(\mathbf{w}_1) \cdot f(\mathbf{w}_2)\big) \qquad \forall \mathbf{w}_1, \mathbf{w}_2 \in \{0, 1\}, \tag{3.5}$$

where $\mathbf{w}_1 \oplus \mathbf{w}_2$ is the binary XOR of $\mathbf{w}_1$ and $\mathbf{w}_2$, or sum in $GF(2)$. We will use this mapping to simulate the check node and induce the value of the bit that is pointing to the root node in a tree graph. In our motivation, we had to enlarge the set of possible values for each non-leaf information bit from $\{0, 1\}$ to $\{0, 1, *\}$ (see bit $\mathbf{w}_2$ in Figure 3.3). We used the $*$ value with each information bit, for which we could not induce the value of this bit. This was because the number of checks forcing $\mathbf{w}_i$ to 0 and to 1 was the same (contradiction occurs). The mapping (3.4) can be generalized to capture this new value by putting $f(*) = 0$. We can now use the same equation (3.5) for calculating the output of the check node. When the check node receives the $*$ value (some bit is uncertain about its value), we cannot induce the value of the bit i that is facing the root. In the previous section, every check node that receives at least one $*$ value belongs to the set $C_*(i)$ and we did not use them to calculate the final value of this bit i. See Figure 3.4 for examples of how to use equation (3.5) and the extended mapping (3.4) to calculate the value of the information bit $\mathbf{w}_1$. Finally, we will use the whole interval $[-1, +1]$ to represent values of each information bit after the mapping f and interpret the bias $|f(\mathbf{w}_i)|$ as a measure of certainty as shown in Figure 3.5.

In the ℓ-th iteration, each information bit ($i \in V$) will send the message $B^{(\ell)}_{i\to a} \in [-1, +1]$ to each connected check node ($a \in C$) and each check node will forward the message $S^{(\ell)}_{a\to i} \in [-1, +1]$ back. We will call the message $B^{(\ell)}_{i\to a}$ *bias of information bit* i to check node a in ℓ-th iteration and interpret this value as described earlier (Figure 3.5). Message $S^{(\ell)}_{a\to i}$ will be called *satisfaction of check node* a with assignment of all connected bits without i (set $\overline{V}(a)\backslash\{i\}$). The interpretation of this message is the following: if $S^{(\ell)}_{a\to i} = +1$ then the check is completely satisfied using the bit assignment from the set $\overline{V}(a)\backslash\{i\}$, which implies that bit i should be set to $\mathbf{w}_i = 0$. When $S^{(\ell)}_{a\to i} = -1$, the check node is completely unsatisfied, and we have $\mathbf{w}_i = 1$ to satisfy the check node (and generate less distortion). When $S^{(\ell)}_{a\to i} = 0$, the check is not sure about its satisfiability.

We can now write the first update rule for calculating the message $S^{(\ell)}_{a\to i}$ from messages $B^{(\ell)}_{i\to a}$

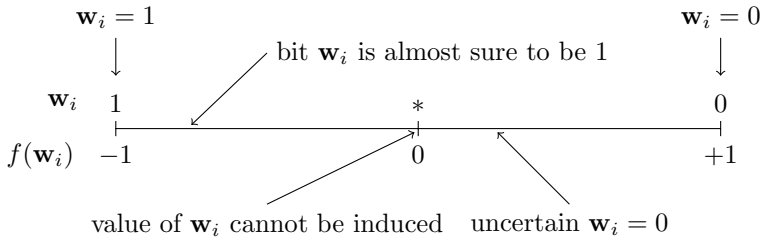


Figure 3.5: Value of information bit after mapping f from equation (3.4) and its generalization to whole interval $[-1, +1]$.

in the form:

$$S_{a\to i}^{(\ell)} = \prod_{j\in\overline{V}(a)\setminus\{i\}} B_{j\to a}^{(\ell)}, \tag{3.6}$$

where the message $B_{\mathbf{s}_a\to a}^{(\ell)} \in [-1, +1]$ represents the constant bias from the source bit $\mathbf{s}_a$. We will specify the value of this message later in this section.

To write the update equation for $B_{i\to a}^{(\ell+1)}$, we will follow the motivation section and evaluate the bias based on the size of each set $C_0(i), C_1(i), C_*(i)$, where we omit the check node a. This can be done efficiently because we already know whether some other check $b \in C, b \neq a$ is forcing bit i to $0, 1$ or is not forcing at all. We know this from the previous message $S_{b\to i}^{(\ell)}$. The only thing we have to incorporate compared to our previous motivation is the knowledge of the strength of satisfaction that each check node is sending to bit i. This can be done using the following update equation:

$$B_{i\to a}^{(\ell+1)} = \frac{\prod_{b\in C(i)\setminus\{a\}}\left(1 + S_{b\to i}^{(\ell)}\right) - \prod_{b\in C(i)\setminus\{a\}}\left(1 - S_{b\to i}^{(\ell)}\right)}{\prod_{b\in C(i)\setminus\{a\}}\left(1 + S_{b\to i}^{(\ell)}\right) + \prod_{b\in C(i)\setminus\{a\}}\left(1 - S_{b\to i}^{(\ell)}\right)}. \tag{3.7}$$

This equation expresses the final decision of information bit i to check a based on incoming suggestions from all connected check nodes without a. We can see that the difference in the numerator acts as the difference between the sizes of the sets $|C_0(i)|$ and $|C_1(i)|$ which we had in our motivation section. Finally, the denominator is used to normalize the difference by the amount of received satisfactions that differ from the $*$ value ($S_{a\to i}^{(\ell)} = 0$). Equation (3.7) returns $B_{i\to a}^{(\ell+1)} > 0$ (bit i tends to be 0) when the majority of satisfaction messages are positive (majority of check nodes is forcing bit i to 0).

Using equations (3.6) and (3.7), we obtain an iterative approach to induce non-root information bits when we consider some bit i as a root. This process does it in parallel for every bit $i \in V$. As we have described earlier, we initialize each leaf node by arbitrary solution which we denoted $\mathbf{w}^0$. For practical implementation, it would be easier to assume that $\mathbf{w}_i^0 = 0$, $\forall i \in V$. We use $B_{i\to a}^{(1)} = 1$ as the initial bias message for each edge in the graph. The iterative process is started by calculating $S_{a\to i}^{(1)}$. In each round, we use $\hat{\ell}$ iterations to induce all none-root variables, where $\hat{\ell}$ correspons to the number of layers in the tree graph. In practice, we will use $\hat{\ell}$ as a parameter expressing the convergence criterion for each round. Finally, to evaluate the bias B_i of each unknown information bit (bias of the root node in the tree graph) in $\hat{\ell}$-th iteration, we use a similar equation to (3.7), however we do not omit

any check node. We use the whole set $C(i)$ to perform this operation, because we assume we are calculating the bias for the root nodes in parallel. Hence,

$$B_i = \frac{\prod_{b\in C(i)}\left(1+S_{b\to i}^{(\hat{\ell})}\right)-\prod_{b\in C(i)}\left(1-S_{b\to i}^{(\hat{\ell})}\right)}{\prod_{b\in C(i)}\left(1+S_{b\to i}^{(\ell)}\right)+\prod_{b\in C(i)}\left(1-S_{b\to i}^{(\ell)}\right)}. \tag{3.8}$$

The $|B_i|$ expresses the nature of an unknown information bit i to be fixed without generating a large number of contradictions in the system. If the value is small ($|B_i| \approx 0$), then this variable does not have a tendency to be fixed without generating many contradictions. On the other hand, $|B_i| \approx 1$ means that this variable should be fixed and this operation will satisfy the majority of connected check nodes. Using this interpretation, we fix the most biased unknown information bit ($j = \arg\max_{i\in U} |B_i|$) to $\mathbf{w}_j = 0$ if $B_j > 0$ and to $\mathbf{w}_j = 1$ otherwise.

To complete the first round, we should specify the constant source messages $B_{\mathbf{s}_a\to a}^{(\ell)}$ we used in each iteration to calculate the message $S_{a\to i}^{(\ell)}$. These messages reflect the value of each source bit and take control of the whole quantization process. Here we show how to initialize these messages in the first round, where $U = V$. The case for the other rounds will be slightly modified and we will discuss this case in the next paragraph. The initialization is done by

$$B_{\mathbf{s}_a\to a}^{(\ell)} = (-1)^{\mathbf{s}_a}\tanh(\gamma) \qquad \forall \ell \quad \text{in the first round,} \tag{3.9}$$

where $\mathbf{s}_a$ is a-th bit from source sequence $\mathbf{s}$ and γ is a constant parameter. The γ parameter reflects the effort of the message-passing algorithm to find the resulting codeword $\hat{\mathbf{s}} = \mathbf{G}\mathbf{w}$ as close to $\mathbf{s}$ as possible. The larger the γ, the stronger is the effort. On the other hand, the structure of the code $\mathcal{C}$ imposes a limit on how strong this effort can be.

In the first round, we have all bits declared as unknown and we choose one ($i \in U$) to be fixed after $\hat{\ell}$ iterations of the message-passing algorithm. The value of this bit will never be changed in any round. To speed up the process of message passing without influencing the quality of the quantizer, we can substitute this value and remove bit i from the graph. By doing this, we simplify the problem by removing one information bit with all edges connected to this bit. In this simplified problem, we do not have bit i and hence each information bit in the graph is unknown. We can now apply the same message-passing algorithm again. The process of removing the fixed information bit and reducing the graph is called *decimation*.

To decimate a fixed bit in the r-th round, we will use the substitution to create matrix $\mathbf{G}^{(r+1)}$ and a new source sequence $\mathbf{s}^{(r+1)}$ that we will be used in the next round. The quantization problem with matrix $\mathbf{G}^{(r+1)}$ and source sequence $\mathbf{s}^{(r+1)}$ should be equivalent to the quantization problem we had in the r-th round with some fixed information bit. As can be seen from Figure 3.6, the decimation process creates a new source sequence by applying XOR function to each bit connected to the fixed information bit (it propagates the fixed value to the graph). This operation ensures that we obtain an equivalent quantization problem after removing the fixed information bit and all its adjacent edges. Finally, we can simplify the graph by removing all check nodes with degree 0. By removing check node $a \in C$, we have to remove its associated source bit $\mathbf{s}_a$ and truncate the whole vector to obtain a new source sequence $\mathbf{s}^{(r+1)}$ for round $r+1$. This truncated sequence is used to calculate new source messages $B_{\mathbf{s}_a\to a}^{(\ell)}$ using the same equation (3.9). Source messages are calculated and are constant in each round. There are no messages from any check node to its source bit.

As we have discussed at the end of the previous section, the whole algorithm was derived using the assumption that the underlying factor graph was a tree. In practical applications,

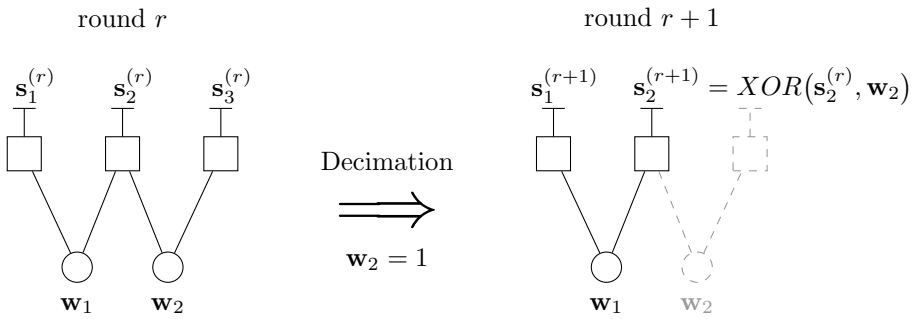


Figure 3.6: Example of decimation process.

we have to deal with random graphs that contain cycles. The problem with cycles is that the message-passing algorithm does not converge well and messages that flow in short cycles tend to oscilate. To avoid (short) cycles we have two possibilities: we can use some "smart" algorithm to generate the random matrix $\mathbf{G}$ without short cycles, or we can use messages from the previous iteration to avoid rapid changes in message values. Here, we will use the second idea and postpone the study of cycle length dependence to our future research. To smooth out the exchanged messages, we will combine the message from the previous iteration with the message calculated in the current iteration and save this combination as the resulting message for the current iteration. This operation is often called *damping*. We will use this operation to smooth $B_{i\to a}^{(\ell+1)}$ messages and calculate these messages from $B_{i\to a}^{(\ell)}$ and $S_{a\to i}^{(\ell)}$ using the following equation:

$$\tilde{B}_{i\to a} = \frac{\prod_{b\in C(i)\setminus\{a\}}\left(1+S_{b\to i}^{(\ell)}\right) - \prod_{b\in C(i)\setminus\{a\}}\left(1-S_{b\to i}^{(\ell)}\right)}{\prod_{b\in C(i)\setminus\{a\}}\left(1+S_{b\to i}^{(\ell)}\right) + \prod_{b\in C(i)\setminus\{a\}}\left(1-S_{b\to i}^{(\ell)}\right)}, \tag{3.10}$$

$$B_{i\to a}^{(\ell+1)} = \frac{\sqrt{(1+\tilde{B}_{i\to a})(1+B_{i\to a}^{(\ell)})} - \sqrt{(1-\tilde{B}_{i\to a})(1-B_{i\to a}^{(\ell)})}}{\sqrt{(1+\tilde{B}_{i\to a})(1+B_{i\to a}^{(\ell)})} + \sqrt{(1-\tilde{B}_{i\to a})(1-B_{i\to a}^{(\ell)})}}. \tag{3.11}$$

Here, we postpone the formal derivation of this equation to Section 3.6.3. First, we use the original update equation (3.7) to calculate the bias $\tilde{B}_{i\to a}$ and then calculate the final message using (3.11). This is done by taking an "average" (using square root and multiplication) with the bias $B_{i\to a}^{(\ell)}$ from the previous iteration.

In practice, we need $n-m$ rounds (we have $n-m$ information bits) to decimate 1 info bit per round to completely reduce the graph. This process is the best we can do by using this algorithm. To speed up the graph reduction, we can try to reduce more information bits per one round. We have to be careful in this process, because we cannot decimate so many bits when the maximal bias is small. When the maximum of $|B_i|$ is almost zero, it means that we are choosing the value of the bit to be fixed almost randomly. In this case, we should avoid removing many bits. Using this dependence, we will describe the so-called *decimation strategy* and use this strategy to remove varying number of bits based on the maximum value of $|B_i|$ in each round. We define the strategy based on three parameters which we set at the beginning of the quantization process and keep them constant for all subsequent rounds. In each round, we decimate `num_min` of the most biased information bits when $\max|B_i| <$ `t`, otherwise we decimate all bits with $|B_i| >$ `t`, but no more than `num_max`. We will specify

(a) pseudo-code

```
procedure w = BiP(G, s)
  while not all_bits_fixed(w)
    bias = BiP_iter(s, G)
    bias = sort(bias)
    if max(bias)>t
      num = min(num_max, num_of_bits(bias>t))
    else
      num = num_min
    [G,s,w] = dec_most_biased_bits(G,s,w,num)
  end
end
```

```
procedure bias = BiP_iter(s, G)
  B_saa = calc_source_msg(s, gamma)   /* eq. (BiP-1) */
  B_ia = 1          /* initialize all bias messages */
  S_ai = calc_ai(B_ia)                /* eq. (BiP-3) */
  while iter<max_iter
    B_ia_old = B_ia
    B_ia = calc_ia(S_ai)              /* eq. (BiP-2) */
    if iter>start_damp then
      B_ia = damping(B_ia, B_ia_old)  /* eq. (BiP-4) */
    end
    S_ai = calc_ai(B_ia, B_saa))      /* eq. (BiP-3) */
    iter = iter+1
  end
  bias = calc_bias(S_ai)              /* eq. (BiP-5) */
end
```

(b) message-passing update rules

Source message initialization:

$$B^{(\ell)}_{\mathbf{s}_a \to a} = (-1)^{\mathbf{s}_a} \tanh(\gamma) \qquad \text{(BiP-1)}$$

Bias update rule:

$$B^{(\ell+1)}_{i\to a} = \frac{\prod_{b\in C(i)\setminus\{a\}}\left(1+S^{(\ell)}_{b\to i}\right) - \prod_{b\in C(i)\setminus\{a\}}\left(1-S^{(\ell)}_{b\to i}\right)}{\prod_{b\in C(i)\setminus\{a\}}\left(1+S^{(\ell)}_{b\to i}\right) + \prod_{b\in C(i)\setminus\{a\}}\left(1-S^{(\ell)}_{b\to i}\right)}. \qquad \text{(BiP-2)}$$

Satisfaction update rule:

$$S^{(\ell)}_{a\to i} = \prod_{j\in \overline{V}(a)\setminus\{i\}} B^{(\ell)}_{j\to a} \qquad \text{(BiP-3)}$$

Equation for damping update in round $\ell+1$ ($\tilde{B}_{i\to a}$ is obtained from (BiP-2)):

$$B^{(\ell+1)}_{i\to a} = \frac{\sqrt{(1+\tilde{B}_{i\to a})(1+B^{(\ell)}_{i\to a})} - \sqrt{(1-\tilde{B}_{i\to a})(1-B^{(\ell)}_{i\to a})}}{\sqrt{(1+\tilde{B}_{i\to a})(1+B^{(\ell)}_{i\to a})} + \sqrt{(1-\tilde{B}_{i\to a})(1-B^{(\ell)}_{i\to a})}}. \qquad \text{(BiP-4)}$$

Equation for calculating final bias B_i in round $\hat{\ell}$:

$$B_i = \frac{\prod_{b\in C(i)}\left(1+S^{(\hat{\ell})}_{b\to i}\right) - \prod_{b\in C(i)}\left(1-S^{(\hat{\ell})}_{b\to i}\right)}{\prod_{b\in C(i)}\left(1+S^{(\ell)}_{b\to i}\right) + \prod_{b\in C(i)}\left(1-S^{(\ell)}_{b\to i}\right)}. \qquad \text{(BiP-5)}$$

Constant parameters:
`num_max`... maximum info bits to decimate
`num_min`... minimum info bits to decimate
`t`... decimation threshold
`gamma` ... check node satisfaction strength
`max_iter` ... max. # of iteration in round
`start_damp` ... # iterations without damping

Figure 3.7: Summary of Bias Propagation algorithm.

the values of all parameters in Chapter 5. Here, we assume that `num_min` < `num_max`.

To present the complete Bias Propagation algorithm, we give the summary of update-equations and pseudo-code in Figure 3.7.

3.5 Weighted binary quantization

In the previous section, we developed an algorithm for binary quantization problem. However, in the beginning we were interested in an algorithm for a weighted quantization problem. We now study the Bias Propagation algorithm and describe its generalization. In order to use this algorithm for finding the nearest codeword in some weighted sum distortion measure. More formally, we should find the minimum

$$\min_{\mathbf{c}\in\mathcal{C}} ||\mathbf{c}-\mathbf{s}||_D = \min_{\mathbf{c}\in\mathcal{C}} \sum_{i=1}^{n} \varrho_i |\mathbf{c}_i - \mathbf{s}_i|,$$

where ϱ_i measures the cost of making a change in the i-th bit.

Our problem is similar to the minimization problem, except now we should recognize the

source bit that can be changed from other bits that cannot. In some sense, we should control the effort of each check node to be satisfied or not. The parameter γ used in (BiP-1) in Figure 3.7 to initialize the source message, was defined uniformly for each source bit $\mathbf{s}_a$. The larger the γ was, the larger was the effort of the algorithm to satisfy all check nodes. By assigning to each source bit $\mathbf{s}_a$ its own parameter γ_a, we can control the probability of each source bit being preserved and thus minimize the total cost of quantizing the source sequence to the nearest codeword. Each γ_a will be a function of the original cost for source bit a. We study the practical results with the algorithm for optimizing vector of γ_a for given profile ϱ in Section 5.7.

3.6 Bias Propagation, formal derivation

Although the algorithm we just described works (even for arbitrary profile ϱ), and we can use the summarized description given in Figure 3.7 to implement it, we present another and more formal derivation of the whole algorithm. We think that both derivations are useful as they provide different views of the same idea. In Section 3.7, we will use the approach derived in this section to study the convergence of the Bias Propagation algorithm in each round and show that the convergence of $|B_i|$ can be predicted from the given degree distribution pair.

Here, we carry out the derivation for an arbitrary profile ϱ assuming that we know the vector $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n)$, $\gamma_i \geqslant 0 \ \forall i = 1, \ldots, n$ and reformulate the weighted binary quantization problem as Maximum A Posteriori (MAP) estimation. Let $\mathcal{C}$ be the LDGM code and $\mathbf{s} \in \{0,1\}^n$ be the fixed source sequence. For a given vector $\boldsymbol{\gamma}$, we define the following conditioned probability distribution:

$$P(\mathbf{c}|\mathbf{s};\boldsymbol{\gamma}) = \begin{cases} \frac{1}{Z} e^{-2\boldsymbol{\gamma}\cdot(\mathbf{c}-\mathbf{s})} & \text{if } \mathbf{c} \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

where $\boldsymbol{\gamma} \cdot (\mathbf{c} - \mathbf{s})$ is the dot product of vectors $\boldsymbol{\gamma}$ and $\mathbf{c} - \mathbf{s}$ (calculated in $\mathbb{F}_2$) and Z is a normalization constant in the form

$$Z = \sum_{\mathbf{c} \in \mathcal{C}} e^{-2\boldsymbol{\gamma}\cdot(\mathbf{c}-\mathbf{s})}. \tag{3.13}$$

From this definition, it should be clear that the most probable codeword is the optimal solution to our original problem. Although the problem of finding the MAP solution has the same complexity as the binary quantization problem, there exists a very efficient algorithm for calculating the marginal probabilities of distributions that has a special form. This algorithm, called *sum-product* or *belief propagation* [16] is used in different scientific fields and achieves great success. We will show the idea behind this algorithm on a simple example and finally state the results. A good text on the sum-product algorithm is [16].

3.6.1 Sum-product algorithm

Suppose that we define the following probability distribution over the space $\{0,1\}^6$

$$P(x_1, \ldots, x_6) = \frac{1}{Z} \psi_1(x_1) \cdots \psi_6(x_6) \psi_{a_1}(x_1, x_2, x_3) \psi_{a_2}(x_3, x_4) \psi_{a_3}(x_3, x_5, x_6), \tag{3.14}$$

where Z is the normalization constant and $\psi_i(\cdot)$, $\psi_{a_i}(\cdot)$ are non-negative functions.

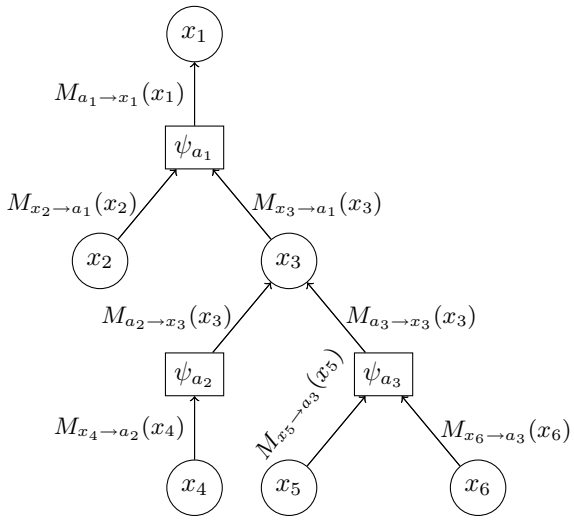


Figure 3.8: Graphical representation of probability distribution from (3.14).

We are interested in calculating the marginal probability for the first variable $P(x_1 = 0)$ and $P(x_1 = 1)$. From the definition of marginal probability and using the distributive law, we can write

$$P(x_1 = 0) = \psi_1(0) \sum_{x_2,x_3} \psi_2(x_2)\psi_{a_1}(0, x_2, x_3)\psi_3(x_3)\Big(\sum_{x_4} \psi_4(x_4)\psi_{a_2}(x_3, x_4)\Big)\Big(\sum_{x_5,x_6} \psi_5(x_5)\psi_6(x_6)\psi_{a_3}(x_3, x_5, x_6)\Big), \tag{3.15}$$

and similarly for $P(x_1 = 1)$. We use $\sum_{x_i}$ to denote $\sum_{x_i=0}^{1}$. Using the form of our distribution, we can draw a graph (see Figure 3.8) that represents the dependencies between functions ψ_{a_i} and variables x_i. We use the same notation to describe the graph structure as in Section 3.2 (imagine that the graph is a factor graph representing some code). Using this graphical representation, we can calculate the final marginal probabilities very efficiently using the following message-passing approach. Messages (2 component vectors) are passed from leaves to the root of the tree graph, where these messages are calculated using recurrent formulas. We start by calculating the term containing $(\sum_{x_4})$ in the (3.15) which depends only on variable x_3. We thus represent the result of this sum as vector $M_{a_2\to x_3}(x_3) = (M_{a_2\to x_3}(0), M_{a_2\to x_3}(1))$ (message passed from $a_2 \to x_3$)

$$\begin{aligned} M_{a_2\to x_3}(x_3) &= \begin{pmatrix} M_{a_2\to x_3}(0) \\ M_{a_2\to x_3}(1) \end{pmatrix} = \begin{pmatrix} \psi_4(0)\psi_{a_2}(0,0) + \psi_4(1)\psi_{a_2}(0,1) \\ \psi_4(0)\psi_{a_2}(1,0) + \psi_4(1)\psi_{a_2}(1,1) \end{pmatrix} = \\ &= \sum_{x_4} \psi_{a_2}(x_3, x_4) M_{x_4\to a_2}(x_j) = \\ &= \sum_{x_{V(a_2)}\setminus x_3} \Big[\psi_{a_2}(x_{V(a_2)}) \prod_{j\in V(a_2)\setminus x_3} M_{j\to a_2}(x_j)\Big], \end{aligned} \tag{3.16}$$

where

$$M_{x_4\to a_2}(x_i) = \psi_4(x_4) = \begin{pmatrix} \psi_4(0) \\ \psi_4(1) \end{pmatrix} = \psi_4(x_4) \prod_{b\in C(x_4)\setminus a_2} M_{b\to x_4}(x_4). \tag{3.17}$$

The last terms used in equation (3.16) and (3.17) are general recurrent formulas for calculating new messages. In (3.17), the general equation reduces to just $\psi_4(x_4)$. We can use the same approach and calculate the term containing $(\sum_{x_5,x_6})$ and denote it by $M_{a_3\to x_3}(x_3)$. Again, since it only depends on variable x_3, $M_{a_3\to x_3}(x_3) = (M_{a_3\to x_3}(0), M_{a_3\to x_3}(1))$. Using these results and the general update equation (3.17), we can calculate the term

$$\psi_3(x_3)\Big(\sum_{x_4}\psi_4(x_4)\psi_{a_2}(x_3,x_4)\Big)\Big(\sum_{x_5,x_6}\psi_5(x_5)\psi_6(x_6)\psi_{a_3}(x_3,x_5,x_6)\Big) \tag{3.18}$$

as $\psi_3(x_3)M_{a_2\to x_3}(x_3)M_{a_3\to x_3}(x_3)$. By using the update equation (3.16) for calculating messages from $a \to x$ and equation (3.17) for calculating messages from $x \to a$, we obtain

$$p(x_1=0) = \psi_1(0)\underbrace{\sum_{x_2,x_3}\underbrace{\psi_2(x_2)}_{M_{x_2\to a_1}}\psi_{a_1}(0,x_2,x_3)\underbrace{\psi_3(x_3)\Big(\sum_{x_4}\psi_4(x_4)\psi_{a_2}(x_3,x_4)\Big)\Big(\sum_{x_5,x_6}\psi_5(x_5)\psi_6(x_6)\psi_{a_3}(x_3,x_5,x_6)\Big)}_{M_{x_3\to a_1}(x_3)}}_{M_{a_1\to x_1}}. \tag{3.19}$$

To calculate the final marginal probability, we have to apply a slightly modified version of the update rule (3.17)

$$P(x_1) = \begin{pmatrix} P(x_1=0) \\ P(x_1=1) \end{pmatrix} = \begin{pmatrix} \psi_1(0)\prod_{b\in C(x_1)} M_{b\to x_1}(0) \\ \psi_1(1)\prod_{b\in C(x_1)} M_{b\to x_1}(1) \end{pmatrix} = \psi_1(x_1)\prod_{b\in C(x_1)} M_{b\to x_1}(x_1). \tag{3.20}$$

This rule combines the messages from each adjacent function node (here we only have one square in the graph). Using this approach, we calculated the marginal probability for one variable. In practice, by applying recurrent formulas and updating all edges in parallel, we can calculate the marginal probability for *all* variables. This can be done even for more general distributions that can be factorized and have the following form

$$P(x_1,x_2,\ldots,x_n) = \frac{1}{Z}\prod_{i=1}^{n}\psi_i(x_i)\prod_{a\in C}\psi_a(x_{V(a)}), \tag{3.21}$$

where Z is a normalization factor, by using so called *sum-product algorithm* that consists of iterative parallel updates of all edges in each direction. For the first iteration, we have $M^{(0)}_{a\to i} = 1\ \forall a \in C,\ \forall i \in V$ and continue by applying the following recurrent equations $(a \in C, i \in V, \ell = 1,2,\ldots)$:

$$M^{(\ell)}_{i\to a} = \psi_i(x_i)\prod_{b\in C(i)\setminus a} M^{(\ell-1)}_{b\to i}(x_i) \tag{3.22}$$

$$M^{(\ell)}_{a\to i} = \sum_{x_{V(a)}\setminus i}\Big[\psi_a(x_{V(a)})\prod_{j\in V(a)\setminus i} M^{(\ell)}_{j\to a}(x_j)\Big]. \tag{3.23}$$

After $\hat{\ell}$ iterations, we calculate the marginal probabilities as

$$P(x_i) = \psi_i(x_i)\prod_{b\in C(i)} M^{(\hat{\ell})}_{b\to i}(x_i). \tag{3.24}$$

The belief propagation algorithm is exact, when the factor graph is tree. When the fac-

tor graph contains cycles, then the belief propagation algorithm approximates the original marginal probabilities. In this case the number of iterations $\hat{\ell}$ can be constant, or it can be determined from the message convergence, e.g, $|M^{(\hat{\ell})} - M^{(\hat{\ell}+1)}| < \epsilon$.

3.6.2 Finding bitwise MAP using sum-product algorithm

To solve the weighted binary quantization problem, we express the probability distribution (3.12) in form (3.21). For $\mathbf{c} \in \mathcal{C}$, we can find $\mathbf{w} \in \mathbb{F}_2^m$, such that $\mathbf{c} = \mathbf{G}\mathbf{w}$. Thus, we can write

$$P(\mathbf{w}|\mathbf{s};\boldsymbol{\gamma}) = \prod_{a \in C} \psi_a(\mathbf{w}_{V(a)}, \mathbf{s}_a), \tag{3.25}$$

where

$$\psi_a\big(\mathbf{w}_{V(a)}, \mathbf{s}_a\big) = \begin{cases} e^{\gamma_a} & \text{if} \quad \mathbf{s}_a = \sum_{i \in V(a)} \mathbf{w}_i \\ e^{-\gamma_a} & \text{if} \quad \mathbf{s}_a \neq \sum_{i \in V(a)} \mathbf{w}_i. \end{cases} \tag{3.26}$$

This distribution is equivalent to (3.12), because we factorized the original distribution according to each check node and because

$$\frac{1}{Z} e^{-2\boldsymbol{\gamma}\cdot(\mathbf{c}-\mathbf{s})} = \frac{1}{Z} \frac{e^{\boldsymbol{\gamma}\cdot\mathbf{1}}}{e^{\boldsymbol{\gamma}\cdot\mathbf{1}}} e^{-2\boldsymbol{\gamma}\cdot(\mathbf{c}-\mathbf{s})} = \frac{1}{Z'} e^{-2\boldsymbol{\gamma}\cdot(\mathbf{G}\mathbf{w}-\mathbf{s})+\boldsymbol{\gamma}\cdot\mathbf{1}} \tag{3.27}$$

we obtain the distribution (3.25). We will use the sum-product algorithm to calculate marginal probabilities for each information bit and find the assignment for each information bit in the form

$$\hat{\mathbf{w}}_i = \arg \max_{\mathbf{w}_i \in \{0,1\}} P(\mathbf{w}_i|\mathbf{s}) = \arg \max_{\mathbf{w}_i \in \{0,1\}} \sum_{\sim \mathbf{w}_i} P(\mathbf{w}|\mathbf{s}) = \arg \max_{\mathbf{w}_i \in \{0,1\}} \sum_{\sim \mathbf{w}_i} \prod_{a \in C} \psi_a\big(\mathbf{w}_{V(a)}, \mathbf{s}_a\big). \tag{3.28}$$

Here, we are using a shortened notation. Instead of the sum over all information variables without the i-th one, we write $\sum_{\sim \mathbf{w}_i}$. To find the best assignment, we calculate the marginal probabilities for all bits and fix one information bit per round, so that the information bit maximizes $|P(\mathbf{w}_i = 0|\mathbf{s}) - P(\mathbf{w}_i = 1|\mathbf{s})|$. In each round, we fix one bit, reduce the graph using decimation, which we described in Figure 3.6, and repeat this process until we obtain all information bits.

To calculate (3.28) using sum-product algorithm efficiently, we simplify the original update equations. We show that the sum in equation (3.23) can be simplified, and the message-passing algorithm can be implemented using one real number. In the derivation, we use $\psi_i(x_i) = 1$. Using the original update equations (3.22), (3.23), (3.24) and notation used in the previous section, we define

$$R_{i \to a}^{(\ell)} = \frac{M_{i \to a}^{(\ell)}(1)}{M_{i \to a}^{(\ell)}(0)} = \frac{\prod_{b \in C(i) \setminus a} M_{b \to i}^{(\ell-1)}(1)}{\prod_{b \in C(i) \setminus a} M_{b \to i}^{(\ell-1)}(0)} = \prod_{b \in C(i) \setminus a} R_{b \to i}^{(\ell-1)} \tag{3.29}$$

and for $\mathbf{M}_{a \to i}$ message (using equation (3.23))

$$R_{a \to i}^{(\ell)} = \frac{\sum_{\mathbf{w}_{V(a) \setminus i}} \Big[\psi_a(1, \mathbf{w}_{V(a) \setminus i}, \mathbf{s}_a) \prod_{j \in V(a) \setminus i} M_{j \to a}^{(\ell)}(\mathbf{w}_j) \Big]}{\sum_{\mathbf{w}_{V(a) \setminus i}} \Big[\psi_a(0, \mathbf{w}_{V(a) \setminus i}, \mathbf{s}_a) \prod_{j \in V(a) \setminus i} M_{j \to a}^{(\ell)}(\mathbf{w}_j) \Big]}.$$

To calculate the last ratio, we will use the definition of function ψ_a (see 3.26) and partition the set $\mathbf{w}_{V(a)\setminus i} = W_{even} \cup W_{odd}$, where

$$W_{even} = \big\{\mathbf{x} \in \{0,1\}^{|V(a)|-1} \big|\ \#\text{ of 1 in } \mathbf{x} \text{ is even}\big\}$$
$$W_{odd} = \big\{\mathbf{x} \in \{0,1\}^{|V(a)|-1} \big|\ \#\text{ of 1 in } \mathbf{x} \text{ is odd}\big\}.$$

We now assume that $\mathbf{s}_a = 0$ and later remove this assumption. In this special case, we have $\psi_a(0, \mathbf{w}_{V(a)\setminus i}, \mathbf{s}_a) = e^{\gamma}$ for all vectors from W_{even} and $\psi_a(0, \mathbf{w}_{V(a)\setminus i}, \mathbf{s}_a) = e^{-\gamma}$ for all vectors from W_{odd}. Conversely, $\psi_a(1, \mathbf{w}_{V(a)\setminus i}, \mathbf{s}_a) = e^{-\gamma}$ for W_{even} and $\psi_a(1, \mathbf{w}_{V(a)\setminus i}, \mathbf{s}_a) = e^{\gamma}$ for W_{odd}. We can substitute and write

$$R^{(\ell)}_{a\to i} = \frac{e^{-\gamma}A + e^{\gamma}B}{e^{\gamma}A + e^{-\gamma}B},$$

where

$$A = \sum_{W_{even}} \prod_{j\in V(a)\setminus i} M^{(\ell)}_{j\to a}(\mathbf{w}_j) \qquad\qquad B = \sum_{W_{odd}} \prod_{j\in V(a)\setminus i} M^{(\ell)}_{j\to a}(\mathbf{w}_j).$$

We can express both sums (A and B) in terms of the ratios $R^{(\ell)}_{i\to a}$ by dividing them by the constant factor $\prod_{j\in V(a)\setminus i} M^{(\ell)}_{j\to a}(0)$

$$\hat{A} = \frac{A}{\prod_{j\in V(a)\setminus i} M^{(\ell)}_{j\to a}(0)} = \sum_{W_{even}} \prod_{j\in V(a)\setminus i} \frac{M^{(\ell)}_{j\to a}(\mathbf{w}_j)}{M^{(\ell)}_{j\to a}(0)} = \sum_{W_{even}} \prod_{j\in V(a)\setminus i} \Big(R^{(\ell)}_{j\to a}\Big)^{\mathbf{w}_j} =$$
$$= \frac{1}{2}\Big[\prod_{j\in V(a)\setminus i} \big(1 + R^{(\ell)}_{j\to a}\big) + \prod_{j\in V(a)\setminus i} \big(1 - R^{(\ell)}_{j\to a}\big)\Big]$$

and similarly for B

$$\hat{B} = \frac{B}{\prod_{j\in V(a)\setminus i} M^{(\ell)}_{j\to a}(0)} = \sum_{W_{odd}} \prod_{j\in V(a)\setminus i} \frac{M^{(\ell)}_{j\to a}(\mathbf{w}_j)}{M^{(\ell)}_{j\to a}(0)} = \sum_{W_{odd}} \prod_{j\in V(a)\setminus i} \Big(R^{(\ell)}_{j\to a}\Big)^{\mathbf{w}_j} =$$
$$= \frac{1}{2}\Big[\underbrace{\prod_{j\in V(a)\setminus i} \big(1 + R^{(\ell)}_{j\to a}\big)}_{=C} - \underbrace{\prod_{j\in V(a)\setminus i} \big(1 - R^{(\ell)}_{j\to a}\big)}_{=D}\Big].$$

Finally, we obtain

$$R^{(\ell)}_{a\to i} = \frac{e^{-\gamma}\hat{A} + e^{\gamma}\hat{B}}{e^{\gamma}\hat{A} + e^{-\gamma}\hat{B}} = \frac{e^{-\gamma}(C+D) + e^{\gamma}(C-D)}{e^{\gamma}(C+D) + e^{-\gamma}(C-D)} = \frac{(e^{\gamma}+e^{-\gamma})C - (e^{\gamma}-e^{-\gamma})D}{(e^{\gamma}+e^{-\gamma})C + (e^{\gamma}-e^{-\gamma})D} =$$
$$= \frac{1 - \frac{e^{\gamma}-e^{-\gamma}}{e^{\gamma}+e^{-\gamma}}\frac{D}{C}}{1 + \frac{e^{\gamma}-e^{-\gamma}}{e^{\gamma}+e^{-\gamma}}\frac{D}{C}} = \frac{1 - S^{(\ell)}_{a\to i}}{1 + S^{(\ell)}_{a\to i}}, \tag{3.30}$$

where we used the substitution

$$S^{(\ell)}_{a\to i} = (-1)^{\mathbf{s}_a}\Big(\frac{e^{\gamma} - e^{-\gamma}}{e^{\gamma} + e^{-\gamma}}\Big) \prod_{j\in V(a)\setminus i} \frac{1 - R^{(\ell)}_{j\to a}}{1 + R^{(\ell)}_{j\to a}}. \tag{3.31}$$

It is easy to see that the case where $\mathbf{s}_a = 1$ can be captured by the given substitution.

Using the substitution

$$B_{i\to a}^{(\ell)} = \frac{1 - R_{i\to a}^{(\ell)}}{1 + R_{i\to a}^{(\ell)}},$$

we can completely rewrite the message-passing rules (3.23), (3.22), and (3.24) only in terms of $B_{i\to a}^{(\ell)}$ and $S_{a\to i}^{(\ell)}$ obtaining thus our final message-passing rules.
From (3.30), we have

$$B_{i\to a}^{(\ell)} = \frac{1 - R_{i\to a}^{(\ell)}}{1 + R_{i\to a}^{(\ell)}} = \frac{1 - \prod_{b\in C(i)\setminus a} R_{b\to i}^{(\ell-1)}}{1 + \prod_{b\in C(i)\setminus a} R_{b\to i}^{(\ell-1)}} = \frac{1 - \prod_{b\in C(i)\setminus a} \frac{1-S_{b\to i}^{(\ell-1)}}{1+S_{b\to i}^{(\ell-1)}}}{1 + \prod_{b\in C(i)\setminus a} \frac{1-S_{b\to i}^{(\ell-1)}}{1+S_{b\to i}^{(\ell-1)}}} =$$

$$= \frac{\prod_{b\in C(i)\setminus a}\left[1 + S_{b\to i}^{(\ell-1)}\right] - \prod_{b\in C(i)\setminus a}\left[1 - S_{b\to i}^{(\ell-1)}\right]}{\prod_{b\in C(i)\setminus a}\left[1 + S_{b\to i}^{(\ell-1)}\right] + \prod_{b\in C(i)\setminus a}\left[1 - S_{b\to i}^{(\ell-1)}\right]}$$

and from (3.31) we have

$$S_{a\to i}^{(\ell)} = \prod_{j\in \overline{V}(a)\setminus i} B_{j\to a}^{(\ell)},$$

where the source message $B_{\mathbf{s}_a\to a}^{(\ell)}$ is defined as

$$B_{\mathbf{s}_a\to a}^{(\ell)} = (-1)^{\mathbf{s}_a}\left(\frac{e^{\gamma} - e^{-\gamma}}{e^{\gamma} + e^{-\gamma}}\right).$$

At the end, we compute the final bias using the fixed point messages $\hat{S}_{a\to i}$

$$B_i = \frac{P(0) - P(1)}{P(0) + P(1)} = \frac{1 - \frac{P(1)}{P(0)}}{1 + \frac{P(1)}{P(0)}} = \frac{1 - \prod_{b\in C(i)} \hat{R}_{b\to i}}{1 + \prod_{b\in C(i)} \hat{R}_{b\to i}} = \frac{\prod_{b\in C(i)}\left[1 + \hat{S}_{b\to i}\right] - \prod_{b\in C(i)}\left[1 - \hat{S}_{b\to i}\right]}{\prod_{b\in C(i)}\left[1 + \hat{S}_{b\to i}\right] + \prod_{b\in C(i)}\left[1 - \hat{S}_{b\to i}\right]}.$$

Thus, we just obtained equations (BiP-1), (BiP-2), (BiP-3), and (BiP-5) from Figure 3.7.

3.6.3 Dealing with cycles in a factor graph

The sum-product algorithm is exact (gives exact results) when the underlying graph structure is a tree. However, many researchers reported good results even for graphs with cycles. In principle, short cycles cause the messages to oscilate. Here, we discuss the damping procedure defined in Section 3.4 and its connection to the above formal derivation. We can think that this procedure is "damping" the oscilations. In Section 5.4, we will study the practical influence of damping.

Using the notation from the previous derivation and equation (3.29), we can write the update equation (BiP-2) as

$$\ln R_{i\to a}^{(\ell+1)} = \ln \prod_{b\in C(i)\setminus a} R_{b\to i}^{(\ell)} = \sum_{b\in C(i)\setminus a} \ln R_{b\to i}^{(\ell)}. \tag{3.32}$$

From this equation, we can think of the update rule (BiP-2) only as addition using $\ln R$. Thus, we can use arithmetic mean in this representation to calculate the output message in

the $\ell+1$-st iteration by averaging input messages from iteration ℓ and $\ell-1$. This will work as a "low-pass filter" and does not permit large changes to output messages. Using (3.32), we can write the output ratio after applying the damping procedure as

$$\ln R_{i\to a}^{(\ell+1)} = \frac{1}{2}\Big(\sum_{b\in C(i)\setminus a} \ln R_{b\to i}^{(\ell)} + \sum_{b\in C(i)\setminus a} \ln R_{b\to i}^{(\ell-1)}\Big), \tag{3.33}$$

and hence

$$R_{i\to a}^{(\ell+1)} = \Big(\prod_{b\in C(i)\setminus a} R_{b\to i}^{(\ell)} \cdot \prod_{b\in C(i)\setminus a} R_{b\to i}^{(\ell-1)}\Big)^{\frac{1}{2}}. \tag{3.34}$$

Using $B_{i\to a}^{(\ell)} = (1-R_{i\to a}^{(\ell)})/(1+R_{i\to a}^{(\ell)})$ and equation (3.30), we obtain

$$\begin{aligned} B_{i\to a}^{(\ell+1)} = \frac{1-R_{i\to a}^{(\ell+1)}}{1+R_{i\to a}^{(\ell+1)}} &\overset{(3.34)}{=} \frac{1-\Big(\prod_{b\in C(i)\setminus a} R_{b\to i}^{(\ell)} \cdot \prod_{b\in C(i)\setminus a} R_{b\to i}^{(\ell-1)}\Big)^{\frac{1}{2}}}{1+\Big(\prod_{b\in C(i)\setminus a} R_{b\to i}^{(\ell)} \cdot \prod_{b\in C(i)\setminus a} R_{b\to i}^{(\ell-1)}\Big)^{\frac{1}{2}}} \overset{(3.30)}{=} \\ &= \frac{\Big(A\cdot B\Big)^{\frac{1}{2}} - \Big(C\cdot D\Big)^{\frac{1}{2}}}{\Big(A\cdot B\Big)^{\frac{1}{2}} + \Big(C\cdot D\Big)^{\frac{1}{2}}}, \end{aligned} \tag{3.35}$$

where

$$A = \prod_{b\in C(i)\setminus a}\big(1+S_{b\to i}^{(\ell)}\big) = \frac{A+C}{2}\Big(1+B_{i\to a}^{(\ell+1)}\Big), \quad B = \prod_{b\in C(i)\setminus a}\big(1+S_{b\to i}^{(\ell-1)}\big) = \frac{B+D}{2}\Big(1+B_{i\to a}^{(\ell)}\Big),$$
$$C = \prod_{b\in C(i)\setminus a}\big(1-S_{b\to i}^{(\ell)}\big) = \frac{A+C}{2}\Big(1-B_{i\to a}^{(\ell+1)}\Big), \quad D = \prod_{b\in C(i)\setminus a}\big(1-S_{b\to i}^{(\ell-1)}\big) = \frac{B+D}{2}\Big(1-B_{i\to a}^{(\ell)}\Big).$$

Finally, term $\sqrt{(A+C)(B+D)/4}$ cancels from (3.35), and we obtain equation (BiP-4) that was used for damping.

3.7 Convergence analysis

While deriving the Bias Propagation algorithm, we hoped that for some information bit the magnitude of final bias $|B_i|$ at the end of each round is high, $|B_i| \approx 1$, and hence this bit has a high tendency to be fixed to some value without generating too much distortion. As we discussed in Section 3.4, this condition should be fulfiled in each round to obtain a small final distortion. From practical experiments, we see that there exist good degree distributions, which work well, and on the contrary there exist degree distributions that cannot be used with Bias Propagation, even if they are theoreticaly good for binary quantization. This is the case of regular degree distributions, which were studied by several research groups. Codes based on regular degree distributions cannot be used with Bias Propagation, because the bias magnitudes are very low in each round, resulting in very high distortion. This behavior was mentioned in the work of Wainwright and Maneva [24] and they left this question unanswered.

From this point of view, the Bias Propagation algorithm works only for a specific class of degree distributions, which may or may not contain codes that are theoreticaly good for

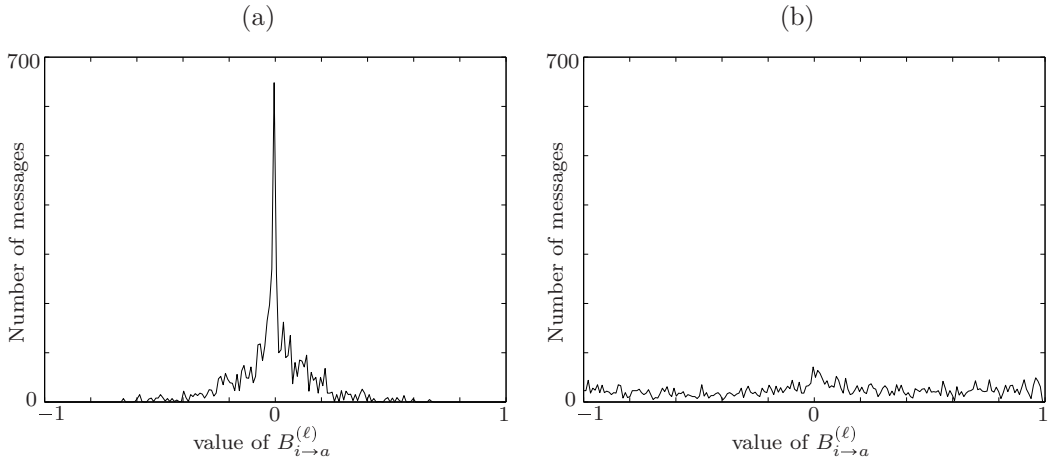


Figure 3.9: Histograms of $B_{i\to a}^{(\ell)}$ messages, where the maximum bias magnitudes (a) do not converge (b) converge to 1.

quantization. The question of bias magnitude convergence which defines this class is therefore important for further development and finally for finding good schemes for steganography. In this section, we derive this condition and practically describe the class of degree distributions that can be used with Bias Propagation. As a special case, we will show why we cannot use regular codes. The result of this section is Theorem 3.1 which states so-called *Convergence condition*.

Here, we will use the notation and study the behavior of the update equations derived in the previous section. The approach used in this section is known as *density evolution* and is used in modern coding theory. Good text to which we will sometimes refer is the book by Urbanke and Richardson [21]. The main idea behind this section is the following. Suppose that the messages $B_{i\to a}^{(\ell)}$ and $S_{a\to i}^{(\ell)}$ in ℓ-th iteration are random variables obtained from some known distribution, say $b^{(\ell)}$ and $s^{(\ell)}$, respectively. From previous sections, we have update equations for these random variables and we can further ask for update equations of the whole distributions $b^{(\ell)}$ and $s^{(\ell)}$. In other words, can we somehow calculate $b^{(\ell+1)}$ when we know $s^{(\ell)}$? Assume that we can do this and have a look at $b^{(\ell)}$. Now it is simple to describe the convergence, because we know whether there is some probability mass (in $b^{(\ell)}$) near 1 or -1 and hence whether there will be some message with a high bias magnitude. To demonstrate this idea, see Figure 3.9 where we plot histograms of bias messages ($b^{(\ell)}$) for two cases: (a) the algorithm does not converge in the round, the maximal bias magnitude is very small ($\max|B_i| \approx 0.5$), (b) algorithm converges and we are able to find highly biased information bits ($\max|B_i| \approx 1$). These distributions were obtained from practical simulations and they are histograms of all $B_{i\to a}^{(\ell)}$ messages in ℓ-th iteration in some round.

We start with some definitions we will use for describing probability distributions:

Definition 3.1 (D-domain, D-density) *We define* D-domain *as interval* $[-1,+1]$*, further we say that* function f is defined in D-domain*, when* $f:[-1,+1]\to[-1,+1]$*. Probability density function defined in D-domain is called* D-density*.*

According to this definition, we can say that all update equations for BiP algorithm are defined in D-domain. When the messages $B_{i\to a}^{(\ell)}$ and $S_{a\to i}^{(\ell)}$ are random variables, their densities $b^{(\ell)}$ and $s^{(\ell)}$ are D-densities. From Figure 3.9, we can see that the condition for convergence

of bias magnitudes can be expressed by calculating the variance of the random variable $B_{i\to a}^{(\ell)}$. The higher the variance, the better the convergence. Here, we are interested in finding some condition that we can calculate from a given degree distribution and predict the convergence of the algorithm in the current round. To do so, we say that the BiP algorithm converges in one round, if the sequence of variances calculated from the distribution $b^{(\ell)}$ converges to 1.

The reason for defining the D-domain is the following. All message update equations are defined in D-domain, but finding a compact formula for updating the whole D-densities (calculate $b^{(\ell+1)}$ directly from $s^{(\ell)}$ in D-domain) is not easy. To find a compact formula for updating these D-densities, we introduce another domain which will simplify the calculation.

Definition 3.2 (L-domain, L-density) *Let A be random variable defined in D-domain with D-density a. Let $l(x) = \ln \frac{1+x}{1-x}$ with inverse $l^{-1}(y) = \tanh(y/2)$ and let B be random variable obtained from this transformation, $B = l(A)$. Then, we say that B is defined in* L-domain*, which is $\overline{\mathbb{R}} = [-\infty, +\infty]$. Let b be pdf associated with random variable B. Every pdf defined in L-domain is called L-density. The L-density associated with random variable B is denoted b.*

From Section 3.6.2, we can see that by using $B_{i\to a}^{(\ell)} = (1 - R_{i\to a}^{(\ell)})/(1 + R_{i\to a}^{(\ell)})$ we can write

$$l\big(B_{i\to a}^{(\ell)}\big) = \ln \frac{1 + B_{i\to a}^{(\ell)}}{1 - B_{i\to a}^{(\ell)}} = -\ln R_{i\to a}^{(\ell)}, \tag{3.36}$$

and thus the transform update rule (BiP-2) from D-domain into L-domain using $\ln R_{a\to i}^{(\ell)} = -l(S_{a\to i}^{(\ell)})$ (see equations (3.29) and (3.30)) becomes

$$l\big(B_{i\to a}^{(\ell+1)}\big) = -\ln R_{i\to a}^{(\ell+1)} = \sum_{b\in C(i)\setminus a} -l\big(S_{a\to i}^{(\ell)}\big) = \sum_{b\in C(i)\setminus a} l\big(S_{a\to i}^{(\ell)}\big). \tag{3.37}$$

We can see that the update equation for calculating $B_{i\to a}^{(\ell+1)}$ is a simple addition when we transform all messages into L-domain using function l. Using this transformation, we can obtain update equations for whole densities as follows. Let a and b be two D-densities and let a and b be L-densities obtained by transforming a and b from D-domain into L-domain. Because the update equation for random variables (messages) in L-domain is a simple addition, the L-density associated to the random variable obtained by this update rule is a convolution of underlying densities. Here, we assume that we have independent random variables, which is true for cycle free graphs. Convolution of two L-densities a and b is also L-density (say c) and we write $\mathsf{c} = \mathsf{a} \circledast \mathsf{b}$. Convolution $\circledast$ is defined in L-domain, which is $[-\infty, +\infty]$ (for proof of correctness of the convolution see §4.1.4 in [21]). For derivation of complete update equation we write $\mathsf{a} \circledast \mathsf{b}$ not only for L-densities, but we extend this convolution to D-domain and write $a \circledast b$ even for D-densities, where we first transform both densities a and b into L-domain, perform convolution, and transform the resulting L-density back to D-domain.

As an example, assume we have a graph with the distribution of information bit nodes from edge perspective $\lambda(x) = 0.2x^2 + 0.8x^3$. It means that 20% of all edges are connected to information bits with degree 3 and 80% of all edges are connected to information bits with degree 4 (see Section 3.2). Let $s^{(\ell)}$ be the D-density of $S_{a\to i}^{(\ell)}$ messages. Then the pdf of

$B_{i\to a}^{(\ell+1)}$ messages $b^{(\ell+1)}$ can be obtained as

$$b^{(\ell+1)} = 0.2\big(s^{(\ell)} \circledast s^{(\ell)}\big) + 0.8\big(s^{(\ell)} \circledast s^{(\ell)} \circledast s^{(\ell)}\big), \tag{3.38}$$

because 20% of $B_{i\to a}^{(\ell+1)}$ messages are obtained by adding 2 randomly chosen $S_{a\to i}^{(\ell)}$ messages in L-domain and 80% are obtained by adding 3 randomly chosen messages in L-domain. As we can see from this example, degree distributions from the edge perspective are important for these calculations. Thus, we extend our notation and for D-density a define

$$\lambda(a) = \lambda_1\delta_0 + \lambda_2 a + \lambda_3 a^{\circledast\, 2} + \cdots + \lambda_{d_L} a^{\circledast\, d_L - 1}, \tag{3.39}$$

where δ_0 is Dirac delta function and $a^{\circledast\, k} = \underbrace{a \circledast a \circledast \cdots \circledast a}_{k-\text{times}}$. Using this notation, we can write

$$b^{(\ell+1)} = \lambda(s^{(\ell)}). \tag{3.40}$$

We now turn our attention to check nodes and find a similar equation for calculating $s^{(\ell)}$ from $b^{(\ell)}$. To be able to calculate $S_{a\to i}^{(\ell)}$ messages using $B_{i\to a}^{(\ell)}$ messages as addition, we introduce another domain as follows.

Definition 3.3 (G-domain, G-density) *Let A be random variable defined in D-domain with D-density a. Let*

$$\mathfrak{h}(x) = \begin{cases} 0 & \text{if } x > 0 \\ 0 & \text{with probability } \frac{1}{2} \text{ if } x = 0 \\ +1 & \text{with probability } \frac{1}{2} \text{ if } x = 0 \\ +1 & \text{if } x < 0, \end{cases}$$

and $g(x) = \big(\mathfrak{h}(x), -\ln|x|\big) = (y_1, y_2)$ with inverse $g^{-1}(y) = (-1)^{y_1} e^{-y_2}$. Let B be random variable obtained from this transformation, $B = g(A)$. Then, we say that B is defined in G-domain*, which is $\mathbb{F}_2 \times [0, +\infty]$. Let $\mathfrak{b}$ be pdf associated with random variable B. Every pdf defined in G-domain is called G-density. The G-density associated with random variable B is denoted $\mathfrak{b}$.*

From this definition, we obtain

$$g\big(S_{a\to i}^{(\ell)}\big) = \sum_{j\in\overline{V}(a)\setminus i} g\big(B_{i\to a}^{(\ell)}\big), \tag{3.41}$$

where the addition is elementwise in $\mathbb{F}_2 \times [0, +\infty]$. Since the G-domain is $\mathbb{F}_2 \times [0, +\infty]$, every G-density $\mathfrak{a}$ can be decomposed as

$$\mathfrak{a}(s, x) = \mathbb{1}_{s=0}\mathfrak{a}(0, x) + \mathbb{1}_{s=1}\mathfrak{a}(1, x),$$

where $\mathbb{1}_{s=0}$ is an indicator function for event $s = 0$ and $\mathfrak{a}(0,\cdot), \mathfrak{a}(1,\cdot) : [0, +\infty] \to [0, +\infty]$. Here, we will use the same idea for calculating D-density $s^{(\ell)}$ from $b^{(\ell)}$. Thus, for two G-densities $\mathfrak{a}$ and $\mathfrak{b}$ we introduce the convolution $\mathfrak{a} \boxed{\circledast} \mathfrak{b}$ as a convolution over $\mathbb{F}_2 \times [0, +\infty]$ (for proof of correctness of this convolution see §4.1.4 in [21]). Again, we assume that we have cycle free graph, thus the random variables are independent. The final distribution is also

G-density (say $\mathfrak{c}$) and can be written in the following form

$$\mathfrak{c}(s,x) = \mathbb{1}_{s=0}\Big(\mathfrak{a}(0,x) \star \mathfrak{b}(0,x) + \mathfrak{a}(1,x) \star \mathfrak{b}(1,x)\Big) + \mathbb{1}_{s=1}\Big(\mathfrak{a}(1,x) \star \mathfrak{b}(0,x) + \mathfrak{a}(0,x) \star \mathfrak{b}(1,x)\Big), \tag{3.42}$$

where $\star$ is the convolution of standard distributions. Again, we extend our notation and for D-densities a and b we calculate $a \boxed{\circledast} b$ first by transforming both D-densities to G-domain, apply $\boxed{\circledast}$ and transform the resulting G-density back to D-domain. Similarly, for some D-density a we define

$$\rho(a) = \rho_1\delta_0 + \rho_2 a + \rho_3 a^{\boxed{\circledast}\,2} + \cdots + \rho_{d_R} a^{\boxed{\circledast}\,d_R - 1}, \tag{3.43}$$

where $a^{\boxed{\circledast}\,k} = \underbrace{a \boxed{\circledast} a \boxed{\circledast} \cdots \boxed{\circledast} a}_{k-\text{times}}$.

Using the same approach as we discussed in the example above and using this notation, we can write the update rule for calculating $s^{(\ell)}$ from $b^{(\ell)}$ as

$$s^{(\ell)} = b_{\mathbf{s}} \boxed{\circledast} \rho(b^{(\ell)}), \tag{3.44}$$

where $b_{\mathbf{s}}$ is D-density of $B_{\mathbf{s}_a \to a}$ (source) messages.

To study the convergence, we are interested in statistical moments of D-densities. Hence, we give the following definition.

Definition 3.4 (D-k-moment) *Let a be the D-density, for $k = 1, 2, \ldots$ we define* D-k-moment *of distribution a as*

$$\mathfrak{D}_k(a) = \int_{-1}^{+1} a(x) x^k \, dx. \tag{3.45}$$

The first statistical moment is called D-mean*, the second moment is called* D-variance.

Next, we define what we mean by symmetry in the case of D-densities.

Definition 3.5 (D-symmetry) *We say that the D-density a is* D-symmetric*, if $a(x) = a(-x)$ for all $x \in [-1, +1]$.*

We note that in modern coding theory, the symmetry of a distribution is defined in a different manner, and is called exponential symmetry (see definition 4.11 in [21]). However, in binary quantization problems we cannot prove this symmetry for our distributions. We will show that our distributions are D-symmetric and that the D-symmetry is necessary to prove the convergence condition. For further work, the following lemmas will be usefull. They show that the D-k-moment is multiplicative under $\boxed{\circledast}$ convolution and that using this result we can find tight bounds on D-k-moments under $\circledast$ convolution.

Lemma 3.1 (Multiplicativity of $\mathfrak{D}_k$ under $\boxed{\circledast}$ convolution) *Let a and b be D-densities. Then, for every $k = 1, 2, \ldots$ we have*

$$\mathfrak{D}_k(a \boxed{\circledast} b) = \mathfrak{D}_k(a)\mathfrak{D}_k(b).$$

Proof: Let a and b be D-densities and let $\mathfrak{a}$ and $\mathfrak{b}$ be the associated G-densities. First, we show how to calculate $\mathfrak{D}_k(\mathfrak{a})$. We use the transformation $y = -\ln(|x|)$ and substitute into equation (3.45) and obtain

$$\mathfrak{D}_k(a) = \int_{-1}^{0} a(x)x^k\,dx + \int_{0}^{+1} a(x)x^k\,dx = \underbrace{(-1)^k \int_0^{+\infty} \mathfrak{a}(1,y)e^{-ky}\,dy}_{D_k^1(\mathfrak{a})} + \underbrace{\int_0^{+\infty} \mathfrak{a}(0,y)e^{-ky}\,dy}_{D_k^0(\mathfrak{a})} =$$

$$= D_k^0(\mathfrak{a}) + D_k^1(\mathfrak{a}) = \mathfrak{D}_k(\mathfrak{a}). \tag{3.46}$$

It is sufficient to show that $\mathfrak{D}_k(\mathfrak{a} \boxtimes \mathfrak{b}) = \mathfrak{D}_k(\mathfrak{a})\mathfrak{D}_k(\mathfrak{b})$. We can write

$$\begin{aligned}\mathfrak{D}_k(\mathfrak{a})\mathfrak{D}_k(\mathfrak{b}) &= \big(D_k^0(\mathfrak{a}) + D_k^1(\mathfrak{a})\big)\big(D_k^0(\mathfrak{b}) + D_k^1(\mathfrak{b})\big) = \\ &= D_k^0(\mathfrak{a})D_k^0(\mathfrak{b}) + D_k^1(\mathfrak{a})D_k^1(\mathfrak{b}) + D_k^1(\mathfrak{a})D_k^0(\mathfrak{b}) + D_k^0(\mathfrak{a})D_k^1(\mathfrak{b})\end{aligned} \tag{3.47}$$

To finish this proof, we show that $D_k^0(\mathfrak{a})D_k^0(\mathfrak{b})$ is equal to D-k-moment of the first term in 3.42. We write

$$\begin{aligned}D_k^0(\mathfrak{a})D_k^0(\mathfrak{b}) &= \int_0^{+\infty} \mathfrak{a}(0,x)e^{-kx}\,dx \cdot \int_0^{+\infty} \mathfrak{b}(0,z)e^{-kz}\,dz = \\ &= \int_0^{+\infty}\int_0^{+\infty} \mathfrak{a}(0,x)\mathfrak{b}(0,z)e^{-k(x+z)}\,dx\,dz = \\ &= \int_0^{+\infty}\int_0^{+\infty} \mathfrak{a}(0,x)\mathfrak{b}(0,y-x)\,dx\,e^{-ky}\,dy = \int_0^{+\infty} \big(\mathfrak{a}(0,\cdot) \star \mathfrak{b}(0,\cdot)\big)(y)e^{-ky}\,dy,\end{aligned}$$

where on the third line we used substitution $y = x + z$ and define $\mathfrak{b}(0,x) = 0$ for $x < 0$. For $D_k^1(\mathfrak{a})D_k^1(\mathfrak{b})$, we have

$$\begin{aligned}D_k^1(\mathfrak{a})D_k^1(\mathfrak{b}) &= (-1)^k \int_0^{+\infty} \mathfrak{a}(1,x)e^{-kx}\,dx \cdot (-1)^k \int_0^{+\infty} \mathfrak{b}(1,z)e^{-kz}\,dz = \\ &= \int_0^{+\infty}\int_0^{+\infty} \mathfrak{a}(1,x)\mathfrak{b}(1,z)e^{-k(x+z)}\,dx\,dz = \int_0^{+\infty} \big(\mathfrak{a}(1,\cdot) \star \mathfrak{b}(1,\cdot)\big)(y)e^{-ky}\,dy.\end{aligned}$$

The proof for terms $D_k^0(\mathfrak{a})D_k^1(\mathfrak{b})$, $D_k^1(\mathfrak{a})D_k^0(\mathfrak{b})$ is similar. Finally, we can write

$$\begin{aligned}D_k^0(\mathfrak{a} \boxtimes \mathfrak{b}) &= D_k^0(\mathfrak{a})D_k^0(\mathfrak{b}) + D_k^1(\mathfrak{a})D_k^1(\mathfrak{b}) \\ D_k^1(\mathfrak{a} \boxtimes \mathfrak{b}) &= D_k^1(\mathfrak{a})D_k^0(\mathfrak{b}) + D_k^0(\mathfrak{a})D_k^1(\mathfrak{b}).\end{aligned}$$

□

As a corollary (using 3.44), we obtain

$$\mathfrak{D}_k\big(s^{(\ell)}\big) = \mathfrak{D}_k(b_{\mathbf{s}})\mathfrak{D}_k\big(\rho(b^{(\ell)})\big) = \mathfrak{D}_k(b_{\mathbf{s}})\rho\big(\mathfrak{D}_k(b^{(\ell)})\big). \tag{3.48}$$

This is an important result, because now we know what the behavior of $\mathfrak{D}_k$ is when the densities go through the check nodes. We would like to write a similar equation for $\circledast$ in information bits. Hence, we derive the following lemma.

Lemma 3.2 (Bounds on $\mathfrak{D}_k(a \circledast b)$) *Assume two D-symmetric D-densities a and b, then*

$$\mathfrak{D}_{2k+1}(a \circledast b) = 1 - \big(1 - \mathfrak{D}_{2k+1}(a)\big)\big(1 - \mathfrak{D}_{2k+1}(b)\big) \qquad \forall k, \tag{3.49}$$

$$\mathfrak{D}_2(a \circledast b) \leqslant 1 - \big(1 - \mathfrak{D}_2(a)\big)\big(1 - \mathfrak{D}_2(b)\big). \tag{3.50}$$

Proof: Let a and b be D-symmetric D-densities and let a and b be their associated L-densities. L-density associated with D-symmetric D-density is also symmetric ($a(x) = a(-x) \Leftrightarrow \mathsf{a}(x) = \mathsf{a}(-x)$). Using the substitution $x = \tanh(y/2)$, we can calculate

$$\mathfrak{D}_k(a) = \int_{-1}^{+1} a(x)x^k\,dx = \int_{-\infty}^{+\infty} \mathsf{a}(y)\tanh^k(y/2)\,dy. \tag{3.51}$$

When we use this result, we can write

$$\begin{aligned} 1 - \mathfrak{D}_k(a \circledast b) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathsf{a}(x)\mathsf{b}(y-x)\big(1-\tanh^k(y/2)\big)\,dx\,dy = \\ &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathsf{a}(x)\mathsf{b}(z)\big(1-\tanh^k((x+z)/2)\big)\,dx\,dy = \\ &= \int_{0}^{+\infty}\int_{0}^{+\infty} \mathsf{a}(x)\mathsf{b}(z)f_k(x,z)\,dx\,dy, \end{aligned} \tag{3.52}$$

where we used the symmetry of a and b and

$$\begin{aligned} f_k(x,z) = {} & \big(1-\tanh^k((x+z)/2)\big) + \big(1-\tanh^k((x-z)/2)\big) + \\ & + \big(1-\tanh^k((-x+z)/2)\big) + \big(1-\tanh^k((-x-z)/2)\big). \end{aligned} \tag{3.53}$$

For odd indices, we obtain $f_{2k+1}(x,z) = 4$.

$$\begin{aligned} \big(1-\mathfrak{D}_k(a)\big)\big(1-\mathfrak{D}_k(b)\big) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathsf{a}(x)\mathsf{b}(z)\big(1-\tanh^k(x/2)\big)\big(1-\tanh^k(z/2)\big)\,dx\,dz = \\ &= \int_{0}^{+\infty}\int_{0}^{+\infty} \mathsf{a}(x)\mathsf{b}(z)g_k(x,z)\,dx\,dz, \end{aligned} \tag{3.54}$$

where we used the symmetry of a and b and

$$\begin{aligned} g_k(x,z) = {} & \big(1-\tanh^k(x/2)\big)\big(1-\tanh^k(z/2)\big) + \big(1-\tanh^k(x/2)\big)\big(1-\tanh^k(-z/2)\big) + \\ & + \big(1-\tanh^k(-x/2)\big)\big(1-\tanh^k(z/2)\big) + \big(1-\tanh^k(-x/2)\big)\big(1-\tanh^k(-z/2)\big). \end{aligned} \tag{3.55}$$

For odd indices, we obtain $g_{2k+1}(x,z) = 4$ and hence (3.52) equals to (3.54) and we obtain equation (3.49). We now show that $f_2(x,z) - g_2(x,z) \geqslant 0$ when $x, z \geqslant 0$. Using this inequality we have $f_2(x,z) \geqslant g_2(x,z)$ which proves equation (3.50). To proof the inequality, we write

$$f_2(x,z) - g_2(x,z) = 2\Big[1-\tanh^2\frac{x+z}{2} + 1 - \tanh^2\frac{x-z}{2} - 2\Big(1-\tanh^2\frac{x}{2}\Big)\Big(1-\tanh^2\frac{z}{2}\Big)\Big],$$

where

$$\begin{aligned} 1-\tanh^2\frac{x+z}{2} &= 1 - \left(\frac{\tanh(x/2)+\tanh(z/2)}{1+\tanh(x/2)\tanh(z/2)}\right)^2 = \frac{\big(1-\tanh^2(x/2)\big)\big(1-\tanh^2(z/2)\big)}{\big(1+\tanh(x/2)\tanh(z/2)\big)^2} \\ 1-\tanh^2\frac{x-z}{2} &= 1 - \left(\frac{\tanh(x/2)-\tanh(z/2)}{1-\tanh(x/2)\tanh(z/2)}\right)^2 = \frac{\big(1-\tanh^2(x/2)\big)\big(1-\tanh^2(z/2)\big)}{\big(1-\tanh(x/2)\tanh(z/2)\big)^2}. \end{aligned}$$

We use $X = \tanh(x/2) \in [0,1)$ and $Z = \tanh(z/2) \in [0,1)$ to shorten the notation. We get

$$f_2(x,y) - g_2(x,y) = 2\underbrace{\big(1-X^2\big)}_{\geqslant 0}\underbrace{\big(1-Z^2\big)}_{\geqslant 0}\underbrace{\left[\frac{1}{(1+XZ)^2}+\frac{1}{(1-XZ)^2}-2\right]}_{A}.$$

To complete the proof, we show that the last term fulfills $A \geqslant 0$. For $X, Y \in [0,1)$, we obtain

$$\begin{aligned} A &= \frac{(1-XZ)^2+(1+XZ)^2-2(1+XZ)^2(1-XZ)^2}{(1+XZ)^2(1-XZ)^2} = \\ &= 2\frac{3X^2Z^2-X^4Z^4}{(1+XZ)^2(1-XZ)^2} \geqslant 2\frac{2X^4Z^4}{(1+XZ)^2(1-XZ)^2} \geqslant 0. \end{aligned}$$

□

As a corollary, for D-symmetric D-density a we obtain the following equality

$$\mathfrak{D}_{2k+1}(\lambda(a)) = 1 - \lambda\big(1-\mathfrak{D}_{2k+1}(a)\big) \tag{3.56}$$

and the bound

$$\mathfrak{D}_{2}(\lambda(a)) \leqslant 1 - \lambda\big(1-\mathfrak{D}_{2}(a)\big). \tag{3.57}$$

For proving convergence, we will need the following simple lemma.

Lemma 3.3 *Let $\big(x_\ell\big)_{l=1}^{\infty}$ be a sequence of real numbers defined as $x_{\ell+1} = a + bx_\ell$, where $0 \leqslant b < 1$ and $a < 1-b$. Then, $\lim_{\ell\to+\infty} x_\ell = x$ exists and $x < 1$.*

Proof: If $x_\ell < 1$, then $x_{\ell+1} = a + bx_\ell < a + b < 1$, thus the sequence is bounded. Next, $x_{\ell+1} - x_\ell = a + bx_\ell - x_\ell = a + bx_\ell - a - bx_{\ell-1} = b(x_\ell - x_{\ell-1})$, hence the limit x exists, because the sequence is monotonic and bounded. The limit x should satisfy $x = a + bx$ and hence $x = \frac{a}{1-b} < 1$. □

Using the equations we derived, it is easy to show the following convergence condition.

Theorem 3.1 (Convergence condition) *Let $b_{\mathbf{s}}$ be D-symmetric D-density of source messages $B_{\mathbf{s}_a\to a}$ and let (ρ,λ) be degree distribution from edge perspective in r-th round. If the BiP algorithm converges in this round ($\mathfrak{D}_2(b^{(\ell)}) \to 1$ when $\ell \to +\infty$), then the degree distribution has to fulfil the following necessary condition:*

$$\mathfrak{C}(b_{\mathbf{s}},\rho,\lambda) = \mathfrak{D}_2(b_{\mathbf{s}})\rho'(0)\frac{\lambda'\big(1-\mathfrak{D}_2(b_{\mathbf{s}})\rho(0)\big)}{\lambda\big(1-\mathfrak{D}_2(b_{\mathbf{s}})\rho(0)\big)} \geqslant 1 \tag{3.58}$$

and all D-densities $s^{(\ell)}$ and $b^{(\ell)}$ exchanged in each iteration ℓ are D-symmetric ($\mathfrak{D}_1(s^{(\ell)}) = \mathfrak{D}_1(b^{(\ell)}) = 0$).

Proof: From the D-symmetry of density $b_{\mathbf{s}}$ (it follows that $\mathfrak{D}_{2k+1}(b_{\mathbf{s}}) = 0$) and using (3.48) and (3.49) we obtain $\mathfrak{D}_{2k+1}(b^{(\ell)}) = 0$ and $\mathfrak{D}_{2k+1}(s^{(\ell)}) = 0$. All densities exchanged in each iteration are D-symmetric, because their moment-generating functions are even. Using our notation we have $b^{(\ell+1)} = \lambda(s^{(\ell)})$. Denoting $x_\ell = \mathfrak{D}_2(b^{(\ell)})$, $y_\ell = \mathfrak{D}_2(s^{(\ell)})$, $z = \mathfrak{D}_2(b_{\mathbf{s}})$ and substituting (3.48) into (3.57) where $a = s^\ell$, we obtain

$$x_{\ell+1} \leqslant 1 - \lambda\big(1-z\rho(x_\ell)\big) = a + bx_\ell + \mathcal{O}(x_\ell^2), \tag{3.59}$$

where $a = 1 - \lambda\big(1 - z\rho(0)\big) = 1 - \lambda(1 - z\rho_1)$, $b = \big(1 - \lambda\big(1 - z\rho(x_\ell)\big)\big)'\big|_{x_\ell=0} = \lambda'(1 - z\rho_1)z\rho_2$. We prove the necessity of (3.58) by contradiction. Suppose that

$$\mathfrak{D}_2(b_{\mathbf{s}})\rho'(0)\frac{\lambda'\big(1 - \mathfrak{D}_2(b_{\mathbf{s}})\rho(0)\big)}{\lambda\big(1 - \mathfrak{D}_2(b_{\mathbf{s}})\rho(0)\big)} < 1, \tag{3.60}$$

and that the BiP algorithm converges in the r-th round. From (3.59), we have that (3.60) is equivalent to $a < 1 - b$. Because $a > 0$, $0 \leqslant b < 1$. From Lemma 3.3 using $a < 1 - b$, we obtain the contradiction, because the sequence of D-variances x_ℓ cannot converge to 1, thus (3.58) must be true. □

Using this necessary condition, we can show that regular codes cannot converge in the first round of the BiP algorithm. From (3.58), we can see that the convergence depends on the number of edges connected to check nodes with degree two ($\rho'(0) = \rho_2$). When $\rho_2 = 0$, then $\mathfrak{C}(b_{\mathbf{s}}, \rho, \lambda)$ is always zero and the algorithm will not converge in the first round.

3.7.1 Evolution over rounds

The convergence condition gives us some information about the behavior of the algorithm only in one specific round, because the degree distribution pair is changed by decimation. To be able to study the convergence behavior in each round, we have to know how the degree distribution will change after one decimation step.

Let $\mathbf{G}$ be a sparse LDGM matrix associated with (n, k) code $\mathcal{C}$ generated using the initial degree distribution (ρ, λ). Define the discrete time (denoted as t) as the number of information bits that was not removed by decimation. The time starts at $t = k$ and ends when $t = 0$ and is decremented by one per removed information bit. For a finite length code ($n < +\infty$), we define the unnormalized degree distribution at time t $\big(R_i(t), L_i(t)\big)$ as the number of edges in the graph connected to check nodes (R_i) or information bits (L_i) with degree i. We are interested in these functions as functions of t. Because the initial matrix $\mathbf{G}$ was generated randomly, we can calculate the average change of the unnormalized degree distribution pair when removing one information bit (performing one decimation step) as

$$\mathbb{E}\big[L_i(t) - L_i(t-1)\big|L_i(t), R_i(t)\big] = \frac{L_i(t)}{t} \tag{3.61}$$

$$\mathbb{E}\big[R_i(t) - R_i(t-1)\big|L_i(t), R_i(t)\big] = \big(R_i(t) - R_{i+1}(t)\big)\frac{i}{t}, \qquad i < d_R \tag{3.62}$$

$$\mathbb{E}\big[R_{d_R}(t) - R_{d_R}(t-1)\big|L_{d_R}(t), R_{d_R}(t)\big] = R_{d_R}(t)\frac{d_R}{t}. \tag{3.63}$$

The first observation is that on average by removing one information bit we do not change the degree distribution of information bits from the nodes perspective. We say that the edge has check, info bit degree equal to i if it is connected to check, info bit with degree i, respectively. To obtain (3.61), calculate the expected number of edges having check degree i removed with one information bit. This is equal to $i\lambda_i^N = i\frac{L_i(t)/i}{t} = L_i(t)/t$. When we remove one edge having check degree i, we subtract i from R_i and add $i - 1$ to R_{i-1}. The first term in (3.62) is obtained as the number of edges removed from $R_i(t)$ when removing one information bit. This can be writen as

$$i\frac{R_i(t)}{\sum_{i=1}^{d_R} R_i(t)}\overline{\lambda^N} = i\frac{R_i(t)}{t\sum_{i=1}^{d_R} i\lambda_i^N}\sum_{i=1}^{d_R} i\lambda_i^N = i\frac{R_i(t)}{t},$$

where $R_i(t)/\sum_{i=1}^{d_R} R_i(t)$ is the probability of removing an edge with check degree i. By removing one info bit, we remove $\overline{\lambda^N}$ on average. The second term in (3.62) is obtain similarly. Finally, we obtain (3.63) as a special case of the previous equation, where $R_{d_R+1}(t) = 0$ $\forall t = k, k-1, \ldots, 0$.

We can assume that the values of the unnormalized degree distributions $\big(R_i(t), L_i(t)\big)$ are close to their means and remove the expected values. This can be formally proved using the Wormald method, see Theorem C.28 in [21]. When we introduce the scaled-time $\tau = t/k$ and consider the limit $n \to +\infty$ while $R = \frac{k}{n}$ stays constant, we can write the set of first-order differential equations

$$\frac{d}{d\tau} l_i(\tau) = \frac{l_i(\tau)}{\tau} \tag{3.64}$$

$$\frac{d}{d\tau} r_i(\tau) = \big(r_i(\tau) - r_{i+1}(\tau)\big)\frac{i}{\tau}, \qquad i < d_R \tag{3.65}$$

$$\frac{d}{d\tau} r_{d_R}(\tau) = r_{d_R}(\tau)\frac{d_R}{\tau}, \tag{3.66}$$

where we used $l_i(\tau) \approx L_i(k\tau)/k$. Up to a multiplicative constant, we interpret the fraction $\frac{l_i(\tau)}{\tau}$ as the (normalized) coefficient of degree distribution from edge perspective λ_i at time τ. Similarly, we do the same with $\frac{r_i(\tau)}{\tau}$ for ρ_i. This set of equations has the following solution

$$l_i(\tau) = \upsilon_i \tau \tag{3.67}$$

$$r_i(\tau) = \sum_{j=i}^{d_R} \kappa_{i,j} \tau^{d_R}, \tag{3.68}$$

where $\upsilon_i = \lambda_i$ is specified from the initial degree distribution (ρ, λ). The matrix $\kappa \in \mathcal{R}^{d_R \times d_R}$ is defined usign the recursive formula: $\kappa_{i,j} = -\frac{i\kappa_{i+1,j}}{j-i}$ for $j > i$ and $\kappa_{i,i} = \rho_i - \sum_{j=i+1}^{d_R} \kappa_{i,j}$. For $\tau = 1$, we have $r_i(1) = \rho_i$.

Although we can write the solution in an analytic form, it is not possible to use it for practical calculations. This is because the coefficients of the polynomial $r_i(\tau)$ are very large and have different signs. A better way of obtaining the solution is to use equations (3.61)–(3.63) and calculate $R_i(t-1)$ from $R_i(t)$. Both $R_i(k)$ and $L_i(k)$ are obtained from the initial degree distribution from the edge perspective (ρ, λ).

3.8 Survey Propagation based quantizer

In this section, we will describe the SP based quantizer proposed by Wainwright and Maneva [24]. We have already discussed their approach in the previous section as a part of our recent work review. As we know, they proposed the first solution to MAX-XOR-SAT problem using message-passing algorithms. Here, we will describe the approach without derivations and in the next section we will show that BiP is a simpler special case of the SP based algorithm. The usage of SP based algorithms for constructing near-optimal steganographic schemes was studied in our previous work [9].

We start the description of the main idea of this algorithm by defining so-called *extended codeword* $(\mathbf{c}, \mathbf{w}) \in \{0,1\}^{2n-m}$ as a concatenation of two vectors that satisfy the equation $\mathbf{c} = \mathbf{G}\mathbf{w}$. Next, we extend the set of all possible values for each bit to $\{0, 1, *\}$ and create so-called set of *generalized codewords*. Each extended codeword is by a definition a generalized

```
procedure w = SP(G, z)
  while not all_bits_fixed(w)
    bias = SP_iter(z, G)
    bias = sort(bias)
    if max(bias)>t
      num = min(num_max, num_of_bits(bias>t))
    else
      num = num_min
    [G,z,w] = dec_most_biased_bits(G,z,w,num)
  end
end
```

```
procedure bias = SP_iter(z, G)
  M_zaa = normalize(calc_source_message(z))
  M_ai = send_src_message(G, M_zaa)
  while |M_ai_old-M_ai|<e OR iter<max_iter
    M_ai_old = M_ai
    M_ia = normalize(calc_ia(M_ai))
    M_ai = normalize(calc_ai(M_ia, M_zaa))
    if iter>start_damp then M_ai = normalize(damp(M_ai))
    iter = iter+1
  end
  bias = calc_bias(M_ai)
end
```

Figure 3.10: Pseudocode for the SP based quantization algorithm. This code is discussed in Section 3.8.

codeword. An arbitrary vector that contains some $*$ symbols is generalized codeword if each check node with only $\{0,1\}$ assignment is satisfied. To be able to capture the structure of the space of all codewords and to be able to quantize to the nearest codeword, we define the following weighted probabilistic distribution over the space of all generalized codewords:

$$P(\mathbf{c},\mathbf{w};w_{sou},w_{info},\gamma) = w_{sou}^{n_*^{sou}(\mathbf{c})} \times w_{info}^{n_*^{info}(\mathbf{w})} \times e^{-2\gamma d_H(\mathbf{s},\mathbf{c})}, \tag{3.69}$$

where $w_{sou}, w_{info}, \gamma$ are constant parameters and $n_*^{sou}(c)$, $n_*^{info}(w)$ represent the number of $*$ in vector $\mathbf{c}$, $\mathbf{w}$, respectively. The last factor expresses the fact that the probability of seeing vector $(\mathbf{c},\mathbf{w})$ depends on the Hamming distance between the given (constant) source sequence and the codeword $\mathbf{c}$. This dependence is further influenced by the constant parameter γ. According to [24], the $*$ weighting factors w_{sou}, w_{info} were typically set to $w_{sou} = 1.1$ and $w_{info} = 1.0$. However, when we set both parameters to zero we will obtain the weighted distribution over ordinary codewords (we stop counting the $*$ values and therefore we do not need them).

Using the probability distribution we just described, we can calculate the marginal probability for each bit and fix the bit that has the largest probability to be 0 or 1. Wainwright and Maneva used the sum-product algorithm to derive compact update equations (see Figure 3.11).

Because SP based quantization algorithm has a similar structure as the Bias Propagation algorithm, we will use the terminology developed in previous sections to describe this work. Figure 3.10 shows the pseudo-code for the whole algorithm and it defines two procedures `SP()` and `SP_iter()` that implement the message-passing iterations.

The SP algorithm (`SP()` function) starts its first round with a graph $\mathbb{G}^{(1)}$ representing the factor graph of the linear code with generator matrix $\mathbf{G}$ and a vector of source bits $\mathbf{s}^{(1)} = \mathbf{s}$. Using these parameters, we run `SP_iter()` to calculate the bias $B_i = |\mu_i(1) - \mu_i(0)|$ for each free information bit (in the beginning, all information bits are free). The bias B_i expresses the tendency of each free info bit to be set to a specific value. In the next step, we use this information to sort the free info bits according to their bias and we select `num` most biased info bits to be set by decimation in this round. Here, we use the same decimation strategy as was described for the Bias Propagation algorithm: set `num` to the number of free info bits with $B_i >$ `t` (constant threshold), but no more than `num_max`. If there are no $B_i >$ `t`, set `num` to some small constant `num_min`. The final step is the decimation function `dec_most_biased_bits()`. The values of the constants `num, num_max, num_min` will be discussed in Chapter 5.

Bits to checks update rules

$$
\begin{aligned}
M_{i\to a}^{0f\,(\ell)} &= \prod_{b\in C(i)\setminus\{a\}} \left[M_{b\to i}^{0f\,(\ell-1)} + M_{b\to i}^{0w\,(\ell-1)}\right] - \prod_{b\in C(i)\setminus\{a\}} M_{b\to i}^{0w\,(\ell-1)} \\
M_{i\to a}^{1f\,(\ell)} &= \prod_{b\in C(i)\setminus\{a\}} \left[M_{b\to i}^{1f\,(\ell-1)} + M_{b\to i}^{1w\,(\ell-1)}\right] - \prod_{b\in C(i)\setminus\{a\}} M_{b\to i}^{1w\,(\ell-1)} \\
M_{i\to a}^{0w\,(\ell)} &= \prod_{b\in C(i)\setminus\{a\}} \left[M_{b\to i}^{0f\,(\ell-1)} + M_{b\to i}^{0w\,(\ell-1)}\right] - \prod_{b\in C(i)\setminus\{a\}} M_{b\to i}^{0w\,(\ell-1)} - \sum_{c\in C(i)\setminus\{a\}} M_{c\to i}^{0f\,(\ell-1)} \prod_{b\in C(i)\setminus\{a,c\}} M_{b\to i}^{0w\,(\ell-1)} \\
M_{i\to a}^{1w\,(\ell)} &= \prod_{b\in C(i)\setminus\{a\}} \left[M_{b\to i}^{1f\,(\ell-1)} + M_{b\to i}^{1w\,(\ell-1)}\right] - \prod_{b\in C(i)\setminus\{a\}} M_{b\to i}^{1w\,(\ell-1)} - \sum_{c\in C(i)\setminus\{a\}} M_{c\to i}^{1f\,(\ell-1)} \prod_{b\in C(i)\setminus\{a,c\}} M_{b\to i}^{1w\,(\ell-1)} \\
M_{i\to a}^{*\,(\ell)} &= w_{\text{info}} \prod_{b\in C(i)\setminus\{a\}} M_{b\to i}^{*\,(\ell-1)}
\end{aligned}
\tag{3.70}
$$

Checks to bits update rules

$$
\begin{aligned}
M_{a\to i}^{0f\,(\ell)} &= \frac{1}{2}\Big(\prod_{j\in V(a)\setminus\{i\}} \left[M_{j\to a}^{0f\,(\ell)} + M_{j\to a}^{1f\,(\ell)}\right] + \prod_{j\in V(a)\setminus\{i\}} \left[M_{j\to a}^{0f\,(\ell)} - M_{j\to a}^{1f\,(\ell)}\right]\Big) \\
M_{a\to i}^{1f\,(\ell)} &= \frac{1}{2}\Big(\prod_{j\in V(a)\setminus\{i\}} \left[M_{j\to a}^{0f\,(\ell)} + M_{j\to a}^{1f\,(\ell)}\right] - \prod_{j\in V(a)\setminus\{i\}} \left[M_{j\to a}^{0f\,(\ell)} - M_{j\to a}^{1f\,(\ell)}\right]\Big) \\
M_{a\to i}^{0w\,(\ell)} &= \prod_{j\in V(a)\setminus\{i\}} \left[M_{j\to a}^{*\,(\ell)} + M_{j\to a}^{1w\,(\ell)} + M_{j\to a}^{0w\,(\ell)}\right] - w_{\text{sou}} \prod_{j\in V(a)\setminus\{i\}} \left[M_{j\to a}^{1w\,(\ell)} + M_{j\to a}^{0w\,(\ell)}\right] \\
M_{a\to i}^{1w\,(\ell)} &= M_{a\to i}^{0w\,(\ell)} \\
M_{a\to i}^{*\,(\ell)} &= \prod_{j\in V(a)\setminus\{i\}} \left[M_{j\to a}^{*\,(\ell)} + M_{j\to a}^{1w\,(\ell)} + M_{j\to a}^{0w\,(\ell)}\right]
\end{aligned}
\tag{3.71}
$$

Bias equations calculated in ℓ-th iteration

$$
\begin{aligned}
\mu_i(0) &= \prod_{a\in C(i)} \left[M_{a\to i}^{0f\,(\ell)} + M_{a\to i}^{0w\,(\ell)}\right] - \prod_{a\in C(i)} M_{a\to i}^{0w\,(\ell)} - \sum_{b\in C(i)} M_{b\to i}^{0f\,(\ell)} \prod_{a\in C(i)\setminus\{b\}} M_{a\to i}^{0w\,(\ell)} \qquad \mu_i(*) = w_{\text{info}} \prod_{a\in C(i)} M_{a\to i}^{*\,(\ell)} \\
\mu_i(1) &= \prod_{a\in C(i)} \left[M_{a\to i}^{1f\,(\ell)} + M_{a\to i}^{1w\,(\ell)}\right] - \prod_{a\in C(i)} M_{a\to i}^{1w\,(\ell)} - \sum_{b\in C(i)} M_{b\to i}^{1f\,(\ell)} \prod_{a\in C(i)\setminus\{b\}} M_{a\to i}^{1w\,(\ell)}
\end{aligned}
\tag{3.72}
$$

Figure 3.11: Update equations for message-passing in the SP algorithm.

The purpose of the decimation function is to set a given number of the most biased info bits, reduce the graph $\mathbb{G}^{(1)}$ and the vector $\mathbf{s}^{(1)}$, and obtain a new graph $\mathbb{G}^{(2)}$ and vector $\mathbf{s}^{(2)}$ for the next round. The process of graph reduction is as follows: set the `num` most biased info bits to one if $\mu_i(1) > \mu_i(0)$, otherwise set them to zero. For each info bit i and its set value w_i^{set}, do the following operation: $\mathbf{s}_a^{(2)} = \text{XOR}(\mathbf{s}_a^{(1)}, w_i^{set})$, $\forall a \in C(i)$, where $\mathbf{s}_a^{(2)} = z_a^{(1)}$ for each unchanged check. This operation creates an equivalent source vector for the next round. Finally, the graph $\mathbb{G}^{(2)}$ is obtained from $\mathbb{G}^{(1)}$ by removing all info bits that were set including their incident edges.

After the decimation step, we obtain a new pair of input parameters $\mathbb{G}^{(2)}$ and $\mathbf{s}^{(2)}$ prepared for the next round of the `SP_iter()` function. Applying these steps again, we obtain a smaller graph $\mathbb{G}^{(3)}$ and a new source vector $\mathbf{s}^{(3)}$. The SP algorithm ends in the r-th round when the graph $\mathbb{G}^{(r)}$ does not contain any edges (all info bits were set).

To finalize the description of the algorithm, we need to describe the `SP_iter()` function in some round r. This function takes the source vector $\mathbf{s}^{(r)}$ and graph $\mathbb{G}^{(r)}$ and returns a vector of biases for each free info bit. The core of this function is the message-passing iteration process. This process is initiated by sending messages $\mathbf{M}_{\mathbf{s}_a\to a}^{(0)}$, defined as

$$
\mathbf{M}_{\mathbf{s}_a\to a}^{(0)} = \big(\psi_a(0), \psi_a(1), 0, 0, w_{sou}\big), \tag{3.73}
$$

where $\psi_a(1) = \mathbf{s}_a e^{\gamma} + (1-\mathbf{s}_a)e^{-\gamma}$, $\psi_a(0) = \frac{1}{\psi_a(1)}$, from source bits in graph $\mathbb{G}^{(r)}$ to their checks.

Checks forward these messages to their neighboring info bits and the process continues by applying the update equations from Figure 3.11. Each iteration consists of applying equations (3.70) for updating messages $\mathbf{M}_{i\to a}^{(\ell)}$ using messages $\mathbf{M}_{a\to i}^{(\ell-1)}$ from the previous iteration and applying equations (3.71) to obtain new $\mathbf{M}_{a\to i}^{(\ell)}$ messages from $\mathbf{M}_{i\to a}^{(\ell)}$. In (3.71), the constant message $\mathbf{M}_{\mathbf{s}_a\to a}^{(\ell)} = \mathbf{M}_{\mathbf{s}_a\to a}^{(0)}$ is used. All messages are *always normalized* so that the sum of all elements of the five-dimensional message vector is equal to 1. This is expressed using the `normalize()` pseudofunction. To speed up the iterations, after a few initial iterations (`start_damp`), the damping process is used. This process adjusts the $\mathbf{M}_{a\to i}^{(\ell)}$ messages using the the following equation: $\mathbf{M}_{a\to i}^{(\ell)} = \left(\mathbf{M}_{a\to i}^{(\ell)} \cdot \mathbf{M}_{a\to i}^{(\ell-1)}\right)^{1/2}$, where the product and square root are elementwise operations. The adjusted messages must be again normalized.

After the message-passing algorithm converged or the maximum number of iteration was reached, the biases $B_i = |\mu_i(1) - \mu_i(0)|$ are calculated for each free info bit i, where the three-dimensional vector $\big(\mu_i(0),\, \mu_i(1),\, \mu_i(*)\big)$ defined in (3.72) is normalized to sum to 1.

In Chapter 5, we will discuss implementation details of this algorithm and show the performance of SP based quantizer. The interpretation of the parameter γ used for initialization of source messages in each round is the same as in the Bias Propagation. Hence, we can use this algorithm for weighted binary quantization problem as well. Although we know how to interpret the value of the parameter γ, we have to determine its value from experiments.

3.9 Bias Propagation as a special case of SP based quantizer

In the two previous sections, we gave a description of two algorithms, BiP and 'SP based quantizer'. From the formal derivation of the BiP algorithm, we can see that the probability distribution (3.12) is a special case of (3.69), where $w_{sou} = w_{info} = 0$.

Precisely, we will obtain update equations (BiP-1)–(BiP-5) from Figure 3.7 by substituting values $w_{info} = 0$ and $w_{sou} = 0$ to update equations (3.70)–(3.72). Using this substitution, it is obvious that:

1. $M_{i\to a}^{*\,(\ell)} = 0 \quad \forall \ell$ (because $w_{info} = 0$)
2. $M_{a\to i}^{*\,(\ell)} = 0 \quad \forall \ell$ (the term from source-messages $M_{j\to a}^{*\,(\ell)} + M_{j\to a}^{1w\,(\ell)} + M_{j\to a}^{0w\,(\ell)} = 0$)
3. $M_{a\to i}^{0w\,(\ell)} = M_{a\to i}^{1w\,(\ell)} = 0 \quad \forall \ell$ (because $w_{sou} = 0$ and $M_{a\to i}^{*\,(\ell)} = 0$)
4. $M_{i\to a}^{0w\,(\ell)} = M_{i\to a}^{0f\,(\ell)} \quad \forall \ell$ (because $M_{b\to i}^{0w\,(\ell)} = 0$ and the form of the initial message)
5. $M_{i\to a}^{1w\,(\ell)} = M_{i\to a}^{1f\,(\ell)} \quad \forall \ell$ (because $M_{b\to i}^{1w\,(\ell)} = 0$ and the form of the initial message)
6. $M_{i\to a}^{0f\,(\ell)} + M_{i\to a}^{1f\,(\ell)} = 1 \quad \forall \ell$ (because of the normalization of messages)
7. $M_{\mathbf{s}_a\to i}^{0f\,(\ell)} + M_{\mathbf{s}_a\to i}^{1f\,(\ell)} = 1 \quad \forall \ell$ (because of the normalization of the source message).

And we obtain

$$\begin{aligned}
\mathbf{M}_{i\to a}^{(\ell)} =& \left(\prod_{b\in C(i)\setminus\{a\}} M_{b\to i}^{0f\,(\ell-1)}, \prod_{b\in C(i)\setminus\{a\}} M_{b\to i}^{1f\,(\ell-1)}, \prod_{b\in C(i)\setminus\{a\}} M_{b\to i}^{0f\,(\ell-1)}, \prod_{b\in C(i)\setminus\{a\}} M_{b\to i}^{1f\,(\ell-1)}, 0 \right) \\
M_{a\to i}^{0f\,(\ell)} =& \frac{1}{2}\left(\prod_{j\in\bar{V}(a)\setminus\{i\}} \left[M_{j\to a}^{0f\,(\ell)} + M_{j\to a}^{1f\,(\ell)} \right] + \prod_{j\in\bar{V}(a)\setminus\{i\}} \left[M_{j\to a}^{0f\,(\ell)} - M_{j\to a}^{1f\,(\ell)} \right] \right) \\
M_{a\to i}^{1f\,(\ell)} =& \frac{1}{2}\left(\prod_{j\in\bar{V}(a)\setminus\{i\}} \left[M_{j\to a}^{0f\,(\ell)} + M_{j\to a}^{1f\,(\ell)} \right] - \prod_{j\in\bar{V}(a)\setminus\{i\}} \left[M_{j\to a}^{0f\,(\ell)} - M_{j\to a}^{1f\,(\ell)} \right] \right) \\
M_{a\to i}^{0w\,(\ell)} =& M_{a\to i}^{1w\,(\ell)} = M_{a\to i}^{*\,(\ell)} = 0.
\end{aligned}$$

Finally, we use the following substitution

$$B_{i\to a}^{(\ell)} = \frac{M_{i\to a}^{0f\,(\ell)} - M_{i\to a}^{1f\,(\ell)}}{M_{i\to a}^{0f\,(\ell)} + M_{i\to a}^{1f\,(\ell)}}, \qquad S_{a\to i}^{(\ell)} = M_{a\to i}^{0f\,(\ell)} - M_{a\to i}^{1f\,(\ell)}, \tag{3.74}$$

to obtain the update equations for the Bias Propagation algorithm shown in Figure 3.7.

The fact that the binary quantization problem can be solved using the simple case $w_{sou} = w_{info} = 0$ was not mentioned in [24]. On the contrary, authors reported that they cannot obtain good results for $w_{sou} \approx 0$ and $w_{info} \approx 0$. According to our experiments, this is true, because for very small, but non-zero, values of these parameters, the algorithm does not produce near-optimal distortion. For $w_{sou} = w_{info} = 0$, we do not obtain exactly the same outputs from both algorithms, but the resulting distortions are the same.

Chapter 4

Determining the coset member and calculating syndromes

During embedding, the sender needs to find an arbitrary member of the coset $\mathcal{C}(\mathbf{m})$ for message $\mathbf{m}$. This operation requires a parity check matrix $\mathbf{H}$ of a given linear code to find a vector $\mathbf{v_m} \in \{0,1\}^n$ such that $\mathbf{Hv_m} = \mathbf{m}$. The extraction mapping algorithm will also use the matrix $\mathbf{H}$ to recover the hidden message from the stego image $\mathbf{y}$. The problem is that we only have a sparse generator matrix $\mathbf{G}$, whose dimension could be very high, e.g., $10^6 \times 5 \cdot 10^5$. We can use Gaussian elimination to find the null space of $\mathbf{G}$ to obtain matrix $\mathbf{H}$. However, the complexity of this procedure would be $\mathcal{O}(n^2)$ and we would lose the key property of the original matrix $\mathbf{G}$ its sparsity. It would not be even possible to store the resulting matrix using current hardware.

Fortunately, we do not need the whole matrix for performing our operations. It will be sufficient to perform a matrix by vector multiplication. Our problem here is similar to the problem of encoding in LDPC codes, where we have a sparse parity check matrix and want to encode a message into a codeword. This problem is solved for the case of LDPC codes by Richardson and Urbanke [20]. The complexity of the algorithm is almost linear for a large variety of sparse codes. We will use this approach to develop a similar algorithm for LDGM codes with linear complexity. We will derive the algorithm for degree distributions listed in Section 5.2. This algorithm will find partial diagonalization of matrix $\mathbf{G}$, so that we will be able to solve the equation $\mathbf{Hv_m} = \mathbf{m}$ in linear time.

Suppose that there exist row and column permutation matrices, $\mathbf{P}_r$ and $\mathbf{P}_c$, such that the matrix $\mathbf{G} \in \{0,1\}^{n \times n-m}$ can be brought into the following form:

$$(\mathbf{P}_r \mathbf{G} \mathbf{P}_c)^T = \hat{\mathbf{G}}^T = \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{pmatrix}, \tag{4.1}$$

where $\mathbf{T}$ is a lower triangular matrix. We search for row and column permutations that will maximize the size of the triangular matrix, because the larger the triangular matrix, the lower the resulting complexity. The dimensions of the matrices are shown in Figure 4.1, where g measures the number of rows (columns) that remain from matrix $\mathbf{T}$.

Using the special form of matrix $\hat{\mathbf{G}}$ from (4.1), we can easily find matrix $\hat{\mathbf{H}}$ in the following systematic form:

$$\hat{\mathbf{H}}^T = \begin{pmatrix} \mathbf{I} \\ \mathbf{\Phi}^{-1}\mathbf{\Psi} \\ \mathbf{T}^{-1}\left[\mathbf{A} + \mathbf{B}\mathbf{\Phi}^{-1}\mathbf{\Psi}\right] \end{pmatrix},$$

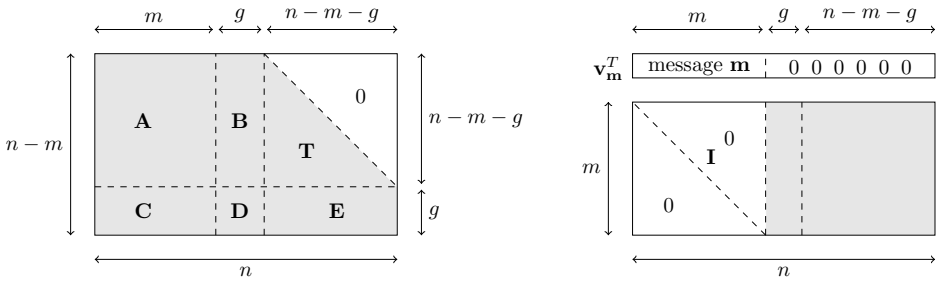


Figure 4.1: (a) Structure of matrix $\mathbf{G}^T$ after row and column permutation. Matrix $\mathbf{T}$ is lower triangular, $\mathbf{D}$ should be as small as possible. (b) Structure of parity check matrix $\mathbf{H}$ and arbitrary coset member $\mathbf{v}_\mathbf{m} \in \mathcal{C}(\mathbf{m})$ before permutation.

where we denoted $\mathbf{\Phi}^{-1} = (-\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D})^{-1}$ and $\mathbf{\Psi} = -\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C}$. We can verify the equation for $\hat{\mathbf{H}}$ by showing that $\hat{\mathbf{G}}^T\hat{\mathbf{H}}^T = 0$. It is not obvious that $\mathbf{\Phi}$ is non-singular and that we can invert it. At the end of this chapter, we will show that we can always make $\mathbf{\Phi}$ non-singular, therefore we will suppose that the inversion exists. Figure 4.1 (b) shows the structure of the parity check matrix. An arbitrary member of the coset $\mathcal{C}(\mathbf{m})$ can be obtained as $\mathbf{v}_\mathbf{m} = \mathbf{P}_r^{-1} \cdot (\mathbf{m}, 0)^T$, where the zero vector has length $n - m$. Here, we used the inverse row permutation matrix to place the vector into the correct order.

We now turn our attention to the extraction mapping, which needs to perform $\mathbf{H}\mathbf{y}$ to extract the message $\mathbf{m}$. We will not need to store the whole matrix $\mathbf{H}$. We decompose the permuted stego vector $\hat{\mathbf{y}} = \mathbf{P}_r\mathbf{y}$ into three parts $\hat{\mathbf{y}}^T = (\hat{\mathbf{y}}_1^T, \hat{\mathbf{y}}_2^T, \hat{\mathbf{y}}_3^T)$ of sizes m, g, $n - m - g$, and perform the following multiplication:

$$\mathbf{m} = \hat{\mathbf{y}}_1^T + \hat{\mathbf{y}}_2^T\mathbf{\Phi}^{-1}\mathbf{\Psi} + \mathbf{y}_3^T\mathbf{T}^{-1}[\mathbf{A} + \mathbf{B}\mathbf{\Phi}^{-1}\mathbf{\Psi}]. \tag{4.2}$$

All terms in this multiplication are sparse except for the (small) matrix $\mathbf{\Phi}^{-1}$, which we assume to be dense. There is no reason to hope that inversion of a sparse matrix will be sparse too. Fortunately, our algorithm for triangularization will always be able to reduce the matrix $\mathbf{D}$ to size 1×1, and because $\mathbf{D} \in \{0,1\}^{g\times g}$ and is non-singular it will always hold that $\mathbf{D} = 1$. From this fact, we can reduce the complexity of the extraction algorithm. Because the multiplication of a sparse matrix by a vector or a backsubstitution has linear complexity, our extraction algorithm will have complexity $\mathcal{O}(n)$, otherwise the complexity will increase due to the dense matrix multiplication to $\mathcal{O}(n + g^2)$, which is acceptable.

4.1 Algorithm for partial triangularization

In this section, we will describe the algorithm for partial triangularization of matrix $\mathbf{G}$. The complexity of this algorithm is not important, because the permutations are constant for each matrix $\mathbf{G}$ and finding the permutations is one time cost. On the other hand, the algorithm should run in a reasonable amount of time. Richardson and Urbanke described many such algorithms in their work [20]. Due to the special degree distributions we use for our codes, we can use the simplest one. We can also prove that the triangularizations are the best we can obtain using row and column permutations. The key property is the high

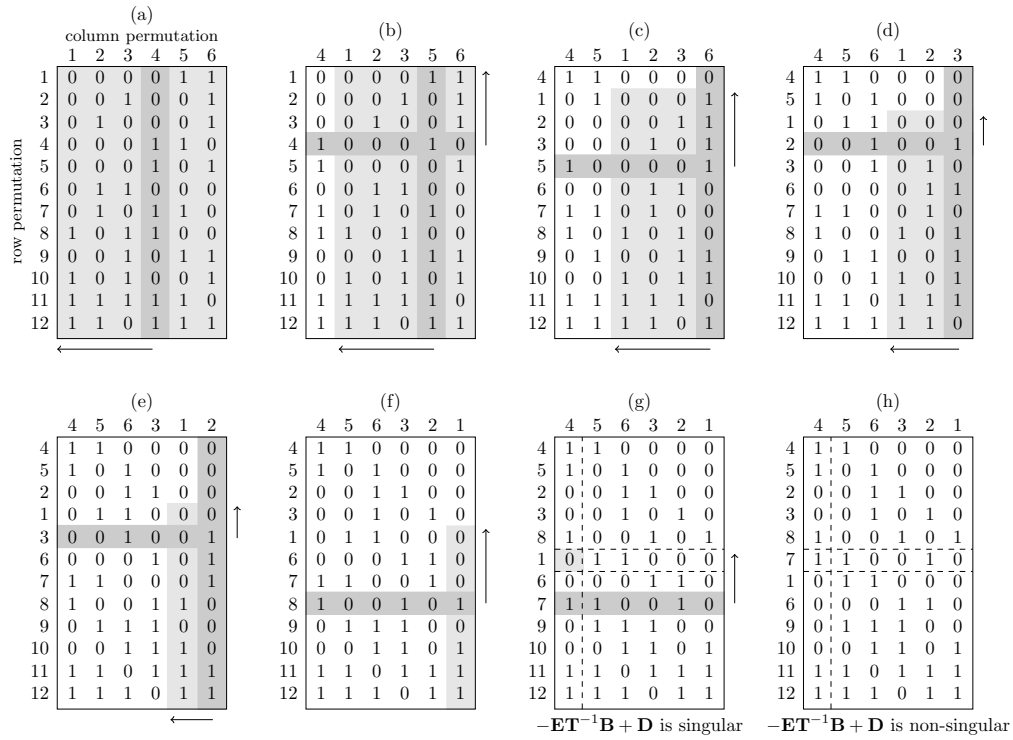


Figure 4.2: Example of partial triangularization process.

number of two degree nodes. In our case, we do have a large number of two degree check nodes.

We now describe the partial triangularization algorithm. It is an iterative, greedy algorithm which performs so called "diagonal extension step" in each iteration until the resulting matrix is empty. The diagonal extension step takes an input matrix, processes given row and column, and outputs a smaller matrix while it updates the resulting permutations. In more detail, assume we are given a matrix $\mathbf{A} \in \{0,1\}^{h\times w}$, and assume that the k-th row and l-th column should be processed. Update the row and column permutation in such a way that the k-th row will move up and become the first row in a matrix. Similarly move the l-th column to become the first column. No other row nor column will move. As an output matrix, return the matrix $\mathbf{A}$ without k-th row and l-th column. We will say that a row, or column, has degree d, if the sum along this row, or column, is equal to d.

The process of finding the row and column permutation is shown in Figure 4.2. Due to our specific degree distributions that have a large portion of 2-degree check-nodes (the resulting matrix $\mathbf{G}$ contains a large portion of 2-degree rows) and all info-bits have roughly the same degree, we can simply start the partial diagonalization of matrix $\mathbf{G}$ by removing an arbitrary column. This operation results with a high probability in creating a one-degree row. From now, we start iterating: in each iteration we apply the diagonal extension step on some one-degree row and its connected column. The diagonal extension step gives us a matrix without this row and column. With a high probability, the number of one-degree rows starts to grow. this iterative process ends when there is no column in the resulting matrix, or there are columns, but it is not possible to find a one-degree row. The first case is a successfull finish, where we output the created permutations. In the second case, we need to restart the algorithm and try to remove a different column at the begining. This unsuccessfull case is

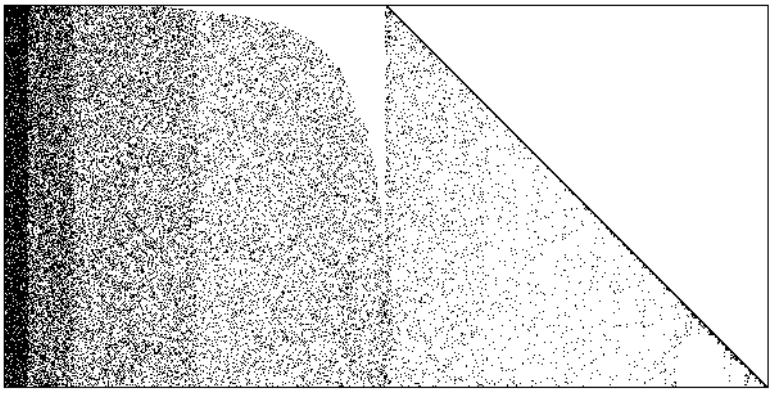


Figure 4.3: Results of applying partial triangularization process.

rare in practice due to the large number of 2-degree rows in our matrices.

4.2 Practical results of triangularization

When we apply the triangularization algorithm to any code obtained using degree distributions from Section 5.2, we will get a row and column permutation with gap $g = 1$. Such a permuted matrix is shown in Figure 4.1. The size of this gap is constant and does not depend on the size of matrix $\mathbf{G}$. This gap is the smallest we can achieve with row and column permutations, because $\mathbf{G}$ does not contain rows with degree one (we do not use degree distributions that will allow one degree nodes).

We now discuss the problem of singularity of matrix $\mathbf{\Phi}$. From the current point of view, there is no reason for this matrix to be non-singular. In our binary case, this matrix can only have two values: zero or one (dimension of the matrix $\mathbf{\Phi}$ is $g \times g$ and $g = 1$). Actually, due to sparsity of the other matrices, the probability that matrix $\mathbf{\Phi}$ is singular is very high. Fortunately, we can change this fact by permuting the columns in $\mathbf{G}^T$ and use another column that has a different value of the element in $\mathbf{D}$. This will change the value of matrix $\mathbf{D}$ and the value of matrix $\mathbf{\Phi}$ will become one which is trivial to invert. This correction step is shown in Figure 4.2 (g) and (h).

The partial triangularization algorithm described in the previous section was implemented in Matlab and tested for codes based on all degree distributions used in this thesis. The length of codes ranged from 500 to 10^6 bits. In all cases, the described algorithm was successfull in finding the row and column permutations with $g = 1$.

Chapter 5

Implementation and results

This last chapter is devoted to the presentation of achieved results.

5.1 Implementation details

Before we begin the discuss of the achieved results, we describe the implementation details for both algorithms. Both the BiP algorithm and the SP based quantizer share the same concept of message-passing. Messages passed along edges were represented using real numbers or vectors. From numerical experiments, there is no significant dependence on the precision we use for this representation. We implemented both algorithms using C++, where the 32 bit float data type was used to represent messages. Matlab was used for running simulations and for plotting graphical results.

For both algorithms, a good way how to implement the message-passing updates is the following. For each node in the graph, update all adjacent messages in the block; load input messages, use the corresponding update rules, and output all outgoing messages at once. Input messages were read using a precalculated permutation, while output messages were written sequentially. This ensures that the cache memory is used in an optimal way. Message updates were implemented for specific node degrees (e.g., 2, 4, 8, 12, 40) and manually optimized using Intel's SSE1 extension. Using this extension, we can perform operations on 4-tuples of float numbers in parallel. When some node has a degree (for example 6) that we did not optimize, we increase the degree using dummy messages. For the BiP algorithm, these dummy messages are $B_{i\to a}^{(\ell)} = 1$ and $S_{a\to i}^{(\ell)} = 0$. These messages do not change the value of other messages and we can use the message-update procedure for the nearest higher degree.

All results presented in this thesis were obtained using an Intel Core2 X6800 2.93GHz CPU machine with 2GB RAM. We ran the machine on Linux in 64 bit mode. All C++ code was optimized to 64 bit mode and compiled using Intel C++ 9.0 compiler. The speed of this algorithm (throughput) was measured, while both CPU cores were utilized. We ran the same algorithm on both cores, resulting in a 1.7–1.8 times higher throughput.

5.2 Degree Distributions

Both the BiP algorithm and the SP based quantizer need a sparse linear code to perform binary quantization. We use degree distributions to describe the generator matrix $\mathbf{G}$ of this

Rate: 0.37
$\rho(x) = 0.2710x + 0.2258x^2 + 0.1890x^5 + 0.0614x^6 + 0.2528x^{13}$
$\lambda(x) = 0.9522x^9 + 0.0478x^{10}$

Rate: 0.5
$\rho(x) = 0.1787x + 0.1762x^2 + 0.1028x^5 + 0.1147x^6 + 0.0122x^{12} + 0.0479x^{13} + 0.1159x^{14} + 0.2516x^{39}$
$\lambda(x) = 0.9988x^9 + 0.0012x^{10}$

Rate: 0.65
$\rho(x) = 0.2454x + 0.1921x^2 + 0.1357x^5 + 0.0838x^6 + 0.1116x^{12} + 0.0029x^{14} + 0.0222x^{15} + 0.0742x^{28} + +0.1321x^{32}$
$\lambda(x) = 0.4987x^5 + 0.5013x^6$

Rate: 0.75
$\rho(x) = 0.2912x + 0.1892x^2 + 0.0408x^4 + 0.0873x^5 + 0.0074x^6 + 0.1126x^7 + 0.0926x^{15} + 0.0187x^{20} + +0.1241x^{32} + 0.0361x^{39}$
$\lambda(x) = 0.8016x^4 + 0.1984x^5$

Figure 5.1: List of good degree distributions used for generating the results.

code. All degree distributions are listed from the edge perspective. In Figure 5.1, we present degree distributions that were used for generating all results. Some distributions were obtained from LdpcOpt site (`http://lthcwww.epfl.ch/research/ldpcopt/`) and optimized for the BSC channel. In this section, we describe these distributions and discuss the choice of the γ parameter.

From Section 3.7, we know the necessary condition that has to be satisfied when the BiP algorithm converges. Although we do not have such a condition for the SP based algorithm, we can use the distributions with both. The convergence condition has to be satisfied in each round. To be able to evaluate this condition, we solve the set of differential equations discussed in Section 3.7 (equations (3.64)–(3.66)). Although we know the analytical solution, we use equations (3.61)–(3.63) and obtain the set of iterative equations for a finite time step. By normalizing the solution by the number of remaining edges in each step, we obtain the check node degree distribution from the edge perspective. Using this approach, we can evaluate the convergence condition for all possible rounds.

For evaluating the performance of a given degree distribution when used with the BiP algorithm, we define *delay of degree distribution*. It is defined as the percentage of info bits that needs to be removed from the graph so that the BiP algorithm will converge in the next round ($\max_{i \in V} |B_i| > \mathtt{t}$). When we report the delay for some degree distribution, we think of average of delays from practical experiments. All degree distributions used with the BiP algorithm should have delay as small as possible. Degree distributions listed in Figure 5.1 have delays roughly around zero.

In Figure 5.2, we plot the behavior of the check node degree distribution ρ and the convergence condition for a regular (4,8) degree distribution. Codes based on regular degree distributions have very large delays. From the convergence graph, we can estimate the delay as an intersection of 1 and the convergence curve. From practical experiments of running the BiP algorithm, we estimate the delay from quantizing 30 random source sequences ($\gamma = 1$) and obtain 31.8%. This value is close to the value what we can estimate from the graph. In Figures 5.3, 5.4, 5.5, 5.6, we plot the convergence graph of each degree distribution from Figure 5.1.

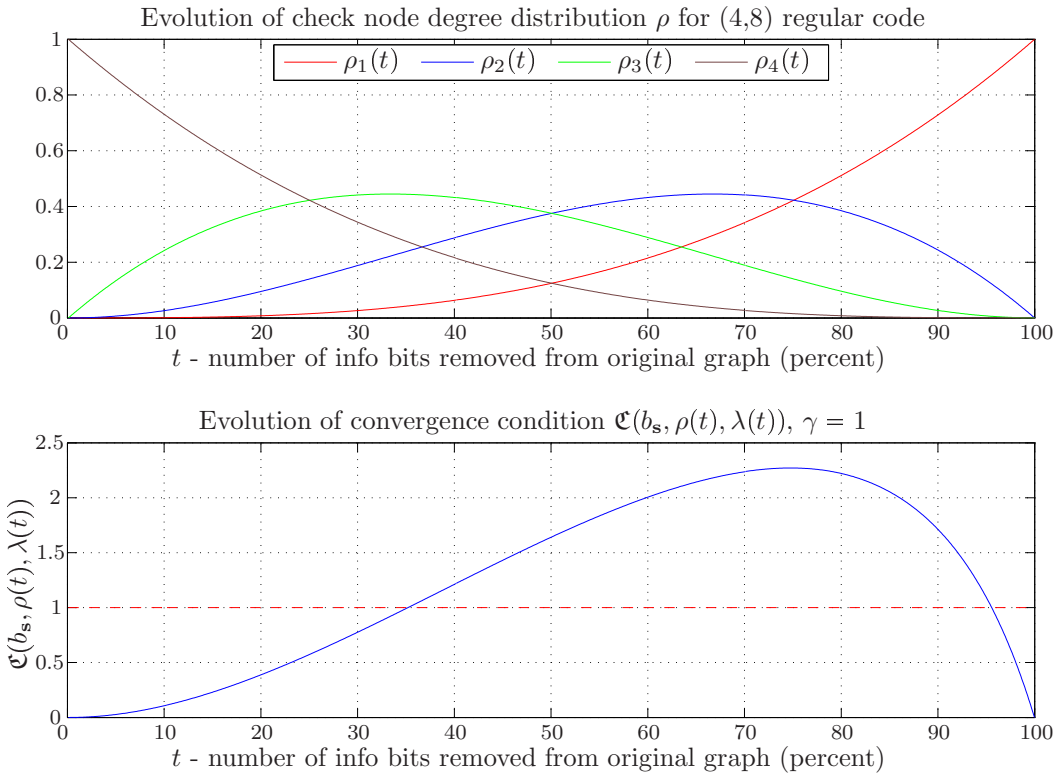


Figure 5.2: Convergence condition graph for regular (4,8) code.

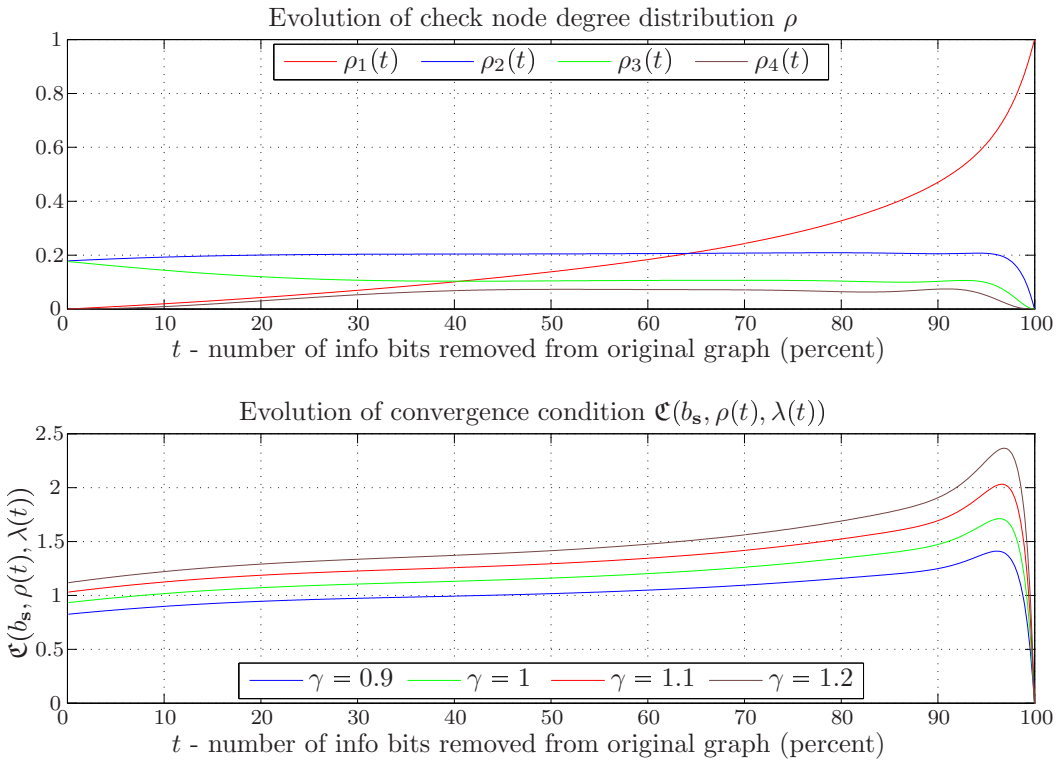


Figure 5.3: Convergence condition graph, degree distribution for $R = 0.5$ from Figure 5.1.

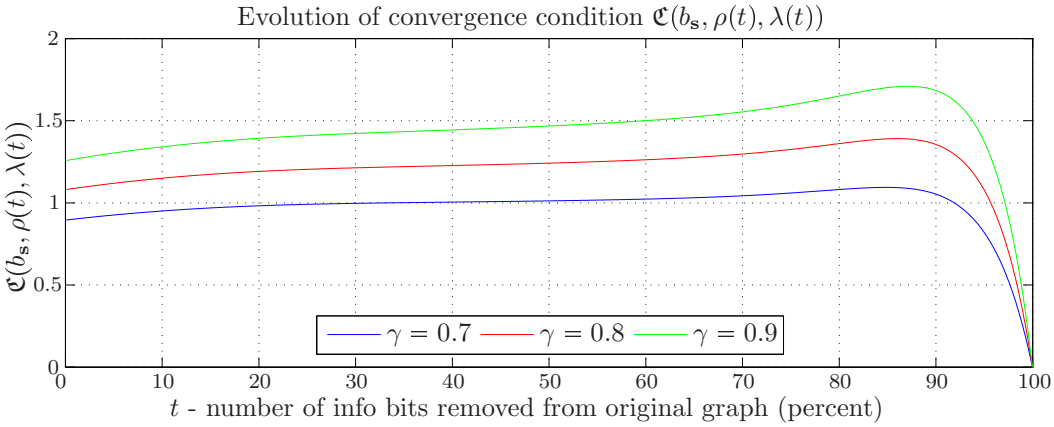


Figure 5.4: Convergence condition graph, degree distribution for $R = 0.37$ from Figure 5.1.

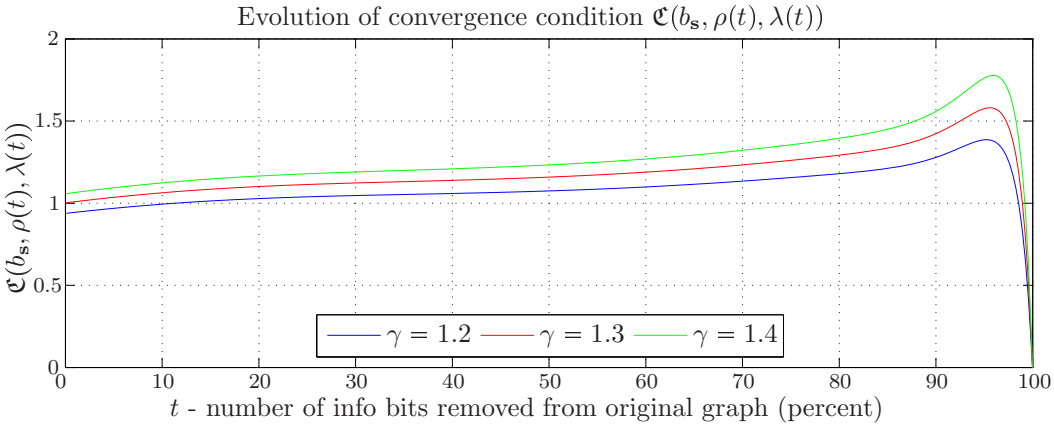


Figure 5.5: Convergence condition graph, degree distribution for $R = 0.65$ from Figure 5.1.

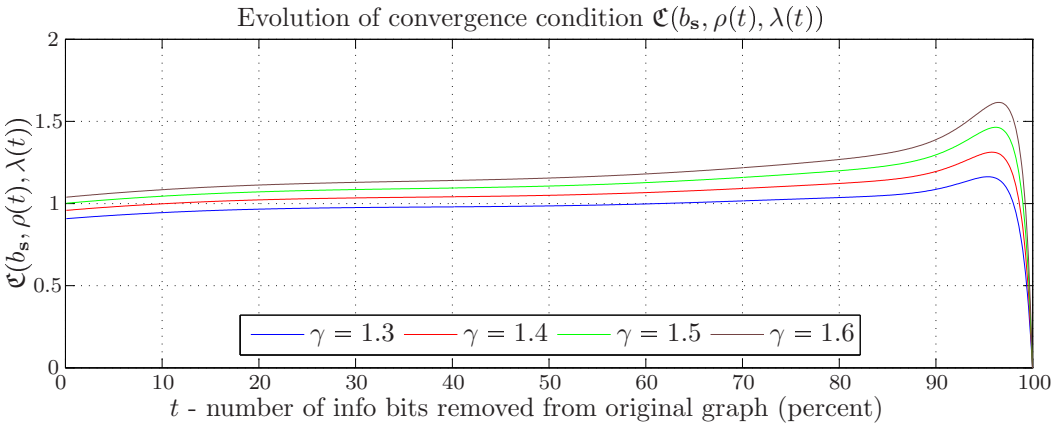


Figure 5.6: Convergence condition graph, degree distribution for $R = 0.75$ from Figure 5.1.

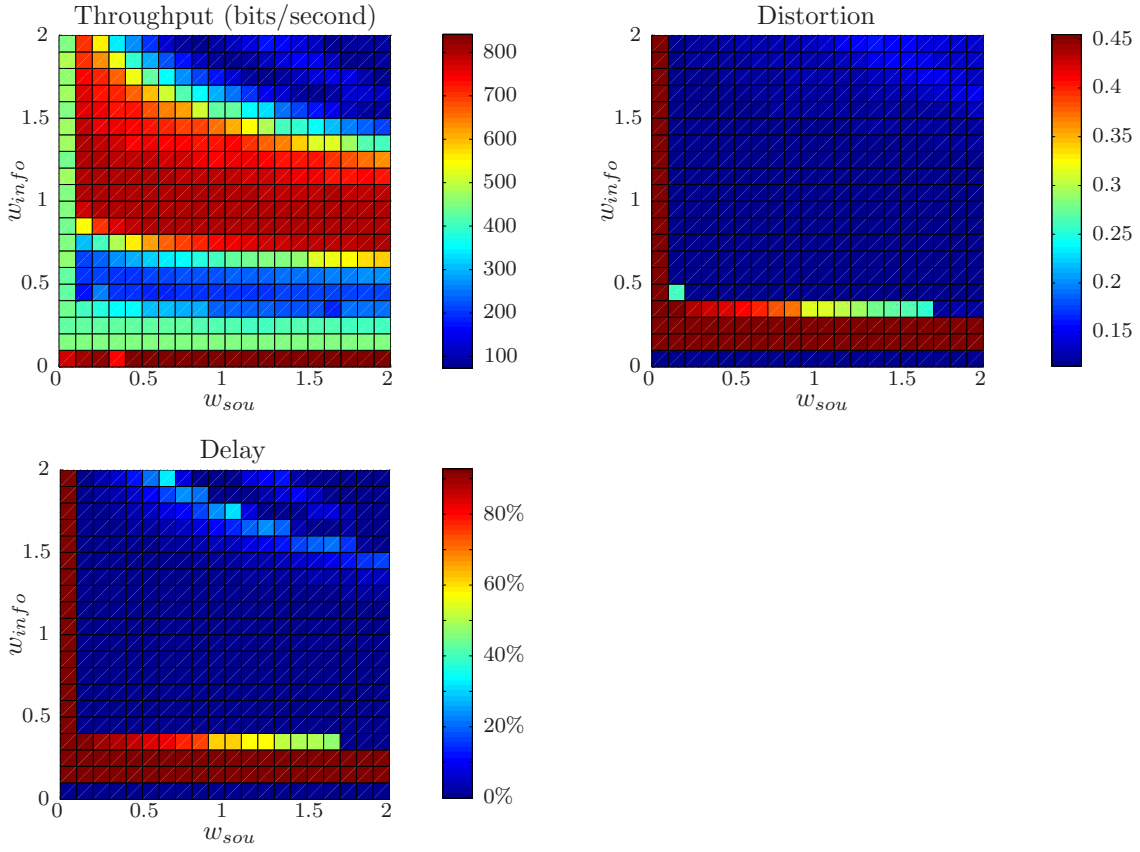


Figure 5.7: Behavior of SP based quantization algorithm for various values of w_{sou} and w_{info}. Each result was obtained as an average over 20 trials, code length $n = 10000$. Analysis is shown for the degree distribution from Figure 5.1, $R = 0.5$.

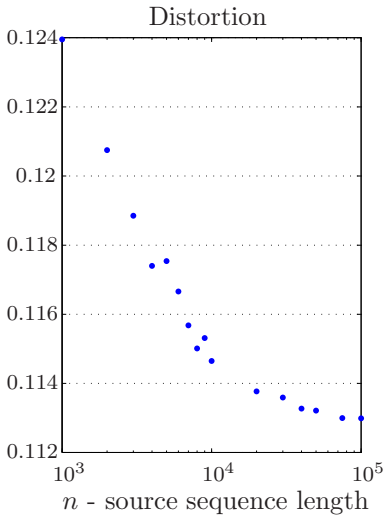
5.3 SP based quantizer vs. BiP algorithm

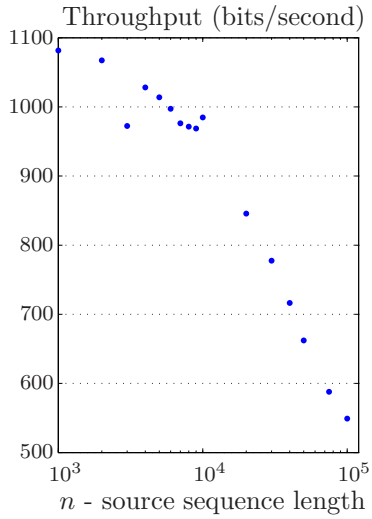
In this section, we present practical comparison of both algorithms — the SP based quantizer and BiP. We use the degree distribution for rate $R = 0.5$ from Figure 5.1. Other results are similar. Here and later in this chapter, we use *average distortion per bit* D (see equation (2.11)) to express the quality of the quantization process. The smaller the value, the better. To compare the speed of the algorithms, we define the *throughput (bits/second)* as the number of quantized source bits per second. Both values are calculated and compared for a constant rate R.

Figure 5.7 shows the dependence of the SP based quantizer on the parameters w_{sou} and w_{info}. In [24], authors suggested to use $w_{sou} = 1.1$ and $w_{info} = 1.0$. From this figure, we can see that the SP based quantizer works well even for other combinations. This figure contains the BiP algorithm as special case for $w_{sou} = w_{info} = 0$.

In Figure 5.8, we present the distortion and throughput comparison for various code lengths (n) for both algorithms. For each code length, we quantize 30 random source sequences and calculate the average. From this comparison, we can see that both algorithms perform very well. However, when we compare the throughputs, the BiP algorithm is almost 10-times faster. Here, we should note that both algorithms were implemented using Intel SSE1. The speed up is mainly due to the much simpler update equations and a smaller message length.

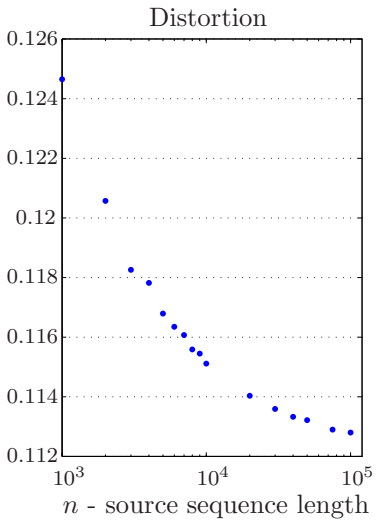
SP based quantizer:

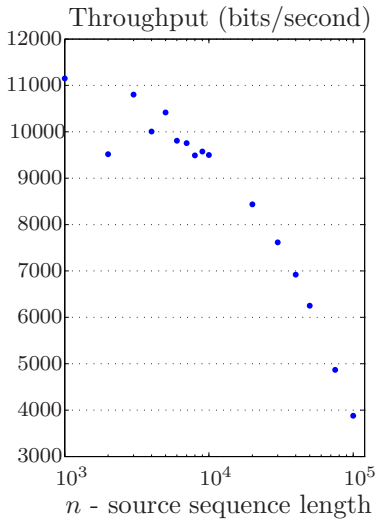




$n \cdot 10^3$	Dist.	Through.
1	0.1239	1081
2	0.1207	1067
3	0.1188	972
4	0.1174	1028
5	0.1175	1013
6	0.1166	997
7	0.1156	976
8	0.1150	971
9	0.1153	968
10	0.1146	984
20	0.1137	845
30	0.1135	777
40	0.1132	716
50	0.1132	662
75	0.1129	587
100	0.1129	549

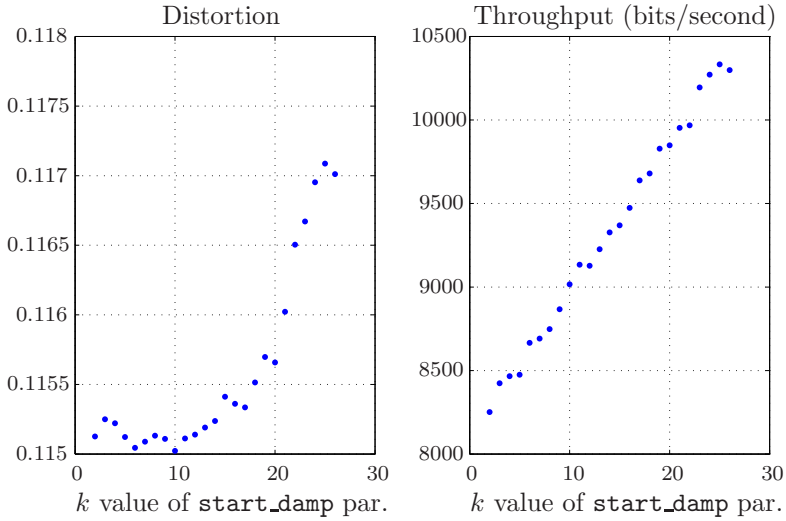
BiP algorithm:





$n \cdot 10^3$	Dist.	Through.
1	0.1246	11149
2	0.1205	9517
3	0.1182	10802
4	0.1178	10005
5	0.1167	10415
6	0.1163	9808
7	0.1160	9756
8	0.1155	9492
9	0.1154	9575
10	0.1151	9498
20	0.1140	8435
30	0.1135	7615
40	0.1133	6921
50	0.1132	6251
75	0.1129	4868
100	0.1128	3879

Figure 5.8: Comparison of SP based quantizer and BiP algorithm. Results were obtained as an average over 30 samples.



k	Dist.	Through.
2	0.1151	8251
4	0.1152	8465
6	0.1150	8665
8	0.1151	8747
10	0.1150	9016
12	0.1151	9127
14	0.1152	9327
16	0.1153	9473
18	0.1155	9680
20	0.1156	9848
21	0.1160	9953
22	0.1165	9968
23	0.1166	10195
24	0.1169	10272
26	0.1170	10299

Figure 5.9: Influence of damping on BiP algorithm. Calculated as average over 100 trials, `max_iter` = 25, code length $n = 10000$. Result for $k = 26$ was obtained without damping.

Later in this chapter, we omit the results obtained from the SP based algorithm and study only the BiP. From our practical experiments, we did not find any reason why to use the SP based algorithm instead of the BiP in practice. Both algorithms can perform weighted binary quantization.

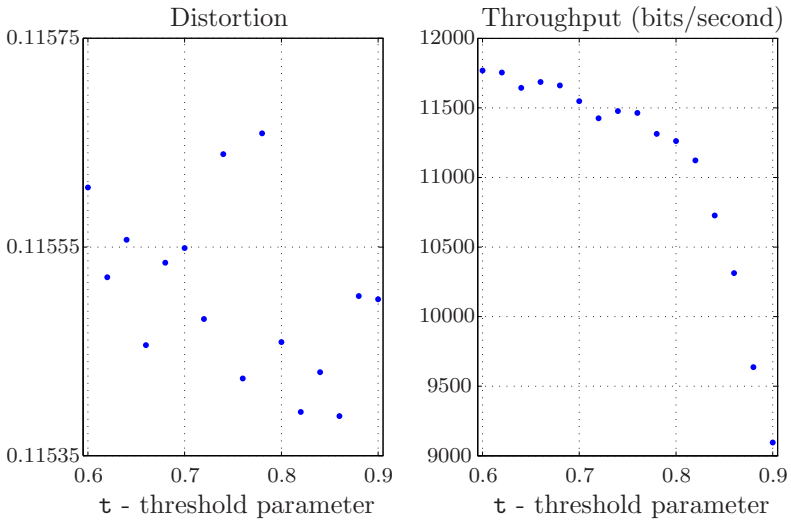
5.4 Damping and restarting

In Section 3.4, we defined the BiP as an iterative algorithm. In each round, we used a constant message $B_{i\to a}^{(1)} = 1$ to start the iterative process. This initialization is correct, however in later rounds we can use messages from the previous round to do the initialization. Formally, in the r-th round ($r > 1$), we use $B_{i\to a}^{(\hat{\ell})}$ messages from the last iteration $\hat{\ell}$ from the $r-1$-th round to initialize the $B_{i\to a}^{(1)}$ messages in the r-th round. In practice, we extend our decimation procedure to truncate the array of $B_{i\to a}^{(\hat{\ell})}$ messages from the previous round and use this array to initialize the message-passing process. This simple approach allows us to decrease the number of iterations we need in each round. We will call this approach *restarting*. From experiments, we observed that the number of iterations can be decreased from 60–90 to 20–30, thus resulting in a speed up by a factor of 3. All results presented later in this chapter were generated using this approach.

In the rest of this section, we study the influence of damping on the BiP algorithm. Damping helps the algorithm converge but lowers the throughput. In practice, we should consider the trade-off between both values. This trade-off is influenced by the `start_damp` parameter. In Figure 5.9, we show the dependence for rate $R = 0.5$.

5.5 Decimation strategy analysis

In this section we study the dependence of the BiP algorithm on the decimation strategy parameters. The decimation strategy was described in Section 3.4 on page 43. We used



t	Dist.	Through.
0.60	0.1156	11768
0.62	0.1155	11754
0.64	0.1155	11644
0.66	0.1154	11686
0.68	0.1155	11661
0.70	0.1155	11547
0.72	0.1154	11425
0.74	0.1156	11476
0.76	0.1154	11463
0.78	0.1156	11313
0.80	0.1154	11261
0.82	0.1153	11122
0.84	0.1154	10726
0.86	0.1153	10312
0.88	0.1155	9636
0.90	0.1155	9094

Figure 5.10: Influence of threshold parameter `t` on BiP algorithm. Calculated as average over 100 trials, `max_iter` = 20, code length $n = 10000$.

three parameters `t`, `num_min`, `num_max` to choose the number of the most biased information bits we remove by decimation. To simplify the notation, we define the *decimation quality factor* Q as the percentage of bits decimated in each round. We set `num_max`$=Q \times 0.01m$ and `num_min`$=Q \times 0.001m$. We use $Q = 1$ as the default value. The dependence on the threshold parameter is shown in Figure 5.10. We show this dependence only for rate $R = 0.5$, because it is similar for other rates. We use `t` $= 0.8$ as the default value for all other results.

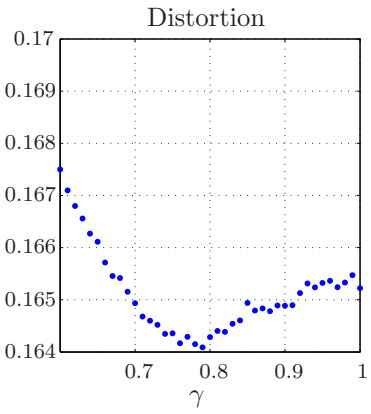
To further improve the performance, we introduced the concept of restarting in the previous section. To finish the description of this concept, we will use different values of the parameters `max_iter` and `start_damp` when $\max_{i \in V} |B_i| <$ `t` and when $\max_{i \in V} |B_i| >$ `t`. In practice, we start the process with constant values `max_iter` $= 60$ and `start_damp` $= 4$ and use these parameters until $\max_{i \in V} |B_i| >$ `t`. After the BiP algorithm starts to converge ($\max_{i \in V} |B_i| >$ `t`), we use the original values of these parameters. We use these (changed) values for the remaining rounds. All results were obtained using this strategy. The idea behind this strategy is that we need fewer iterations when the algorithm converges than in the beginning.

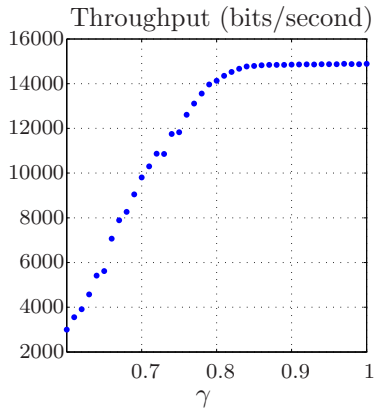
On page 77, 78, 79, 80 we present the decimation strategy analysis for all code rates. This analysis shows the behavior of the BiP algorithm for different values of the parameters. All results were obtained as an average over 100 random source sequences. On each page, we provide the recommended parameter values for the BiP algorithm for each rate.

5.6 Codeword quantization

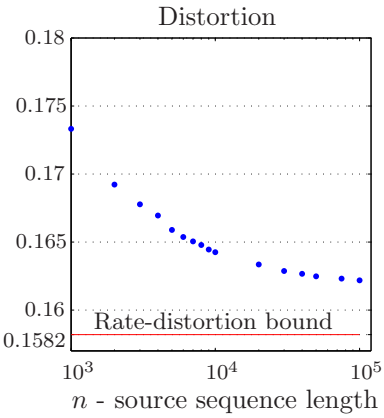
In this section, we study the behavior of the BiP algorithm when quantizing codewords. We expect that a codeword should be quantized to itself. This is because the closest codeword in $\mathcal{C}$ is the codeword itself. Unfortunately, the BiP algorithm does not behave this way. In Figure 5.11, we present the results of the following experiment. Each point was obtained by quantizing 100 codewords $\mathbf{Gw}$, for a random sequence $\mathbf{w}$ with a constant relative Hamming weight w. As a result of the quantization process, we obtain another codeword $\mathbf{G\hat{w}}$. Finally we plot the average relative Hamming weight of the vector $\hat{\mathbf{w}}$ and denote this value $\hat{w}$.

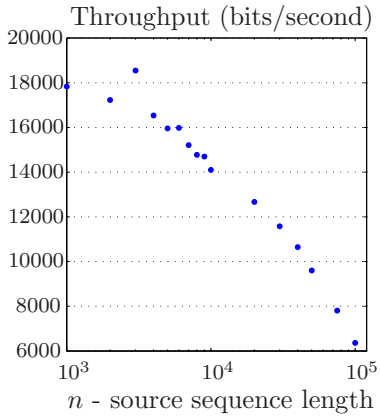
Decimation strategy analysis, $R = 0.37$.
Default parameters: `t` $= 0.8$, $Q = 1$, `max_iter` $= 25$, $n = 10000$, $\gamma = 0.8$.



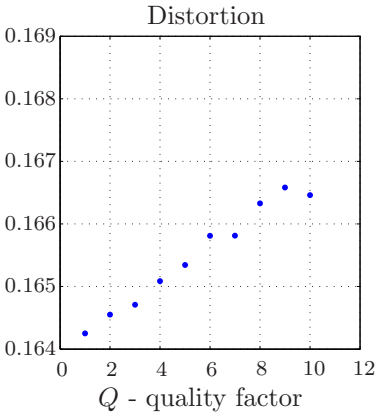


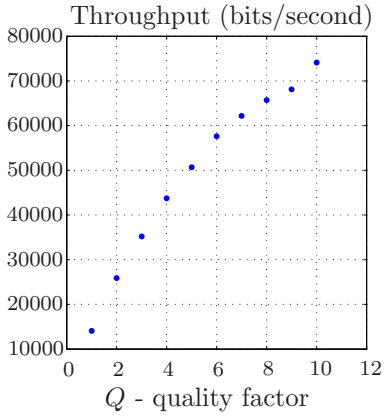
γ	Dist.	Through.
0.6	0.167498	3001
0.65	0.166112	5619
0.7	0.164935	9805
0.75	0.164357	11827
0.77	0.164289	13107
0.79	0.164089	13964
0.8	0.164284	14131
0.81	0.164403	14352
0.83	0.164541	14666
0.85	0.164939	14792
0.9	0.164885	14854
0.95	0.165322	14870



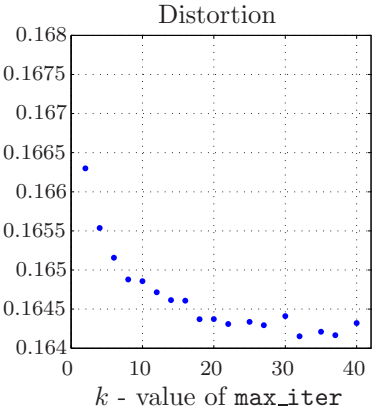


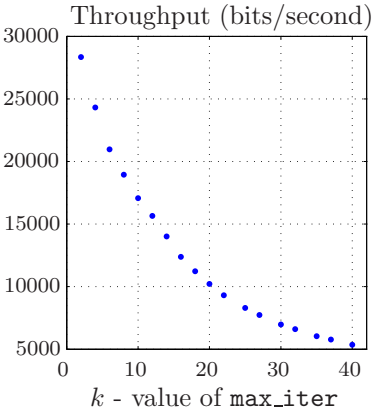
$n \cdot 10^3$	Dist.	Through.
1	0.1733	17831
3	0.1677	18545
5	0.1659	15954
7	0.1650	15209
9	0.1644	14698
10	0.1642	14098
20	0.1633	12666
30	0.1628	11579
40	0.1626	10643
50	0.1624	9602
75	0.1623	7805
100	0.1621	6363





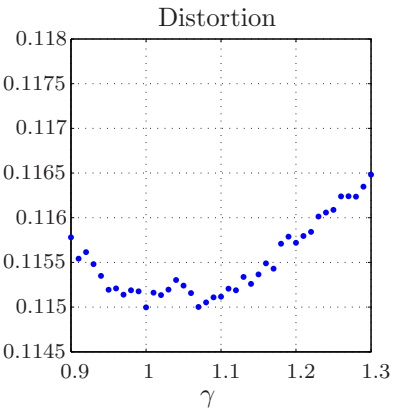
Q	Dist.	Through.
1	0.1642	14066
2	0.1645	25887
3	0.1647	35196
4	0.1650	43717
5	0.1653	50687
6	0.1658	57616
7	0.1658	62170
8	0.1663	65700
9	0.1665	68111
10	0.1664	74093

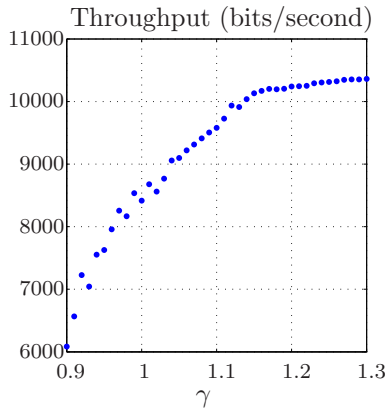




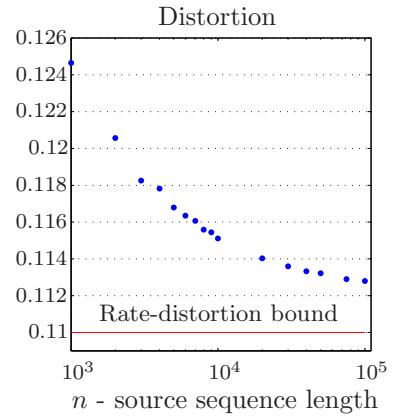
k	Dist.	Through.
2	0.1662	28338
6	0.1651	20973
10	0.1648	17070
14	0.1646	14015
18	0.1643	11227
20	0.1643	10218
22	0.1643	9309
25	0.1643	8297
27	0.1642	7733
30	0.1644	6978
35	0.1642	6038
40	0.1643	5359

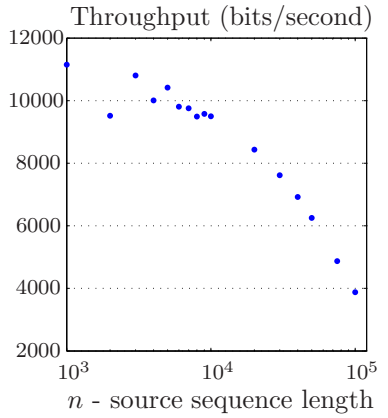
Decimation strategy analysis, $R = 0.5$.
Default parameters: `t` $= 0.8$, $Q = 1$, `max_iter` $= 25$, $n = 10000$, $\gamma = 1.07$.



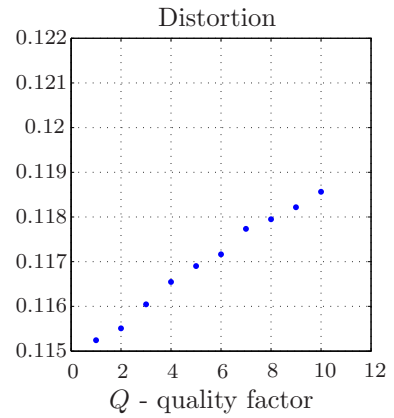


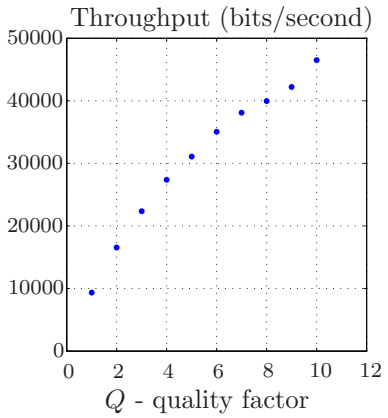
γ	Dist.	Through.
0.9	0.115781	6084
0.95	0.115194	7626
1	0.114998	8414
1.03	0.115196	8767
1.05	0.115241	9097
1.07	0.115002	9312
1.09	0.11511	9505
1.1	0.115117	9580
1.11	0.115206	9727
1.15	0.115366	10130
1.2	0.115719	10240
1.25	0.116088	10312



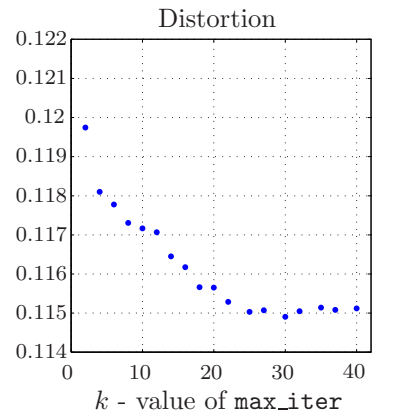


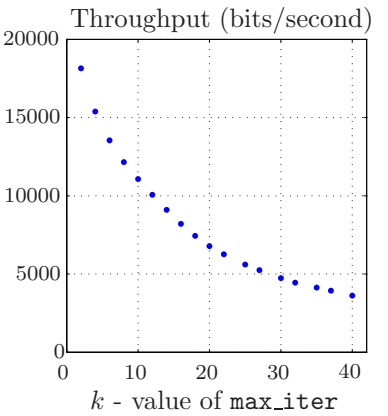
$n \cdot 10^3$	Dist.	Through.
1	0.1246	11149
3	0.1182	10802
5	0.1167	10415
7	0.1160	9756
9	0.1154	9575
10	0.1151	9498
20	0.1140	8435
30	0.1135	7615
40	0.1133	6921
50	0.1132	6251
75	0.1129	4868
100	0.1128	3879





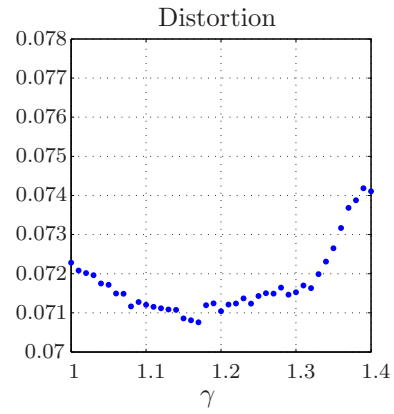
Q	Dist.	Through.
1	0.1152	9341
2	0.1155	16543
3	0.1160	22344
4	0.1165	27376
5	0.1169	31074
6	0.1171	35050
7	0.1177	38092
8	0.1179	39955
9	0.1182	42207
10	0.1185	46489

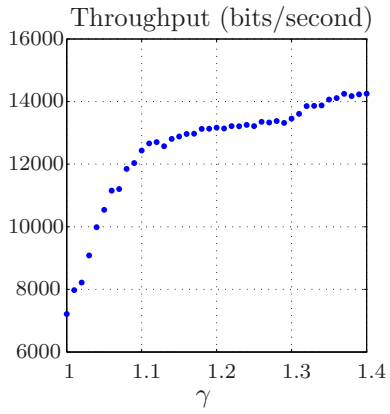




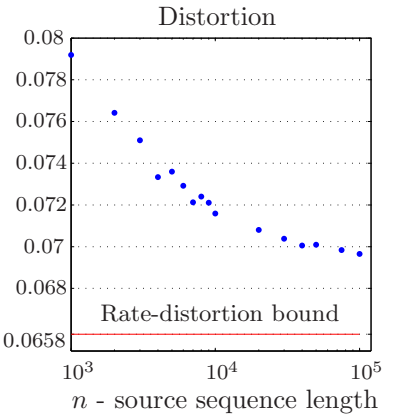
k	Dist.	Through.
2	0.1197	18145
6	0.1177	13543
10	0.1171	11069
14	0.1164	9097
18	0.1156	7435
20	0.1156	6783
22	0.1152	6261
25	0.1150	5606
27	0.1150	5242
30	0.1149	4735
35	0.1151	4133
40	0.1151	3622

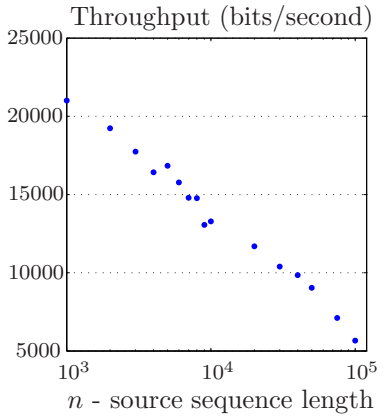
Decimation strategy analysis, $R = 0.65$.
Default parameters: `t` $= 0.8$, $Q = 1$, `max_iter` $= 25$, $n = 10000$, $\gamma = 1.3$.



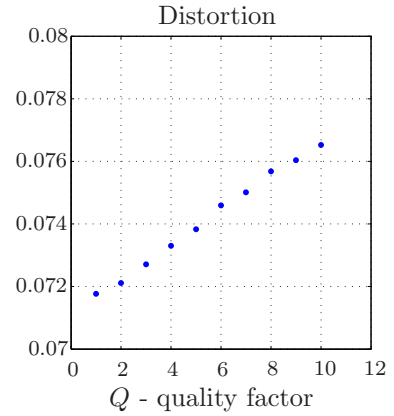


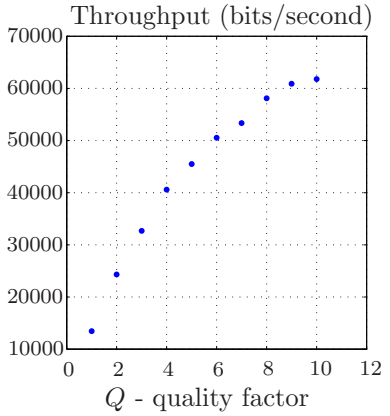
γ	Dist.	Through.
1	0.072281	7211
1.05	0.071717	10540
1.1	0.071204	12437
1.15	0.070856	12881
1.2	0.071041	13164
1.25	0.07143	13215
1.27	0.071492	13332
1.29	0.071464	13316
1.3	0.071526	13452
1.31	0.0717	13607
1.33	0.071989	13863
1.35	0.072651	14063



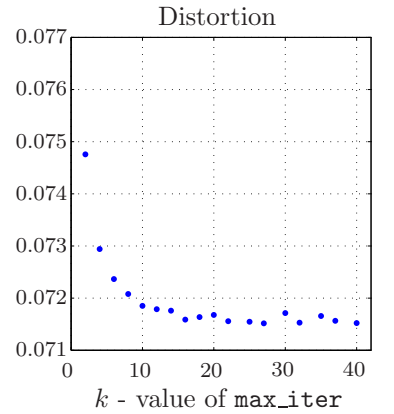


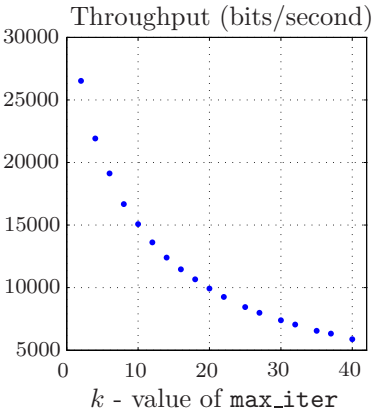
$n \cdot 10^3$	Dist.	Through.
1	0.0791	21006
3	0.0750	17737
5	0.0735	16840
7	0.0721	14793
9	0.0721	13056
10	0.0715	13283
20	0.0708	11688
30	0.0703	10393
40	0.0700	9847
50	0.0700	9032
75	0.0698	7106
100	0.0696	5661





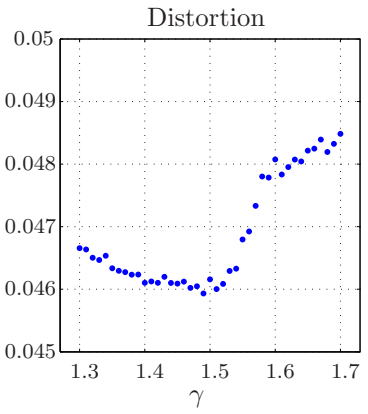
Q	Dist.	Through.
1	0.0717	13443
2	0.0721	24305
3	0.0727	32661
4	0.0733	40569
5	0.0738	45477
6	0.0745	50530
7	0.0750	53325
8	0.0756	58096
9	0.0760	60887
10	0.0765	61778

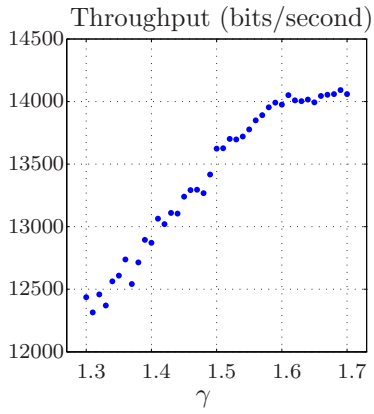




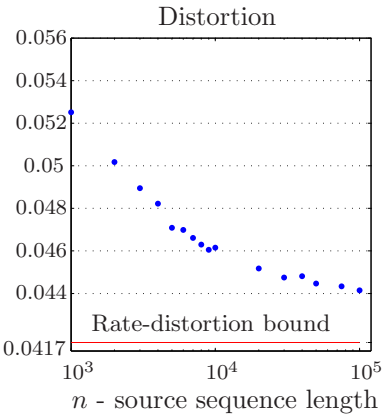
k	Dist.	Through.
2	0.0747	26517
6	0.0723	19122
10	0.0718	15076
14	0.0717	12399
18	0.0716	10660
20	0.0716	9930
22	0.0715	9253
25	0.0715	8448
27	0.0715	7992
30	0.0717	7395
35	0.0716	6553
40	0.0715	5883

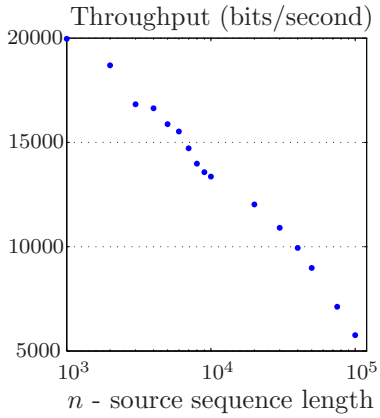
Decimation strategy analysis, $R = 0.75$.
Default parameters: `t` $= 0.8$, $Q = 1$, `max_iter` $= 25$, $n = 10000$, $\gamma = 1.5$.



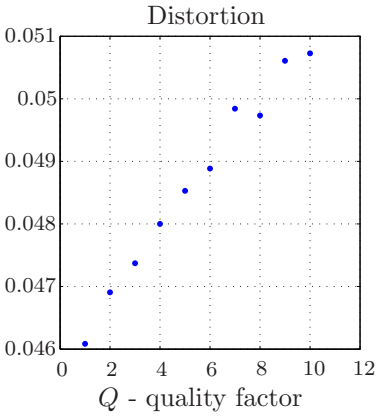


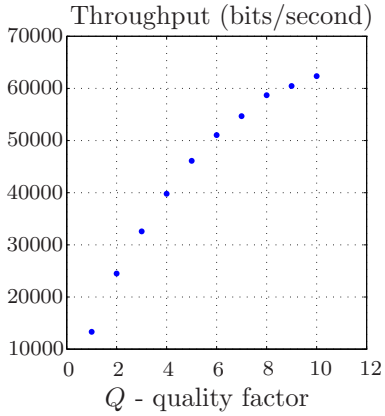
γ	Dist.	Through.
1.3	0.046657	12436
1.35	0.046336	12609
1.4	0.046102	12871
1.45	0.04609	13240
1.47	0.046022	13295
1.49	0.045933	13417
1.5	0.046158	13623
1.51	0.046003	13627
1.53	0.046292	13697
1.55	0.046796	13777
1.6	0.048075	13975
1.65	0.048215	13992



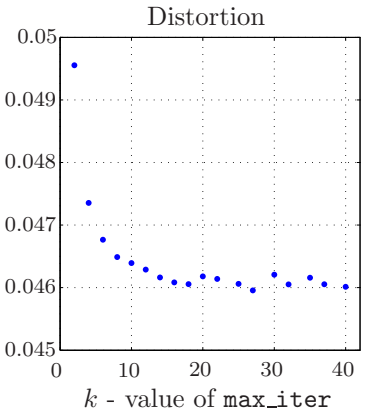


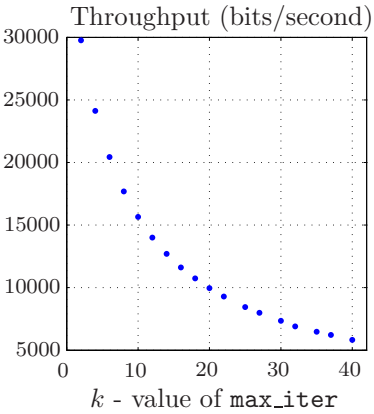
$n \cdot 10^3$	Dist.	Through.
1	0.0525	19962
3	0.0489	16823
5	0.0470	15873
7	0.0466	14714
9	0.0460	13568
10	0.0461	13362
20	0.0451	12024
30	0.0447	10906
40	0.0448	9938
50	0.0444	8980
75	0.0443	7122
100	0.0441	5758





Q	Dist.	Through.
1	0.0460	13314
2	0.0469	24485
3	0.0473	32570
4	0.0480	39771
5	0.0485	46109
6	0.0488	51046
7	0.0498	54658
8	0.0497	58670
9	0.0506	60438
10	0.0507	62340





k	Dist.	Through.
2	0.0495	29757
6	0.0467	20432
10	0.0463	15646
14	0.0461	12695
18	0.0460	10733
20	0.0461	9956
22	0.0461	9287
25	0.0460	8444
27	0.0459	7987
30	0.0462	7349
35	0.0461	6472
40	0.0460	5826

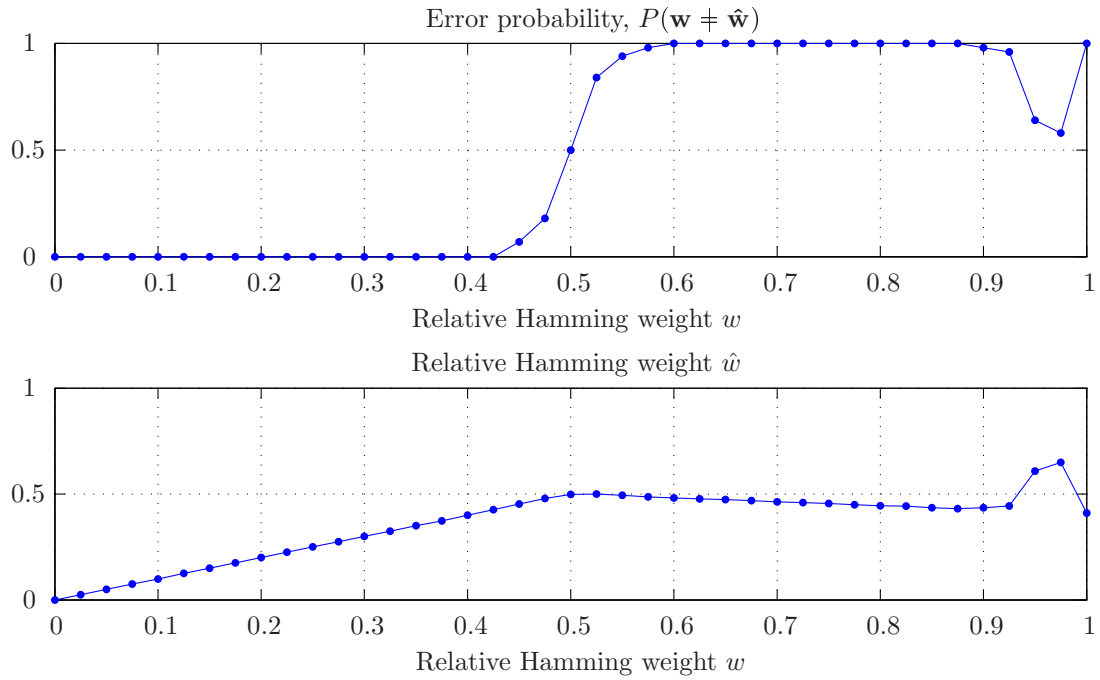


Figure 5.11: Results from quantizing codeword $\mathbf{Gw}$, where $\mathbf{w}$ is random with constant relative Hamming weight w. As a result of quantization process, we obtain codeword $\mathbf{G\hat{w}}$ with relative Hamming weight $\hat{w}$. Each point is an average over 100 trials, and $R = 0.5$.

Theretically, we should obtain a straight line $\hat{w} = w$, because all codewords should quantize to itself.

5.7 Weighted case of Bias Propagation

Up to now, we presented the results for uniform binary quantization. As discussed in Section 3.5, we can use the BiP algorithm with a non-constant γ and hence perform weighted binary quantization. Here, we should be carefull in setting γ_a to each check node $a \in C$. From convergence analysis, we have the requirement on the variance of $b_{\mathbf{s}}$. In practice, we can change γ_a, but the variance of $b_{\mathbf{s}}$ should be equal to the original variance with constant γ.

We now present the results for the so-called linear profile. For a source sequence $\mathbf{s}$ of length n, we define $\varrho_i = 2(n-i)/n$. The distortion is now defined as $D = \sum_{i=1}^{n} \varrho_i |\mathbf{s}_i - \hat{\mathbf{s}}_i|$, where $\hat{\mathbf{s}}$ is the reconstructed sequence. To obtain the theoretical probability of flipping (see equation (2.19)), we solve equation (2.20) and find the corresponding parameter ζ for given rate. This can be done easily using binary search. When we know ζ, we can obtain the lower bound on $p_a(1)$ for each source bit $a \in C$ (substitute ζ to (2.19)). To find optimal values of γ_a for each source bit, we use the following iterative approach. Start with $\gamma_a^{(0)} = \gamma$, where γ is taken from the uniform case. Find an estimate of $p_a(1)$ (denote it $\hat{p}_a(1)$) by quantizing k random source sequences. We calculate the estimate as an arithmetic average and finally we use a convolution filter to smooth the resulted sequence. We can now compare the estimate $\hat{p}_a(1)$ with the bound. Finally, we set $\gamma_a^1 = \gamma_a^0 + c\big(\hat{p}_a(1) - p_a(1) - (\hat{p} - p)\big)$, where $\hat{p}$ and p are the arithmetic averages of $\hat{p}_a(1)$ and $p_a(1)$ respectively, and c is a constant. This approach uses

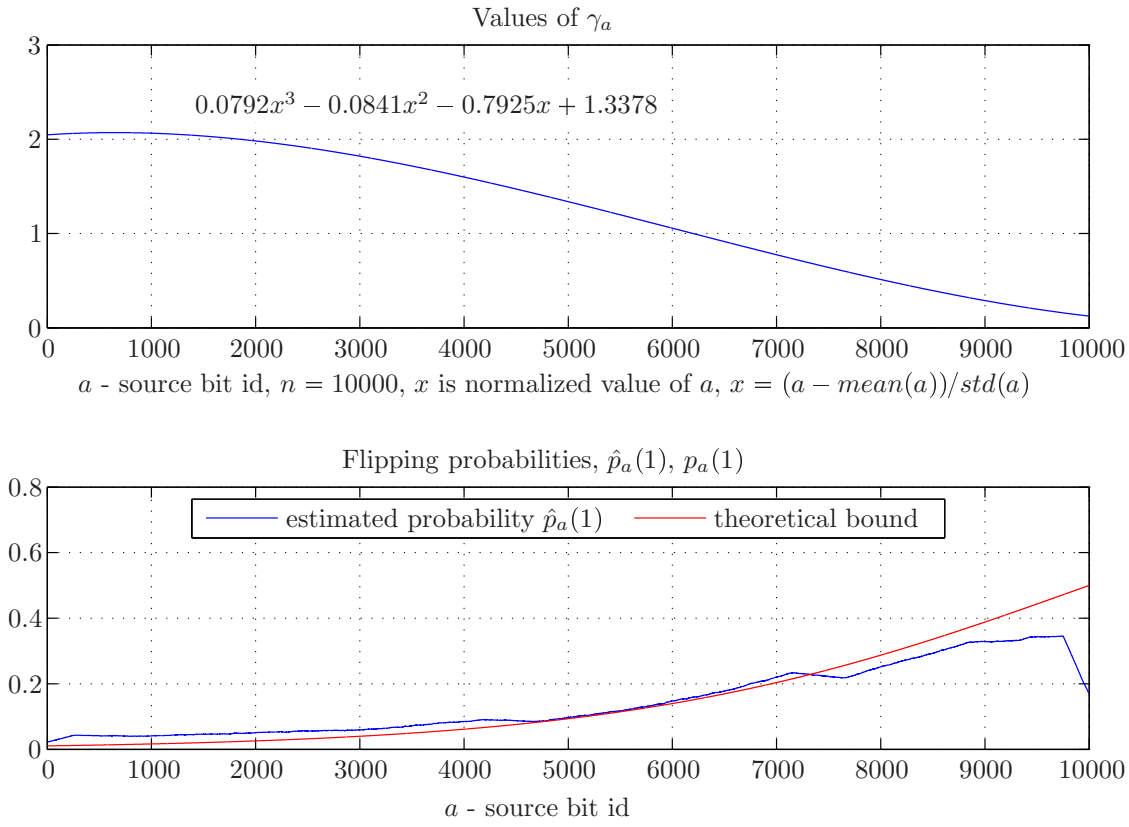
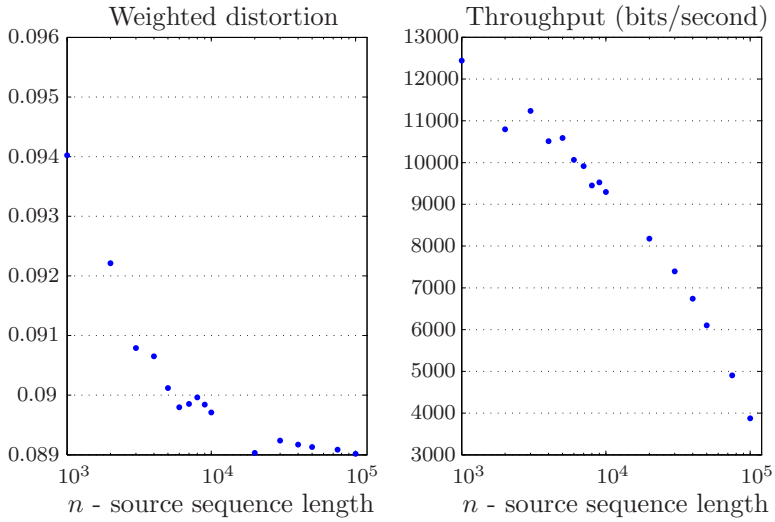


Figure 5.12: Flipping probabilities for linear profile ϱ, $R = 0.5$, $\zeta = 4.544$. Values of γ_a were obtained using an iterative approach for $n = 10000$ and interpolated by a cubic polynomial.

the difference between estimate and the lower bound to change the sequence of γ_a values. In practice, we use $c = 3$. Usually, 10 iterations were sufficient to obtain good results. In Figure 5.12, we present the resulting $\hat{p}_a(1)$ and γ_a for rate $R = 0.5$. We fit a cubic polynomial through the data points. The cubic polynomial uses a centralized variable x. In Figure 5.13, we use this polynomial and show how the BiP depends on the dimension. We can see that the throughput is roughly the same as for the non-weighted BiP. In Figure 5.14, we present the overall comparison of weighted vs. non-weighted BiP algorithm for all degree distributions. Here, we use the ordinary (non-weighted) BiP algorithm and measure the distortion using the weighted norm (linear case). Although the degree distributions were not optimized for the weighted case, the weighted BiP algorithm can still achieve very good results.

5.8 Application of proposed framework and results comparison

Here, we present the results when applying the proposed framework in steganography. We evaluate the performance of the codes by their embedding efficiency. Figure 5.15 shows the comparison with other previously proposed codes. Our results are labeled as 'LDGM codes' and each embedding efficiency was obtained by averaging over 100 randomly generated messages. For each relative message length, we ran the BiP algorithm for two different code lengths $n = 10000$ and $n = 100000$. The codes labeled as 'random codes' are obtained from



$n \cdot 10^3$	Dist.	Through.
1	0.0940	12444
2	0.0922	10798
3	0.0907	11237
5	0.0901	10589
7	0.0898	9915
9	0.0898	9525
10	0.0897	9294
20	0.0890	8175
30	0.0892	7393
40	0.0891	6738
50	0.0891	6099
75	0.0890	4904
100	0.0890	3873

Figure 5.13: Results from using weighted BiP algorithm for linear profile ϱ using different code lengths, $R = 0.5$. Values of γ_a for each check node $a \in C$ were obtained using the polynomial fit from Figure 5.12.

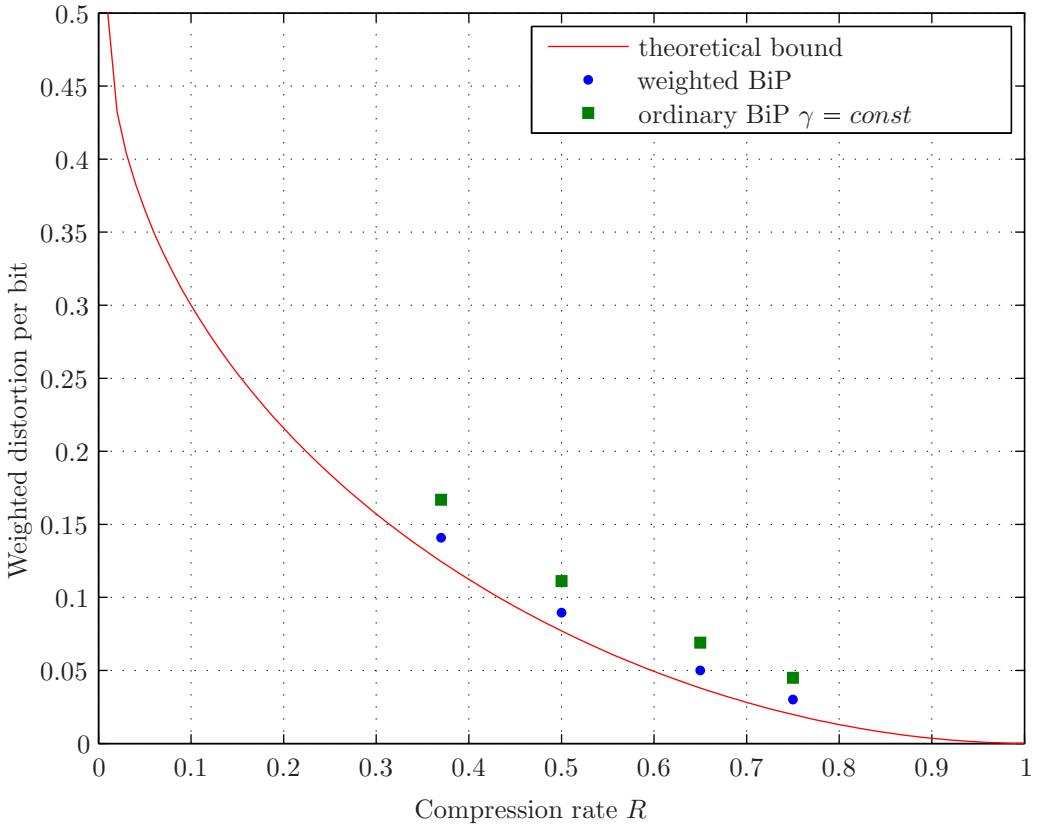


Figure 5.14: Overall comparison of weighted vs. non-weighted BiP algorithm for a linear profile ϱ. Codes were obtained from degree distributions from Figure 5.1 and were not optimized for weighted quantization, $n = 10000$. Values of γ_a were optimized using the above described iterative algorithm.

[11] and [12]. The remaining codes were taken from [2] and consist primarily of block-wise direct sum (BDS) of non-linear factor codes constructed using Preparata codes.

In Section 1.4, we describe adaptive and public key steganography. Now, we can use the approach based on Matrix Embedding to solve both problems without loss on capacity.

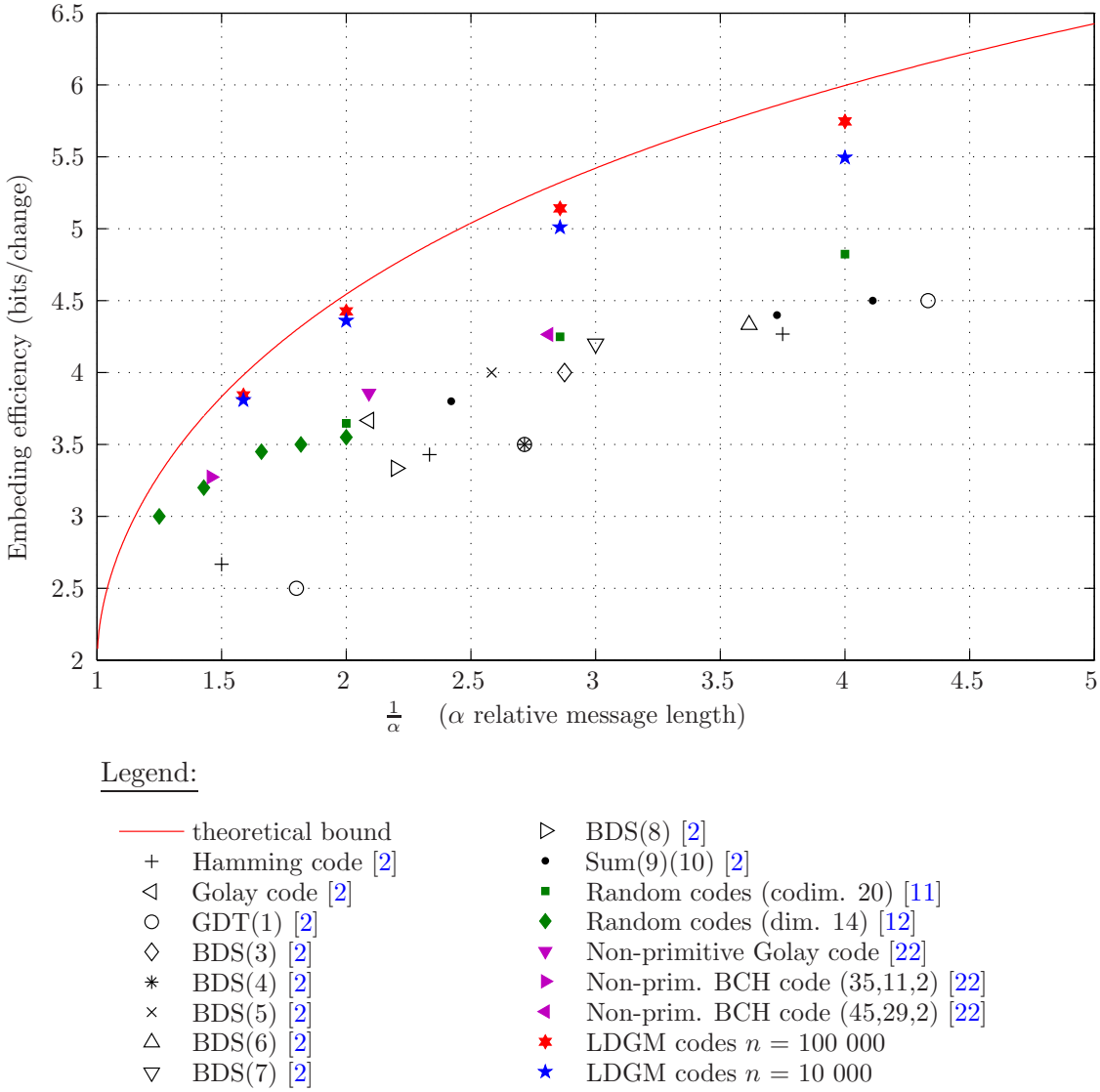


Figure 5.15: Comparison of LDGM codes with other previously proposed codes.

Conclusion

The focus of this thesis was the design of near-optimal steganographic schemes. We minimize the act of hiding a message in a digital image by minimizing the number of pixels that need to be changed. We showed that this problem is equivalent to binary quantization.

We propose a new algorithm for binary quantization based on the Belief Propagation algorithm with decimation over factor graphs of LDGM codes. We call the algorithm Bias Propagation (BiP). It allows performing binary quantization very close to the theoretical bound and thus enables construction of near-optimal steganographic schemes. Although the original problem is NP-complete in general, the Bias Propagation algorithm has log-linear complexity.

Using the Bias Propagation algorithm, we drastically reduce the complexity of binary quantization using LDGM codes. We believe this reduction is new and constitutes an important contribution as it allows us to theoretically study the algorithm. In our analysis, we used the tools originated from density evolution. We derived a necessary condition for the BiP algorithm to converge. This condition describes the form of LDGM codes that can be used with this algorithm.

In comparison to the state of the art work of Zechina et al. [5] or Wainwright et al. [24], the proposed algorithm is 10–100 times faster and is amenable to theoretical analysis. The problem of binary quantization is not restricted to steganography. It has many other applications in data compression and watermarking.

Bibliography

[1] J. Bierbrauer. On Crandall's problem. *Personal Communication, (available from http://www.ws.binghamton.edu/fridrich/covcodes.pdf)*, 1998.

[2] J. Bierbrauer and J. Fridrich. Constructing good covering codes for applications in Steganography. *In preparation, preprint available from http://www.ws.binghamton.edu/fridrich/stegocovsurveyOct06.pdf)*, 2006.

[3] A. Braunstein, M. Mezard, and R. Zecchina. Survey propagation: an algorithm for satisfiability. *Random Structures and Algorithms*, 27:201, 2005.

[4] C. Cachin. An information-theoretic model for steganography. In D. Aucsmith, editor, *Information Hiding, 2nd International Workshop*, volume 1525 of *LNCS*, pages 306–318. Springer-Verlag, New York, 1998.

[5] S. Ciliberti, M. Mezard, and R. Zecchina. Message passing algorithms for non-linear nodes and data compression, 2005.

[6] R. Crandall. Some notes on steganography. *Available from http://os.inf.tu-dresden.de/∼westfeld/crandall.pdf*, 1998.

[7] S. Dumitrescu, X. Wu, and Z. Wang. Detection of lsb steganography via sample pair analysis. In *IH '02: Revised Papers from the 5th International Workshop on Information Hiding*, pages 355–372, London, UK, 2003. Springer-Verlag.

[8] J. Fridrich. Minimizing the embedding impact in steganography. In *Proceedings ACM Multimedia and Security Workshop, Geneva, September 26–27*, 2006.

[9] J. Fridrich and T. Filler. Practical methods for minimizing embedding impact in steganography. In *Proceedings SPIE Photonics West, Electronic Imaging 2007, Security and Watermarking of Multimedia Contents, San Jose, CA*, 2007.

[10] J. Fridrich, M. Goljan, and D. Soukal. Perturbed quantization steganography. *ACM Multimedia and Security Journal*, 11(2):98–107, 2005.

[11] J. Fridrich, M. Goljan, and D. Soukal. Wet paper codes with improved embedding efficiency. *IEEE Transactions on Information Security and Forensics*, 1(1):102–110, 2006.

[12] J. Fridrich and D. Soukal. Matrix embedding for large payloads. *IEEE Transactions on Information Security and Forensics*, 1(3):390–394, 2006.

[13] F. Galand and G. Kabatiansky. Information hiding by coverings. In *Proceedings ITW2003, Paris, France*, pages 151–154, 2003.

[14] S. I. Gel'fand and M. S. Pinsker. Coding for channel with random parameters. *Probl. Pered. Inform. (Probl. Inform. Trans.)*, 9(1):19–31, 1980.

[15] A. Ker. A general framework for structural analysis of LSB replacement. In *Proceedings 7th Information Hiding Workshop, Barcelona, Spain, June 6–8*, 2005.

[16] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

[17] P. Lu, X. Luo, Q. Tang, and L. Shen. An improved sample pairs method for detection of LSB embedding. In J. Fridrich, editor, *Information Hiding, 6th International Workshop*, volume 3200 of *LNCS*, pages 116–127. Springer-Verlag, Berlin, 2004.

[18] E. N. Maneva, E. Mossel, and M. J. Wainwright. A new look at survey propagation and its generalizations, 2004.

[19] E. Martinian and M. J. Wainwright. Analysis of LDGM and compound codes for lossy compression and binning. In *Workshop on Information Theory and its Applications, San Diego*, February 2006.

[20] T. Richardson and R. Urbanke. Efficient encoding of low-density parity check codes. *IEEE Transactions on Information Theory*, 47(2):638–656, February 2001.

[21] T. Richardson and R. Urbanke. *Modern coding theory*. Cambridge University Press, 2007. Not published yet, download from http://lthcwww.epfl.ch/mct/.

[22] A. Schneidewind and D. Schönfeld. Embedding with syndrome coding based on BCH codes. In J. Dittman and J. Fridrich, editors, *Proceedings ACM Multimedia and Security Workshop, Geneva, Switzerland, September 26–27*, pages 214–223. ACM Press, New York, 2006.

[23] G. J. Simmons. The prisoners' problem and the subliminal channel. In D. Chaum, editor, *Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, CA, August 22–24*, pages 51–67. Plenum Press, New York, 1984.

[24] M. J. Wainwright and E. Maneva. Lossy source encoding via message-passing and decimation over generalized codewords of LDGM codes. In *Proceedings of the International Symposium on Information Theory, Adelaide, Australia*, September 2005.

[25] A. Westfeld. High capacity despite better steganalysis (F5—a steganographic algorithm). In I. S. Moskowitz, editor, *Information Hiding, 4th International Workshop*, volume 2137 of *LNCS*, pages 289–302. Springer-Verlag, New York, 2001.