UNIVERSIDADE FEDERAL FLUMINENSE

André Luiz Brazil

Path Relinking and AES Cryptography in Color Image Steganography

NITERÓI

- 2008 -

UNIVERSIDADE FEDERAL FLUMINENSE

André Luiz Brazil

Path Relinking and AES Cryptography in Color Image Steganography

A Dissertation submitted to the post graduate program in Computing of Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master in Computing. Area: Computação Visual e Interfaces (Visual Computing and Interfaces).

Supervisor: Aura Conci

NITERÓI

- 2008 -

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

B823 Brazil, André Luiz.
Path relinking and AES cryptography in color image steganography / André Luiz Brazil – Niterói, RJ : [s. n.], 2008.
93 f.

Orientador: Aura Conci
Dissertação (Mestrado em Computação) – Universidade Federal Fluminense, 2008.

1. Segurança de dados on-line. 2. Esteganografia. 3. Algoritmo genético. 4. Proteção de dados. I. Título.

CDD 005.8

Path Relinking and AES Cryptography in Color Image Steganography

André Luiz Brazil

A Dissertation submitted to the post graduate program in Computing of Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master in Computing. Area: Computação Visual e Interfaces (Visual Computing and Interfaces).

Approved by Dissertation Examining Committee:

_____

Profª. Aura Conci – IC/UFF (Supervisor)

_____

Luis Satoru Ochi – IC/UFF

_____

Profª. Célia Aparecida Zorzo Barcelos- CC/UFU

Niteroi, April 15th, 2008.

God, I am a lucky one;

To all my family and friends;

Acknowledgments

I would like to thank mommy, Isa Soares, and especially my dear aunt, Elcy Silva Soares, for providing everything at their reach to my former education and all emotional and particular care that they always have with me, until ever.

Also at my right side, an untiring helper, my goodhearted wife, Lúcia Blondet Baruque, always lighting my road and taking care of our child, so I have time and space to go on.

Another important little one is Allan, my son, for his humor and happiness, bringing back my smile even on those times when the things look gray.

Aura Conci, my advisor and great friend, couldn´t be better and kind one, every time giving me special attention and looking even at small details on our works together to make sure that everything is going smooth, never letting me down.

I must thank my fellows and bosses of Fundação Getulio Vargas too, for the professional development opportunities, everyday good life and excellent company, sometimes saving my "skin", and my chief Jean, for tolerance and flexibility on the clock so I could take small trips from Botafogo to UFF and have the classes without worry.

Last but not least, all the staff of Computer Institute of UFF, specially the teachers, for their share of knowledge, without these ones surely I would never have concluded that work. Special thanks to Luis Satoru Ochi, for trusting and incentives to turn our discipline's work into an article, which finally resulted into this bigger one. I hope you like this one too.

Thank you all.

Resumo

O uso da Internet aumenta a cada dia, conectando mais pessoas e aumentando o fluxo de informação e a necessidade de transmitir os dados de uma forma segura. Uma das formas de proteger as informações enviadas pela rede é ocultando os dados importantes dentro de uma imagem (o que é chamado de esteganografia), de forma a desviar a atenção e esconder essa informação de possíveis interceptadores. Este trabalho apresenta uma heurística híbrida, que combina duas técnicas de inteligência computacional: os algoritmos genéticos e a reconexão de caminhos, para aperfeiçoar a busca de melhores soluções no processo de esteganografia. O trabalho também incorpora o algoritmo de criptografia avançada AES, para aumentar a segurança das informações escondidas. Os resultados computacionais mostram que o algoritmo proposto supera em muito a técnica de substituição dos bits menos significativos (LSB) e também os resultados obtidos em nove outros trabalhos, em relação à qualidade da imagem stego. A incorporação da reconexão de caminhos e a possibilidade de esconder informações em imagens coloridas pode melhorar significativamente o desempenho dos algoritmos genéticos na esteganografia, ampliando o espaço disponível para ocultação em mais de três vezes quando comparado com a esteganografia em imagens de tons de cinza, principalmente devido a uma melhor utilização dos bits menos significativos. Outra grande vantagem em relação aos demais algoritmos é que este está apto a esconder qualquer tipo de mídia, incluindo documentos, arquivos-texto ou comprimidos, ou ainda um executável dentro da imagem de cobertura, o que amplia razoavelmente a gama de aplicações desse trabalho.

Palavras-chave: esteganografia em imagens coloridas, substituição LSB, ocultação de informações, sistema de criptografia avançado, algoritmo genético, reconexão de caminhos, segurança da informação, proteção de dados.

Abstract

Each day the Internet use grows up and connects more people, increasing the information flux and the need for transmitting data in a secure way. One of the ways to protect the data sent over the web is to conceal the relevant information inside a typical image, diverting the attention and hiding this data from intruders. This work proposes a hybrid heuristic, combining two methods of computational intelligence: a genetic algorithm and the path relinking refinement, to help solving an image processing issue. It also incorporates the AES advanced cryptography algorithm, to improve the hidden data security. Computational results show that the proposed algorithm outperforms the LSB (least significant bits) substitution technique and also the results of other nine works reported on the literature, concerning the quality of stego image. The inclusion of the path relinking procedure and the possibility to hide data inside colorful images can significantly improve the performance of genetic algorithms in image steganography, increasing the space available for information hiding by more than three times when compared to steganography using grayscale cover images, mostly related to a better least significant bits usage. Anything at all, including documents, text or compressed files, or yet an executable can be hidden inside the cover image, which considerably widens the scope of appliances of this work.

Keywords: color image steganography, LSB substitution, information hiding, advanced encryption standard, genetic algorithm, path relinking, information security, data protection.

List of Figures

Figure 1 – Mona Lisa, by Da Vinci : Who is the person portrayed? 1

Figure 2 - Expulsion of Heliodorus from the Temple, by Raphael : What the public symbols inside the rooms means? 2

Figure 3 - Creation of Adam, by Michelangelo : Can you see the human brain? 2

Figure 4 – Image hiding example 7

Figure 6 – Image hiding by least significant bits substitution 10

Figure 7 – Extraction of residual image (R) 12

Figure 8 – Generating the image to hide (E') 12

Figure 9 – Stego image (Z) generation by replacement of residual image (R) by image to hide (E') in cover image (C) 13

Figure 10 – Image hiding using a substitution matrix 13

Figure 11 – A three bits substitution matrix sample (k=3) 14

Figure 12 – A substitution matrix converting the color 0000 to 1111 and vice-versa ... 15

Figure 13 – Conversion of the solution S into an individual G 17

Figure 14 – Crossover of two individuals generating the offspring G1' and G2' 18

Figure 15 – Validation and fixing of invalid individuals G1' and G2' 18

Figure 16 – Mutation of an individual, exchanging the values of genes 19

Figure 17 – The path relinking process 20

Figure 18 – State array, input and output 23

Figure 19 – Add Round Key step: Each byte of the state is combined with a byte of the round sub-key, using the XOR operation 23

Figure 20 –Substitute Bytes step: Each byte in the state is replaced with its entry in a fixed 8-bit lookup table (S) 23

Figure 21 – Shift Rows step: Bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row 24

Figure 22 – Mix Columns step: Each column of the state is multiplied with a fixed polynomial c(x). 24

Figure 23 – The complete image steganography process 26

Figure 24 – The RGB cube, representing the color channels of the tri-dimensional substitution matrix 36

Figure 25 – CxImageHider software with an open image of Lena 38

Figure 26 – Dialog window asking for a file to be embedded inside the cover image .. 38

Figure 27 – Dialog showing PSNR value, bits per pixel used, total of bytes hidden and time elapsed in seconds after the image steganography 39

Figure 28 – Dialog window asking for genetic algorithm and path relinking parameters, and a stego key 39

Figure 29 – Dialog window showing results after image steganography with path relinking at each genetic algorithm generation..... 41

Figure 30 - Dialog window showing results after image steganography with path relinking after the end of genetic algorithm 42

Figure 31 – Dialog window asking for parameters to undo image steganography 42

Figure 32 – Dialog window showing undo steganography success 43

Figure 33 – Dialog window presenting MSE value between the two compared images 43

Figure 34 – grayscale Lena 512x512 pixels, Figure 35 – grayscale Baboon 512x512 pixels..... 49

Figure 36 – grayscale Text 512x512 pixels, Figure 37 – grayscale Jet 256x512 pixels, Figure 38 – grayscale Sailboat 256x512 pixels 49

Figure 39 – grayscale Tiffany 256x512 pixels, Figure 40 – grayscale Text 256x512 pixels, Figure 41 – grayscale Splash 256x512 pixels,..... 49

Figure 42 – grayscale Fishing Boat 256x512 pixels, Figure 43 – grayscale Peepers 512x512 pixels..... 50

Figure 44 – grayscale Barbara 512x512 pixels, Figure 45 – grayscale Jet 512x512 pixels 50

Figure 46 - grayscale Jet 256x512 pixels (squeezed), Figure 47 - grayscale Baboon 256x512 pixels (squeezed), Figure 48 - grayscale Peepers 256x512 pixels (squeezed), Figure 49 - grayscale Lena 256x512 pixels (squeezed) 50

Figure 50 – color Lena 512x512 pixels, Figure 51 – color Baboon 512x512 pixels 51

Figure 52 – color Text 512x512 pixels, Figure 53 – color Peepers 512x512 pixels..... 51

Figure 54 – color Barbara 512x512 pixels, Figure 55 – color Jet 512x512 pixels..... 51

Figure 56 – color House 512x512 pixels, Figure 57 – color Sailboat 512x512 pixels .. 52

Figure 58 – color Zelda 512x512 pixels, Figure 59 – color Tiffany 512x512 pixels..... 52

Figure 60 – color Pacman3D 640x480 pixels 52

Figure 61 – Dhrystones Benchmark of first computer system using SiSoftware Sandra Lite XI b 53

Figure 62 – Dhrystones Benchmark of second computer system using SiSoftware Sandra Lite XII SP1..... 54

Figure 63 – Lena stego image produced by image hiding with LSB substitution (some text appears in background)..... 60

Figure 64 – Lena stego image obtained by image hiding with path relinking (no background text and appears smoother) 61

Figure 67 – Baboon stego image produced by image hiding with genetic algorithm 66

Figure 68 – Baboon stego image obtained by image hiding with path relinking (not much visually different from Figure 43) 67

List of Tables

Table 1 – Quality comparison (PSNR values) with our results and those presented on reference [16] 57
Table 2 – Time spent in seconds with our tests 58
Table 3 – Time spent in BI units (billions of instructions) with our tests 59
Table 4 – Quality comparison (PSNR values) with our results and those presented on reference [13] 63
Table 5 – Time spent in seconds with our tests 64
Table 6 – Time spent in BI units (billions of instructions) with our tests 65
Table 7 – Quality comparison (PSNR values) with our results and those presented on reference [14] 69
Table 8 – Time spent in seconds with our tests 70
Table 9 – Time spent in BI units (billions of instructions) with our tests 71
Table 10 – Quality comparison (PSNR values) with our results and those presented on reference [17] 73
Table 11 – Time spent in seconds with our tests 74
Table 12 – Time spent in BI units (billions of instructions) with our tests 75
Table 13 – Quality comparison (PSNR values) with our results and those presented on reference [20] 77
Table 14 – Time spent in seconds with our tests 78
Table 15 – Time spent in BI units (billions of instructions) with our tests 79
Table 16 – Quality comparison (PSNR values) with our results and those presented on reference [37] 80
Table 17 – Time spent in seconds with our tests 81
Table 18 – Time spent in BI units (billions of instructions) with our tests 81
Table 19 – Quality comparison (PSNR values) with our results and [38] 82
Table 20 – Time spent in seconds with our tests 82
Table 21 – Time spent in BI units (billions of instructions) with our tests 83
Table 22 – Quality comparison (PSNR values) with our results and those presented on reference [39] 84
Table 23 – Time comparisons in seconds between our results and [39] 84
Table 24 – Time spent in BI units (billions of instructions) with our tests 84
Table 25 – Quality results (PSNR values), after hiding the executable file Pacman3D.exe inside Figure 60 85
Table 26 – Time results in seconds, after hiding the executable file Pacman3D.exe inside Figure 60 85
Table 27 – Time spent in BI units (billions of instructions) with our tests 85

Contents

1. Introduction 1
1.1 Past uses and curiosities 3
1.2 Motivation 4
1.3 Goals 5
1.4 Contribution 5
1.5 Work organization 6
2. Image Hiding and other concepts 7
2.1 Least significant bits (LSB) substitution 8
2.2 Image hiding by LSB substitution 9
2.2.1 Grayscale image hiding 9
2.2.2 Color image hiding 10
2.3. Substitution matrix 11
2.4 Genetic algorithm 16
2.5 Path relinking 19
2.6 Peak-to-signal noise ratio (PSNR) 20
2.7 The AES cryptosystem 22
3. The complete image steganography proposal 25
3.1 Cover image preparation 27
3.2 Secret data preparation 27
3.3 Search for a good solution 27
3.4 Data hiding 29
4. The novel use of path relinking refinement in image steganography 30
4.1 Path relinking use in grayscale image steganography 30
4.2 Path relinking use in color image steganography 35
4.2.1 The RGB cube: an alternative structure for constructing solutions 35
5. Implementation 37
5.1 The CxImage Class 37
5.2 The CxImageHider software 38
6. Experimental results 46

6.1 Previous works 47
6.2 The BI (billions of instructions) performance measurement concept 53
6.3 Comparison with simple LSB substitution and the genetic algorithm results..... 55
6.4 Comparison with modulus functions results 62
6.5 Comparison with dynamic programming strategy results 68
6.6 Comparison with cryptosystem and modulus operations results..... 71
6.7 Comparison with color quantization and DES cryptosystem 75
6.8 Comparison with difference expansion 79
6.9 Comparison with lossless block truncation coding 81
6.10 Comparison with binary space partitioning tree..... 83
6.11 Steganography of an executable file..... 85
7. Conclusions and future works 86
7.1 Conclusions 86
7.2 Future works 88
References 89

1. Introduction

Nowadays, at 21st century, the World Wide Web connects people all over the world for data transmission and the frequent use of e-commerce shops and banks, and security is needed more than never, to avoid the risk of information falling into wrong hands. In this scenario, steganography, the art of secret communication, plays an important role, by supporting and enhancing typical cryptography methods. Figures 1-3 show examples of historical paintings "hiding" some kind of information.

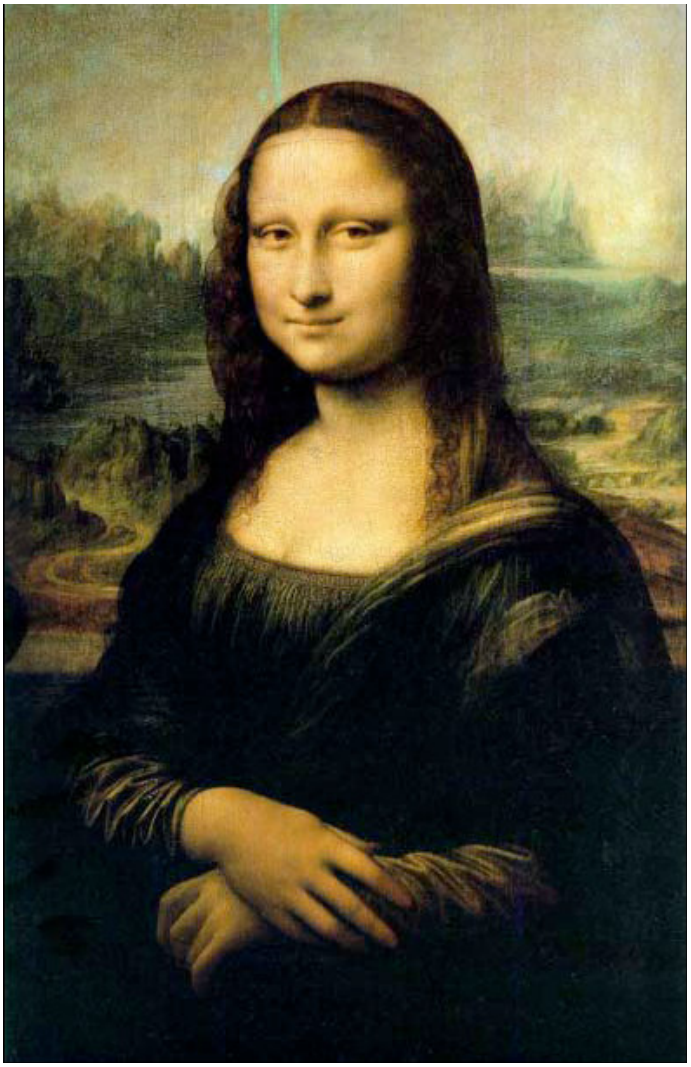


Figure 1 – Mona Lisa, by Da Vinci : Who is the person portrayed?

A lot of effort has already been taken on behalf of data protection. The easier and most common ways to do this are protect the data using passwords or data cryptography. These ways are somehow efficient in denying an intruder from getting important information. However,

they face some difficulties and vulnerabilities when trying to grant access to a new user on sending him a password to access the information.

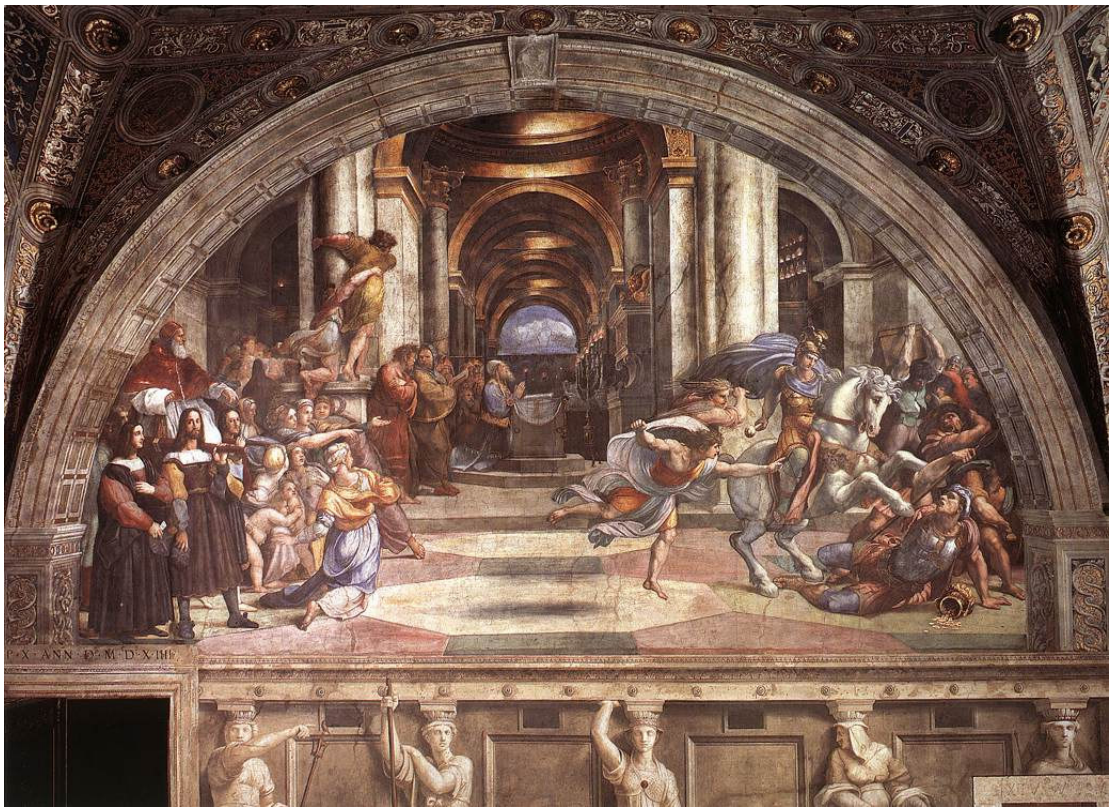


Figure 2 - Expulsion of Heliodorus from the Temple, by Raphael : What the public symbols inside the rooms means?

Figure 3 - Creation of Adam, by Michelangelo : Can you see the human brain?

Cryptography [1-4] can be used to transform the password into apparently meaningless information, protecting the relevant data even if it becomes captured during the transmission process. Many other forms of protection can be used, but most of them possess a significant drawback: the invader easily perceives or detect there is something being hidden. The simple

evidence of hiding may be sufficient for the grabbers to start trying to get the information. The image hiding [5,6] process does not present this disadvantage.

People typically send pictures inside the electronic mail. If the important data can be hidden inside a picture without degrading its quality in a perceivable level, crackers may not even notice there is important information inside the message. So, steganography protects not only the data, by encoding it inside the image [5], but it avoids also the risk of hackers catch the message and try to unlock the protections. If the image sent is a common picture, like a photo or something alike, the invader will probably not even notice there is hidden data inside and will not even begin to analyze the picture, thus protecting the message too. As the proverb: "Out of sight, out of mind".

There are many schemes to hide data inside an image. This work focuses in the use of genetic algorithm combined with the path relinking [7-10] refinement procedure to intensify the search for better solutions on color image steganography.

1.1 Past uses and curiosities

Steganography is very ancient, having notable reports dating from 6th century BC. On that epoch, according to Herodotus histories [11], Harpagus, a general from Median Empire sent a message hidden inside a belly of a dead hare to warn Cyrus II about the coming of a revolt against the current king of Media, Astyages. After receiving the message, Cyrus II united his forces with Harpagus and they won the battle. This resulted in Cyrus II becoming known as Cyrus the Great, the king of Persia, incorporating the Median Empire into the Persian Empire.

Some other noticeable historical feats can also be listed as acts of steganography [12]:

By the end of 6th century BC, in order to convince his allies that it was time to begin a revolt against Medes and the Persians, the Greek Histaieus shaved the head of his most trusted slave, tattooed the message on his head and waited until his hair grew back. After that, he sent him along to Greece hoping to encourage the Ionian tyrant of Miletto, Aristagoras, resulting later in the start of the Ionian Revolt against the Persians.

In the fifth century BC, Demeratus, a Greek exiled in the Persian Court, struggled to find a way of alerting Sparta that the Persian Great King Xerxes was gearing up to invade Greece. Knowing that any overt message would be intercepted easily by the Persians, he scraped off the wax surface of a wooden writing tablet and scratched his warning into the underlying wood. Demeratus then re-coated the tablet with a fresh layer of wax, thus allowing the apparently blank writing tablet to be carried off to Sparta without arousing suspicion.

Another famous case is the book *Hypteronomachia Poliphili* of 1499. Francesco Colonna, a Dominican monk, decided to declare his love to a young lady named Polia by putting the message "Brother Francesco Colonna passionately loves Polia" in the first letter of each chapter of his supposed book.

Another well known steganography method is lemon ink. Invisible when used to write a paper, it turns to brown after heating. Gallotanic acid, made from gall nuts, also becomes visible when coming in contact with copper sulfate. Several other interesting methods of cryptography and steganography were used along the time as well.

1.2 Motivation

Computer science is an interesting area of research because it provides help and support to all other areas, organizing, automating and enhancing many aspects of them. It is also very captivating to see the mutational aspect of this area, which never stays the same along the time, always requiring a great effort of the students and professionals to sharpen their knowledge and skills toward its constant evolution.

This dissertation can be considered hybrid, which makes this work yet more challenging, integrating information and expertise from some diverse sub-areas of the Computer Science: image processing, image analysis, computational intelligence, information security and cryptography. The work presented here follows a specific branch of the Computer Science called steganography, with several published papers, being the first dated from 1994 and entitled "Data Hiding". This branch is included in researches related to information hiding, and is closely aligned with watermarking, being similar to it in several aspects.

The majority of published works, like [13-17] focuses on using only gray images as cover images. Other ones, as [15, 17-20] propose color image hiding, but use the outdated DES cryptography algorithm. Another group, like [15, 18-19, 21-25] presents very good results but works with lossy compression of hidden data, which limits its applications and turns unviable message or file hiding. They are all good, but very specific and generally work on hiding only one data type, like images or text messages inside the cover image. None of them is said to allow the hiding of diverse medium inside the cover image, like a video, compressed file or yet an executable. This work is more general and presents good results with many of them.

1.3 Goals

This work proposal is to amplify the scope of appliances for image steganography, to allow the hiding of any digital media inside color cover images, typically used on the internet. The steganography methods are based on three main aspects to characterize them. These aspects are the capacity, security and robustness. Capacity is related to the quantity of data that can be hidden inside the media without degrading the quality of the image to a visually perceivable level. Security refers to the protection of data and the use of a cryptography algorithm to shield it against detection and hacking. Robustness is the resistance of the method against alterations done in the stego-media, preserving the secret content intact after media transformations.

This work focuses mainly on the aspects of capacity and security, once robustness always affects negatively the capacity aspect, often replicating hidden data to increase its resistance against transformations. Despite differing from watermarking, which generally focuses on robustness, most of this work can effectively be adapted to improve the capacity or security aspects of watermarking applications too. Others, like [26], focus on robust image hiding.

1.4 Contribution

The main contribution of this work relates to a novel adaptation of the general path relinking procedure, from the computational intelligence, to the scope of steganography, in image processing, to improve the results obtained with genetic algorithm. An interesting aspect

is this adaptation could be easily used with almost all other computational intelligence methods, like the GRASP [27] or Tabu Search [28, 29], for example. This gives a new way of possibilities for future works on steganography and other related or similar image processing areas.

Other contributions are the use of Advanced Encryption Standard (AES) cryptography algorithm [30-32] in color image steganography and the creation of a benchmarking measure called Billions of Instructions (BI) to allow the comparison of the speed and time results between tests done on different computer systems. This facilitates comparisons of results of this work with future ones, often running on faster computer systems with the quick evolution of technology.

1.5 Work organization

This dissertation is organized as follows: The second chapter explains the image steganography, outlining the typical applied methods to hide data inside an image: the LSB (least significant bits) substitution and the use of genetic algorithm to improve the LSB substitution performance and quality of final solutions. It also outlines the concepts of: Substitution Matrix, Picture Signal-to-Noise Ratio (PSNR), Mean Square Error (MSE), Path Relinking and the AES Cryptosystem. On chapter 3, a general approach for the adaptation of the Path Relinking procedure to be used in conjunction with the genetic algorithm and the LSB substitution techniques. On chapter 4, the novel refinement introduced on chapter 3 is described in details, a step by step guide to the improvement. Chapter 5 presents implementation details, describing the main classes and structures adopted to build a prototype. In chapter 6, the Billions of Instructions (BI) concept is shown along with the comparisons with five papers, demonstrating the quality and speed results and also showing the values for parameters and configurations used in the tests. Is also includes some discussion about the image hiding methods used in these works and presents some comments about a sixth paper. Finally, on chapter 7, the conclusions and achievements to this study, and some future ideas are discussed.

2. Image Hiding and other concepts

Image Hiding is a specific branch of Steganography and has the main goal of keeping the communication process as a secret by concealing the information inside a digital picture. This is different than other branches, like Cryptography, whose objective is to protect the information by turning data unintelligible to the ones without permissions to access it.

Some definitions are typically used in an image hiding process. They are the secret data, the cover image, the stego-image and the stego-key.

Secret data or embedding data is the message or information to be hidden.

Cover image is the image used to "cover" or conceal the secret data in an image hiding process.

Stego image is the image containing the secret data after the image hiding process. The word "stego" comes from steganography. It carries the secret data inside.

Stego key is the cryptography key when the cryptography process is applied in image hiding, to increase the information security and protection of the embedded data.

The Figure 4 helps to illustrate the image hiding terminology:

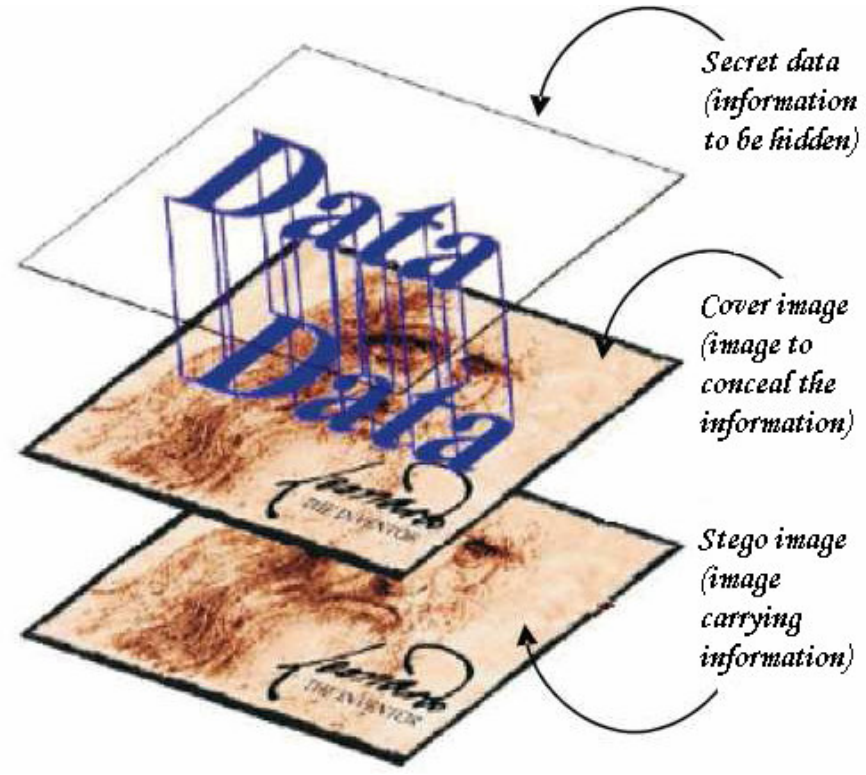


Figure 4 – Image hiding example

2.1 Least significant bits (LSB) substitution

The least significant bits (LSB) substitution is the process of replacing the values of the less important bits composing each pixel of an image. It has many uses in image analysis. One of his most common applications is the threshold of digital images, but can be used in steganography as well.

Digital images are composed and represented by a 2D array of pixels. Each value of this array is formed by a number of bits, usually eight bits for a grayscale or thirty two bits in a color image. These values represent either the existing colors or intensities of gray. In eight bits grayscale images, for example, the bits 00000000 represent the black color and the bits 11111111 represent the white color. Any other values ranging between black and white will be shades of gray. It is important to note that the gray 00000001 is very different from 10000000. In fact, 00000001 ($2^0 = 1$) is almost a black, while 10000000 ($2^7 = 128$) is a midrange gray. This happens because the bits located at right are the least significant ones (LSBs), so their values influences the pixel tone only a little. The bits located at left, otherwise, influence a lot more, so one of these is sole sufficient to turn the pixel intensity to midrange gray. The Figure 5 helps to illustrate the least significant bits concept.

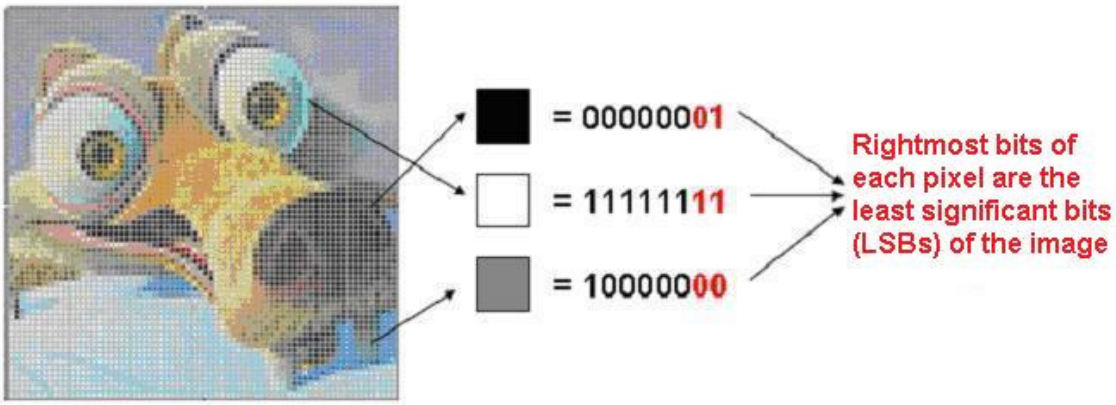


Figure 5 – Binary representation of black, white and gray colors, evidencing its least significant bits position (rightmost bits)

For example, consider a pixel of a grayscale image, whose bits are 00000111, and replace only its two least significant bits values to zero. The result will be the value (00000100) after the LSB substitution, which for human eyes is almost the same color.

2.2 Image hiding by LSB substitution

When applying LSB substitution in image hiding, the least significant bits in each pixel of the cover image (C) are used to store the information to be hidden. The values of these bits will be changed to store the information inside the cover image (C).

2.2.1 Grayscale image hiding

To hide information inside a grayscale image using LSB substitution, the following steps must be considered:

Step 1: Determine how many least significant bits (k) will be needed to store the information;

Step 2: Extract the least significant bits of the cover image (C) to generate a residual image (R);

Step 3: Convert the information to be hidden (E) into an image to hide (E');

Step 4: Replace of the residual image (R) by the image to hide (E'), producing the stego image (Z).

To discover how many least significant bits (k) will be needed to store the information to be hidden (E) inside a cover image (C) it is necessary to divide the size of the information to be hidden (E) by the size of the cover image (C). The result obtained is a percentage of bits that will be needed for embedding the information (E) into the cover image (C). For example, hiding an information of 128 Kb inside a cover image of 512 Kb, will use 128/512 = ¼ = 25 % of the bits in the cover image. Supposing the cover image is a grayscale (8 bits), such embedding process will use only the two least significant bits to embed data (k=2). As a rule, the hidden data size shall not exceed one half of the cover image (C) size, or the degradation level in the cover image (C) will get too high.

In Step 2, each pixel of the cover image (C) must be analyzed, extracting from them only a determined number of least significant bits (k), sufficient to hide the information. These least significant bits extracted from the cover image (C) will be stored as pixels, composing a new image, called the residual image (R). After the extraction, this residual image (R) will have the

same number of pixels than the cover image (C), but fewer bits per pixel, once only the k least significant bits will be extracted to form its pixels, as shown on Figure 6.

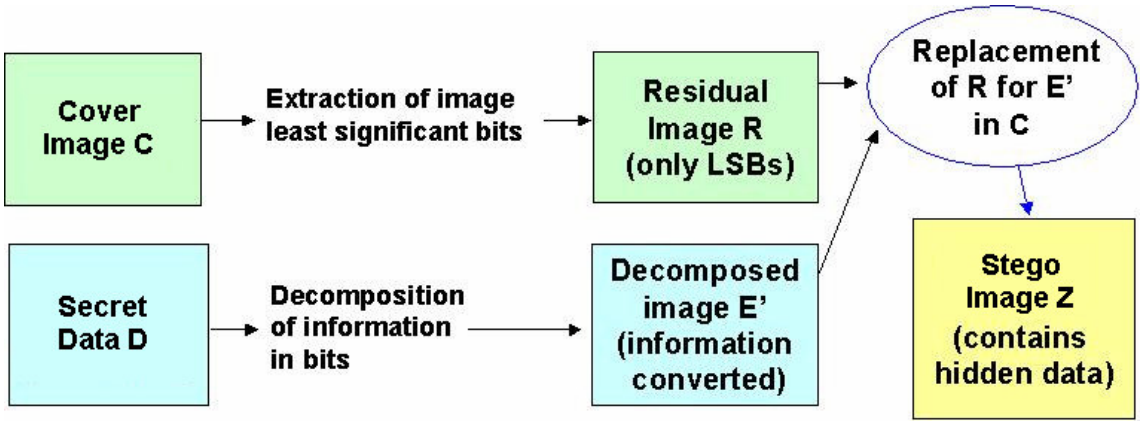


Figure 6 – Image hiding by least significant bits substitution

In Step 3, the information to be hidden (E) must be converted to the same depth (same number of bits per pixel) than the residual image (R), producing an image to hide (E'). Each pixel in the information to be hidden (E) will correspond to one or more pixels inside the image to hide (E'), depending on the number of bits per pixel of the existing residual image (R). See Figure 6.

In Step 4, after creating the image to hide (E') on the same depth than the residual image (R), it must be replaced by (E') inside the cover image (C), producing the stego image (Z), as shown on Figure 6.

On the threshold process, instead replacing the values of the bits, some specific bit positions inside the image are normally reset (these values are set to zero). This shows details in the image like contours or yet modifies its appearance.

For example, suppose grouping near black tones inside an image by resetting the four bits most at right in a grayscale image, it turns all the pixels with near black to absolute black (ex. 00000111 turns to 00000000).

2.2.2 Color image hiding

The steganography using color images as cover images is similar to the process in grayscale images, detailed in previous sub-section 2.2.1. The main difference is instead having only one 8-bits grayscale intensity channel to hide data, as happens in grayscale images, a total

of three 8-bits color intensity channels are available, each one for its corresponding color (red, green and blue). So, the hidden information can be distributed among these three color channels, increasing up to three times the capacity of data hidden inside the image.

The calculations of how many bits per pixel will be used to hide data remains the same. However, these bits must be distributed most equally possible among the red, blue and green color channels. For an example, if the calculations determine 5 bits of each pixel inside the color image will be necessary to hide data, then we will use 2 of 8 bits from the red color channel, 2 of 8 bits from the blue channel and 1 of 8 bits from the green channel. The green channel is always the last one to be used to hide data, because human sensibility to green color change is higher that the other two colors. So, the order of bit usage in color channels will be first red, then blue and finally green. The remaining of the process is the same, always considering three color channels, instead of one grayscale channel.

2.3. Substitution matrix

By using only the LSB substitution, a good speed on image hiding is achieved, but it cannot guarantee a good final image, that is without noticeable degradation. This can be seen easily with the following example: Let us suppose an extreme case, where a completely black image of 8 bits per pixel, with 512x 512 pixels (all of them 00000000 = true black) is used as the cover image (C) and a completely white image (E) is the object for embedding. Consider E composed of 8 bits per pixel and 256 x 512 pixels (all of them 11111111 = white). The main objective is to hide the white image (E) inside the black cover image (C) without loosing too much quality. By using the method explained in section 2, we must:

1) Determine how many least significant bits will be used from the cover image:

256 x 512 / 512 x 512 = ½ → 50% of 8 bits → 4 least significant bits used (k=4)

2) Extract the residual image (R) from the cover image (C):

Since all the pixels in the cover image (C) are true black (00000000), when the 4 least significant bits of each pixel are extracted to construct the residual image (R), this will produce

a residual image of 4 bits per pixel with 512x 512 pixels (all of them = 0000, so black too). Figure 7 shows this.

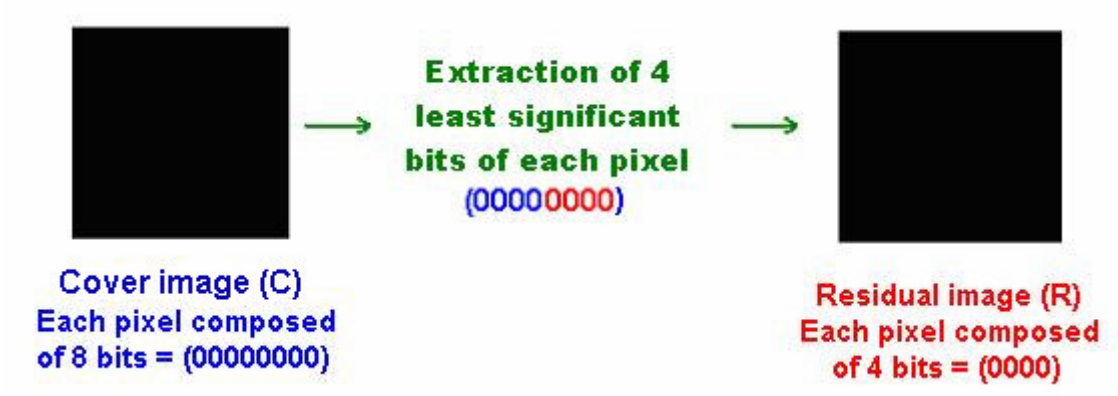


Figure 7 – Extraction of residual image (R)

3) Convert the information to be hidden (E) into the same format of the residual image (4 bits per pixel), producing the image to hide (E'):

Since all the pixels in the information to be hidden (E) are white (11111111), each 8 bits pixel in (E) will generate two 4 bits pixels (1111) in the image to hide (E'). As a result, it will produce an image to hide (E') of 4 bits per pixel with 512 x 512 pixels (all of them = 1111, so only white pixels). Figure 8 helps to illustrate this.

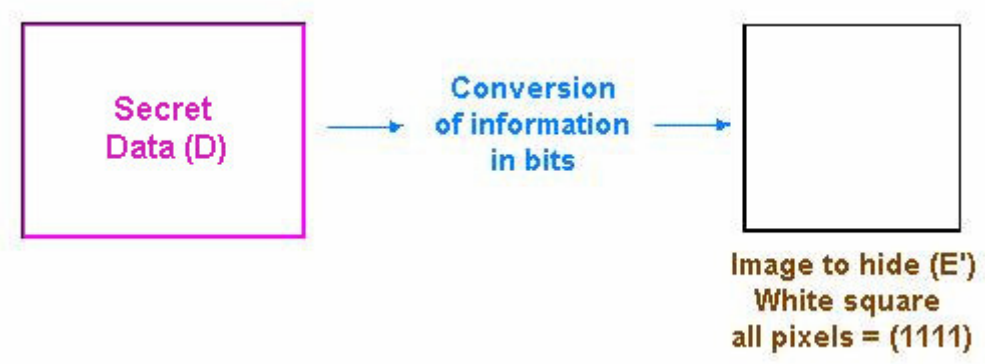


Figure 8 – Generating the image to hide (E')

4) Replace the residual image (R) by image to hide (E') in the cover image (C), generating the stego image (Z):

The pixels of the residual image (R) (all = 1111) must be replaced by the pixels of the image to hide (E') (all = 0000), substituting the values present in the 4 least significant bits inside each pixel of the cover image (C). This will produce a stego image (Z) of 8 bits per pixel with 512x 512 pixels, whose pixels are all 00001111 (gray). Here, a great difference can be seen

between the cover image (C), composed of pixels 00000000 (black) and the stego image (Z), constructed with 00001111 pixels (gray). That difference degrades significantly the quality of the final result. This is shown on Figure 9.

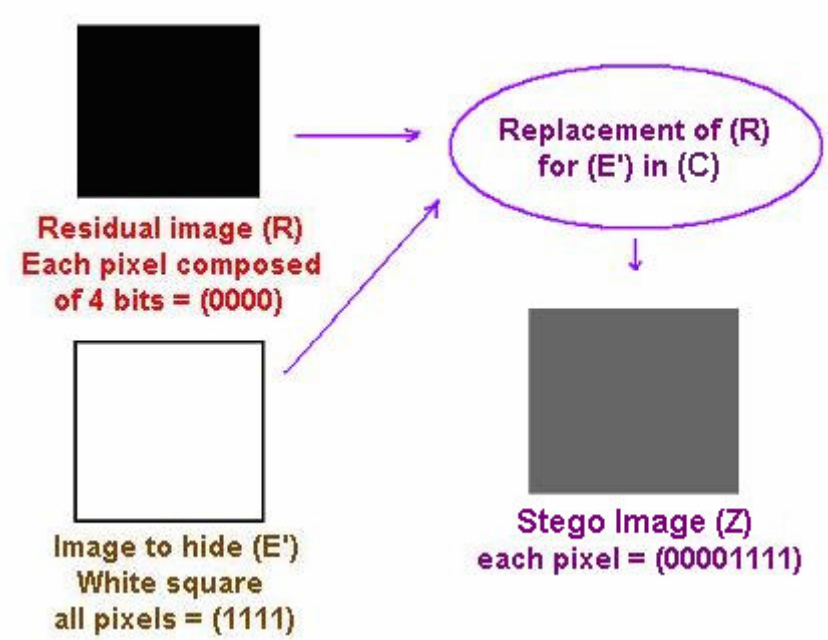


Figure 9 – Stego image (Z) generation by replacement of residual image (R) by image to hide (E') in cover image (C)

This problem can be overcome by using a substitution matrix. The substitution matrix is used to convert some colors or intensities of the image to hide (E') into other ones, producing a converted image to hide (E''). The idea is produce an image (E'') most similar to the residual image (R). So, the replacement of the residual image (R) by (E'') instead of (E') will be smoother, improving the quality of the stego image (Z). The whole scheme is shown on Figure 10.

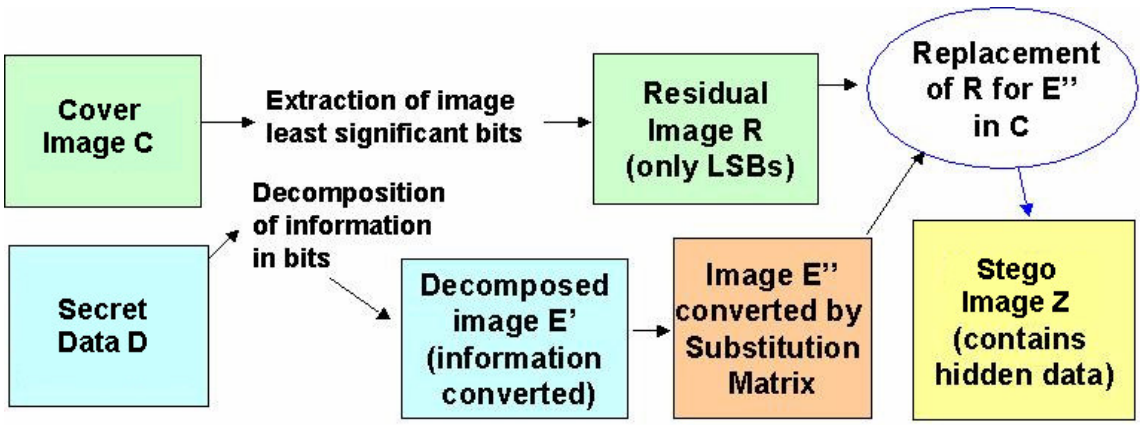


Figure 10 – Image hiding using a substitution matrix

The structure of the substitution matrix is as follows:

Each row in the matrix is an original color;

Each column in the matrix is a resulting color;

Each value inside a coordinate [row, column] of the matrix can be a zero (0) or a one (1), where the value one (1) indicates the row color that will be converted into the column color. A one (1) value can never appear more than once for each given row or column of the matrix. (ex. a single color can not be converted into other two different colors or vice-versa).

The number of rows and columns in the substitution matrix will always be 2^k, where (k) is the number of least significant bits substituted in the process. So when three least significant bits are replaced (k = 3), this will produce a substitution matrix of size 2^3, an 8 x 8 substitution matrix. An example can be viewed on Figure 11.

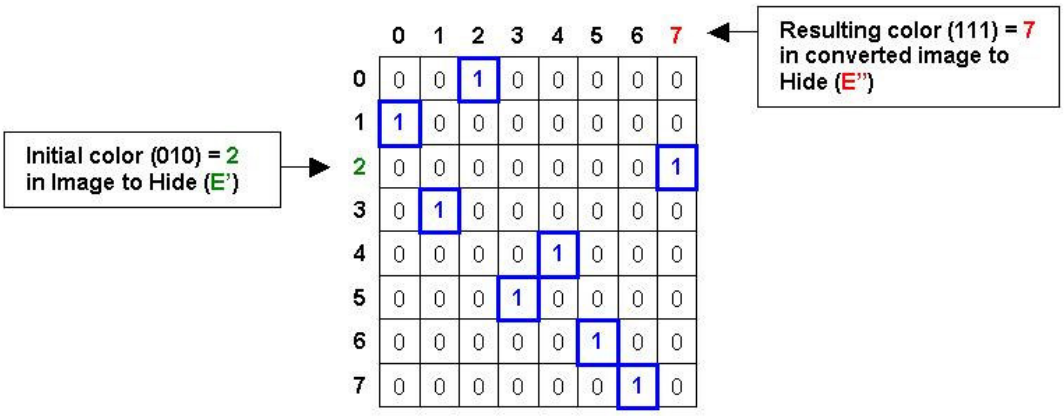


Figure 11 – A three bits substitution matrix sample (k=3)

Note that this substitution matrix must also be incorporated into the information to be hidden (E), so the receptor of the stego image (Z) can revert the image hiding process afterwards to obtain the desired information.

Now, coming back to the substitution problem, after a white image (E') is hidden inside a black image (H) resulting in a gray image (Z), a substitution matrix can be used to improve the quality of the stego image (Z) in the following way: The 4 least significant bits will be substituted (k=4), generating a 16 x 16 substitution matrix. This matrix can be something like the one shown on Figure 12:

		RESULTING COLOR				
		0000	0001	...	1110	1111
ORIGINAL COLOR	0000	0	0	0	0	1
	0001	0	1	0	0	0
	...	0	0	1	0	0
	1110	0	0	0	1	0
	1111	1	0	0	0	0

Figure 12 – A substitution matrix converting the color 0000 to 1111 and vice-versa

By applying this substitution matrix in the image to hide (E'), every color will be maintained, except:

Color 0000 → will be converted to 1111

Color 1111 → will be converted to 0000

The image to hide (E') has the color 1111 in all of its pixels, so after applying the substitution matrix above, it will produce a converted image to hide (E'') with all pixels of color 0000!

Following the process, the residual image (R) having its pixels values 0000 is replaced by the converted image to hide (E''), also composed only by pixels of values 0000, producing the stego image (Z), full of pixels 00000000, and equal to the cover image (C), thus obtaining the best possible quality!

Of course this situation only illustrates an extreme case where all pixels of the image to hide (E') had the same value (1111), thus making it possible.

In the problem above, the optimal substitution matrix to be used was easily found, because both the entire image to hide (E') and the cover image (C) were composed of only one color. Typical cover images (C) and images to hide (E') are composed of several colors or intensities which leads to several possible substitution matrices, so the challenge is to choose a good one. To be more exact, the total number of possible substitution matrices generated is $2^k!$, where k is the number of least substitution bits used to hide data in the cover image (C) and ! is the factorial operation. So, if only two bits are used to hide data (k=2), there will be $2^2! = 4! = 4$

x 3 x 2 x 1 = 24 possible substitution matrices, so it is easy to determine which one is the best by testing all of them. Otherwise, for greater k values, like k = 4, the number of possibilities become huge, $2^4! = 16!$ (more than 2,000,000,000). For these cases ($k \geq 3$), the wide range of possibilities turns the time for determine the optimal substitution enormous, so the use of a genetic algorithm becomes an interesting approach to help in choosing a near-optimal substitution matrix.

2.4 Genetic algorithm

The image hiding by genetic algorithm [16] process, has an additional step over the existing LSB substitution, that is the choice of a good substitution matrix to be applied into converting the image to hide (E'), generating the converted image to hide (E'') to replace the residual image (R), and produce the stego image (Z).

In color image steganography, each individual or specific solution present in genetic algorithm is composed by a total of three color substitution matrices, each one for its respective RGB color channel (red, green and blue substitution matrices). In grayscale steganography, only one substitution matrix composes the individual or solution.

To use a genetic algorithm it is necessary to convert the solutions (one or three substitution matrices, one for grayscale or three for color cover images) into a format that can be handled by the algorithm: an individual that is a gene vector. It is a simple process: given the one (1) value appears only once for each given row/column in the solution, the substitution matrix can be converted into a vector, where each position of the vector corresponding to a row in the matrix and the values inside the vector positions (genes) represent each column index used in the substitution matrix. This conversion is shown on Figure 13.

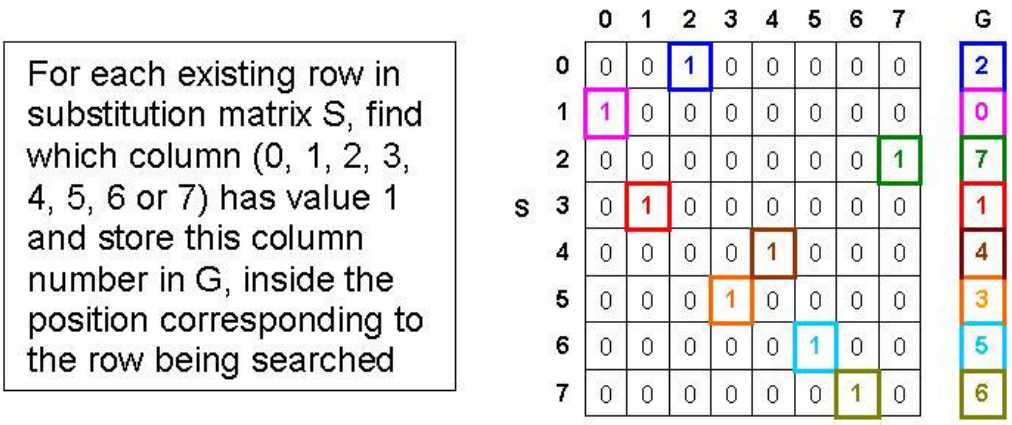


Figure 13 – Conversion of the solution S into an individual G

The genetic algorithm can be detailed in the following steps:

Step 1: Generation of an initial population;

Step 2: Combination of fragments of existing individuals to produce new ones (crossover);

Step 3: Mutation of existing individuals, generating new ones;

Step 4: Choice of the fittest individuals between all obtained in this generation, to form the population for the next generation, until some stopping criterion apply (ex. maximum number of generations, individuals or elapsed time).

The initial population to start the algorithm (step 1) can be obtained by several ways, like a heuristic algorithm, or generating random individuals and so on. The genetic algorithm in [16] generates a set of 10 random individuals to its initial population.

The crossover (step 2) is a well known method of combining individuals, where some criterion splits two or more individuals into parts and one or more parts of each individual are combined together to form new individuals.

In [16], the crossover adopted was a random choice of two individuals (parents) G1 and G2 to be combined. Each individual is divided in two equal parts and the first part of G1 is joined with the second part of G2 and vice-versa, composing two new individuals (offspring), G1' and G2'. Figure 14 shows this.

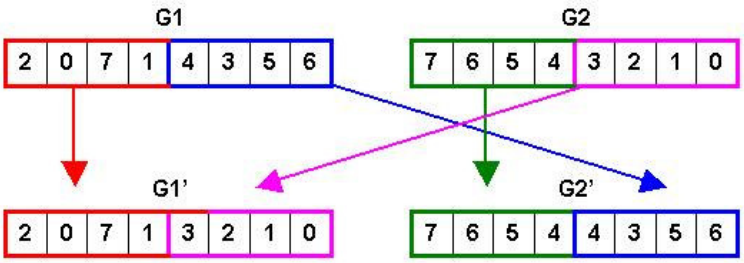


Figure 14 – Crossover of two individuals generating the offspring G1' and G2'

After the crossover, the offspring must be validated, because they may contain some repeated genes and be missing others. In these cases, these repetitions must be fixed by replacing the repeating genes by the missing ones. This validation/fix process occurs in the following way (see Figure 15):

Check each gene of an offspring and form a list of found genes. If a repeated gene is found (appears a second time in the individual) this value is substituted by -1 to mark a vacant position in the individual;

Analyze the list of found genes, verifying which ones are missing and then put these missing values in order in each vacant position in the individual (marked by value -1).

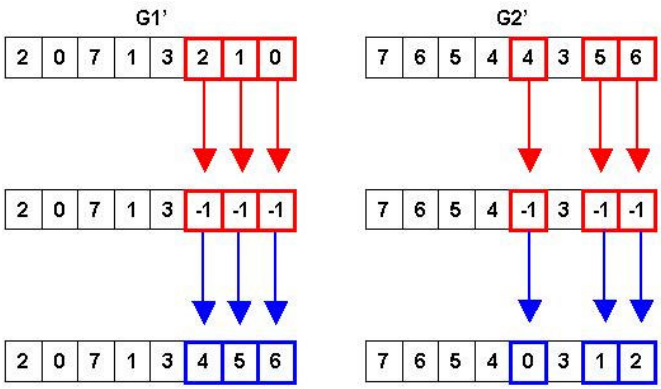


Figure 15 – Validation and fixing of invalid individuals G1' and G2'

This crossover process was repeated 10 times, combining the 10 random chosen pairs of individuals to form the offspring. Note the same individual can participate more than once in the whole process (he can be chosen several times).

The mutation itself (step 3) is very interesting because it helps the genetic algorithm to avoid falling into the local minima problem. It becomes more important when the crossover process starts to combine very similar individuals, generating offspring almost equal to its

parents and hanging the evolution/search of the genetic algorithm. However, the mutation should be applied in a small rate, to not compromise what was found until then.

In [16], the mutation process is concluded with the following steps (see Figure 16):

Step 1: Application of a mutation rate of 0.1 into the population, so each individual in the population has a 10% chance of being mutated;

Step 2: If the individual is chosen for mutation, two genes (g1 and g2) of this individual are chosen to exchange their values, so the gene g1 stays with value of g2 and vice-versa. Note that this process does not require any validation, since the values of the genes are permuted and there is no chance of generating an invalid individual afterwards.

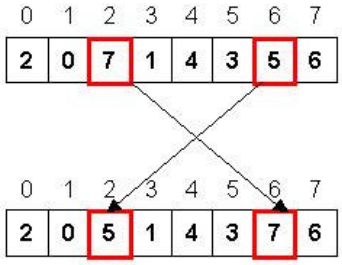


Figure 16 – Mutation of an individual, exchanging the values of genes

After concluding the steps above, producing a population of individual candidates, it is necessary to choose some among them to continue the process. These choices can be done by several ways, like adopting an elitist criterion of selecting only the best part of this population (10% best individuals, for example) or giving a probability of survival to each individual based on its fitness value.

In [16], an elitist criterion was utilized, selecting only the 10 best individuals to proceed into the next generation, discarding the others. The fitness of each individual can be measured by quantitative criterions, like the PSNR (Peak Signal-to-Noise Ratio). They adopted a stopping criterion of 8 generations.

2.5 Path relinking

The path relinking, proposed by [8], is a very interesting method of local search, and can be applied to a wide range of applications.

Virtually, almost any method that works with an elite candidates list or a selection of best solutions, like Greedy Randomized Adaptive Search Procedure (GRASP) [27], Tabu Search [28, 29], Variable Neighborhood Search (VNS) [33, 34] and many others, like [35-36], may benefit from this refinement step.

The Path Relinking starts by choosing two among the solutions present in the best solutions list and electing one of them as the start solution (S1) and the other one as the guiding solution (S2). Note the ideal condition is to choose two very different solutions from the list, allowing a greater diversity on the solutions generated on the path relinking process and providing a bigger chance to escape from local minima.

The next step is to verify the differences between S1 and S2, and gradually starting to modify parts of S1 turning it more similar to S2 with each change, and "take a picture" of S1 after each change is done to map these newly found solutions as intermediary solutions, keeping only the ones that excel S1 and S2 in aptitude. The main idea is shown on Figure 17.

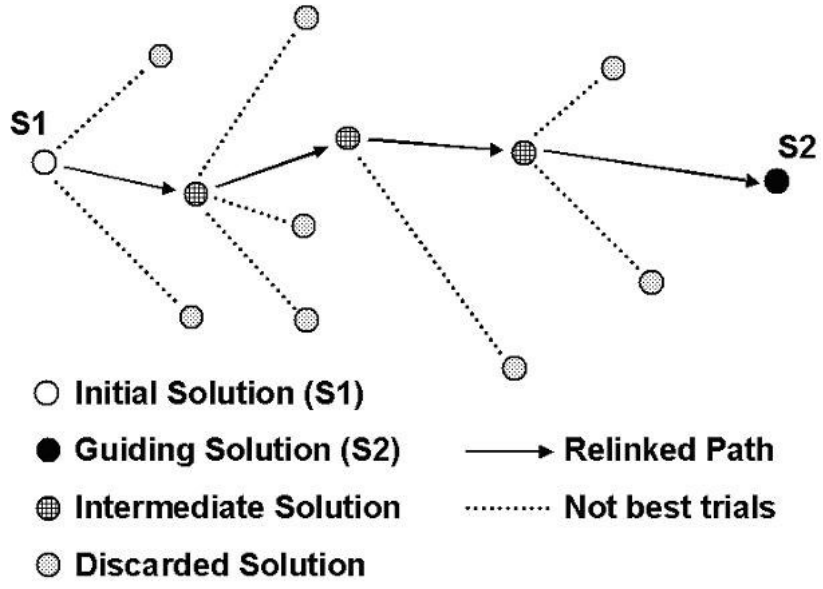


Figure 17 – The path relinking process

2.6 Peak-to-signal noise ratio (PSNR)

The PSNR is a simple way to evaluate loss of quality in image analysis and can be calculated by comparing two instances of the same image: the cover image (C) and the stego

image (Z). Alternatively, this value can also be calculated qualitatively by the comparison between the residual image (R) and the converted image to hide (E'').

Before calculating the PSNR value, it is necessary to obtain the MSE (mean square error) between the two images. The MSE value can be obtained by taking each pixel of the first image with its corresponding one in the second image, and subtracting their intensity values (subtracting the image 1 pixel intensity value from the image 2 pixel intensity value). This result is powered by 2 and added to the MSE total value, so the MSE is a summation of powered by 2 differences between the intensities of the pixels from both images.

The MSE equation is:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left(I(i,j) - K(i,j)\right)^2$$

Where *m* is the total of image rows, and *n* the total of image columns (m x n = total of pixels in the image) and *I*, *K* are the compared images.

The PSNR formula is:

$$PSNR = 20.\log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

Where MAX_I is the maximum possible intensity or color value. In an image of 8 bit pixels, the MAX_I value would be 255, since the pixel values would range from 0 (00000000) to 255 (11111111). PSNR values typically range from 0 to 100. Minimum possible value is always zero, while maximum value is correlated to the total number of pixels inside the image, being 102.3162 for an image with 512x512 pixels. A good target PSNR value could be the one assuming an intensity difference of only 1 inside each pixel, which would result in a PSNR value of 48.1308, for an image with 512x512 pixels.

For color images, according to [20], the MSE is the sum over all squared value differences obtained from each color channel present in each pixel from the images. This sum is then divided by the image size and by the number of color channels present. In RGB color system, for instance, there are 3 color channels: red, green and blue.

2.7 The AES cryptosystem

The Advanced Encryption Standard (AES) [30], also known as Rijndael [31], is a block cipher adopted as an encryption standard by the U.S. government. It has been analyzed extensively and is now used worldwide, as was the case with its predecessor, the Data Encryption Standard (DES). AES was announced by National Institute of Standards and Technology (NIST) as U.S. on 2001, after a 5-year standardization process, becoming effective as a standard on 2002, being one of the most popular algorithms used in symmetric key cryptography. It is available by choice in many different encryption packages.

The cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and submitted to the AES selection process under the name "Rijndael", a portmanteau of the names of the inventors.

The encryption process converts data to an unintelligible form called ciphertext. Decrypting the ciphertext afterwards, converts the data back into its original form, called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. The number of rounds required to cipher or decipher a data block is directly related to the crypt key size, being 10, 12 or 14 rounds.

Most mathematical operations of this algorithm use the finite field or Galois field GF (2^8) concept, where a limited number of elements exist and all calculations performed inside the field will result in an element within that field.

The simplified structure of the AES algorithm [32] to cipher a block can be depicted below:

1) Key Expansion - Generate a series of cryptography keys (called Round Keys) to be used in the following rounds, derived from the provided Cipher Key.

2) Initial Round:

2.1) Copy of bytes from the block to be encrypted to a matrix structure called State, where the subsequent operations will work. Figure 18 shows this process.

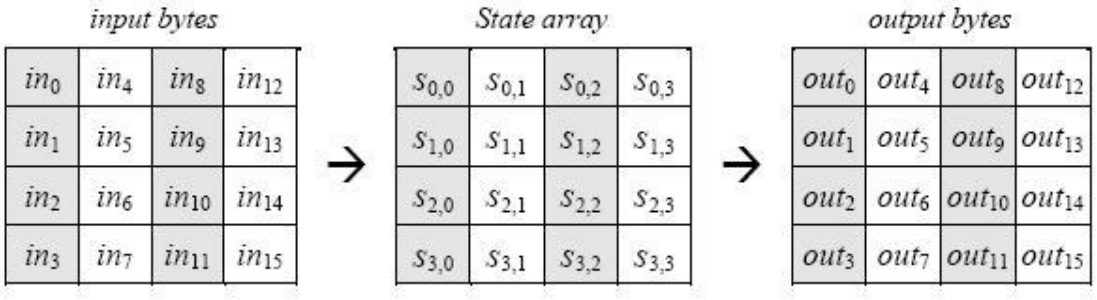


Figure 18 – State array, input and output

2.2) Add Round Key - The combination of the current Round Key with each byte present inside the State. The combination is done by XOR operations (see Figure 19).

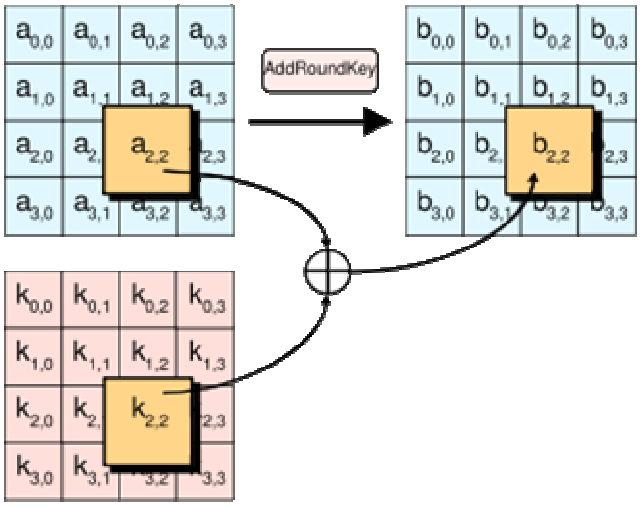


Figure 19 – Add Round Key step: Each byte of the state is combined with a byte of the round sub-key, using the XOR operation

3) Other subsequent Rounds:

3.1) Substitute Bytes - Transformation in the Cipher that processes the State using a nonlinear byte substitution table called S-box, that operates on each of the State bytes independently (see Figure 20);

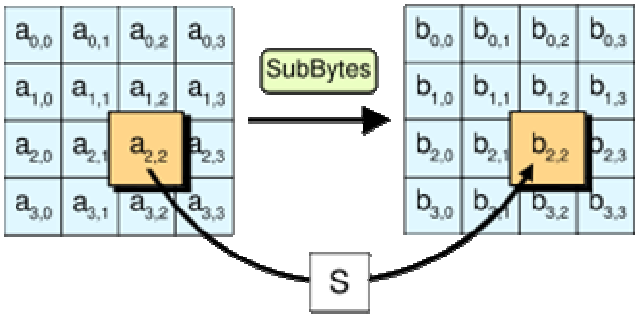


Figure 20 –Substitute Bytes step: Each byte in the state is replaced with its entry in a fixed 8-bit lookup table (S)

3.2) Shift Rows - Transformation in the Cipher that processes the State by cyclically shifting the last three rows of the State by different offsets (see Figure 21);

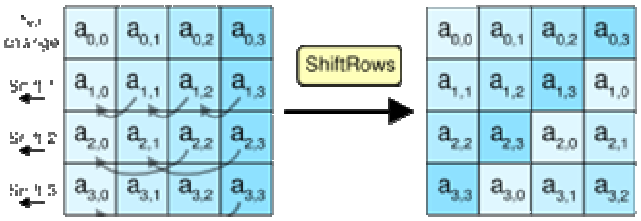


Figure 21 – Shift Rows step: Bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row

3.3) Mix Columns - Transformation in the Cipher that takes all of the columns of the State and mixes their data (independently of one another) to produce new columns (see Figure 22);

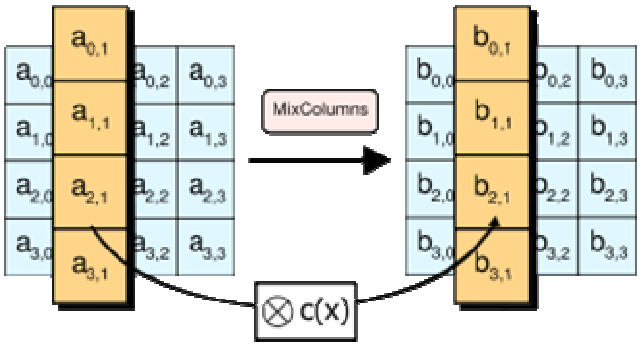


Figure 22 – Mix Columns step: Each column of the state is multiplied with a fixed polynomial c(x).

3.4) Add Round Key - The combination of the current Round Key with each byte present inside the State. The combination is done by XOR operations (see Figure 19).

4) Final Round - similar to the other rounds, but without the Mix Columns step.

The structure to decipher an encrypted block is very similar, having the corresponding inverse forms of Substitute bytes, Shift Rows and Mix Columns operations taking the place of the original ones.

3. The complete image steganography proposal

In order to improve the already known LSB substitution method, several additions were incorporated to the process:

To improve the quality of the stego image (Z), the genetic algorithm and the path relinking methods were included. These search for a good solution to convert the information to be hidden, lessening the stego image (Z) degradation after the embedding of the information inside the cover image (C).

To improve the security, the AES cryptosystem was implemented and a new step was inserted immediately before the decomposition of information in bits. This ensures the information being embedded into the cover image (C) is protected by a powerful cryptography method, eliminating the chance of data recovery by unauthorized people.

To improve the embedding capacity, the genetic algorithm was improved to work on three RGB color channels (red, green and blue), thus allowing the embedding of information inside color images and tripling the total capacity of data hiding when compared to grayscale image steganography.

To allow greater flexibility, a binary conversion of all information to be hidden was added. This generalizes and amplifies the image steganography scope of appliances, presenting the possibility to hide any kind of media inside the image.

Figure 23 helps to illustrate the complete process, by showing all the steps of complete image steganography.

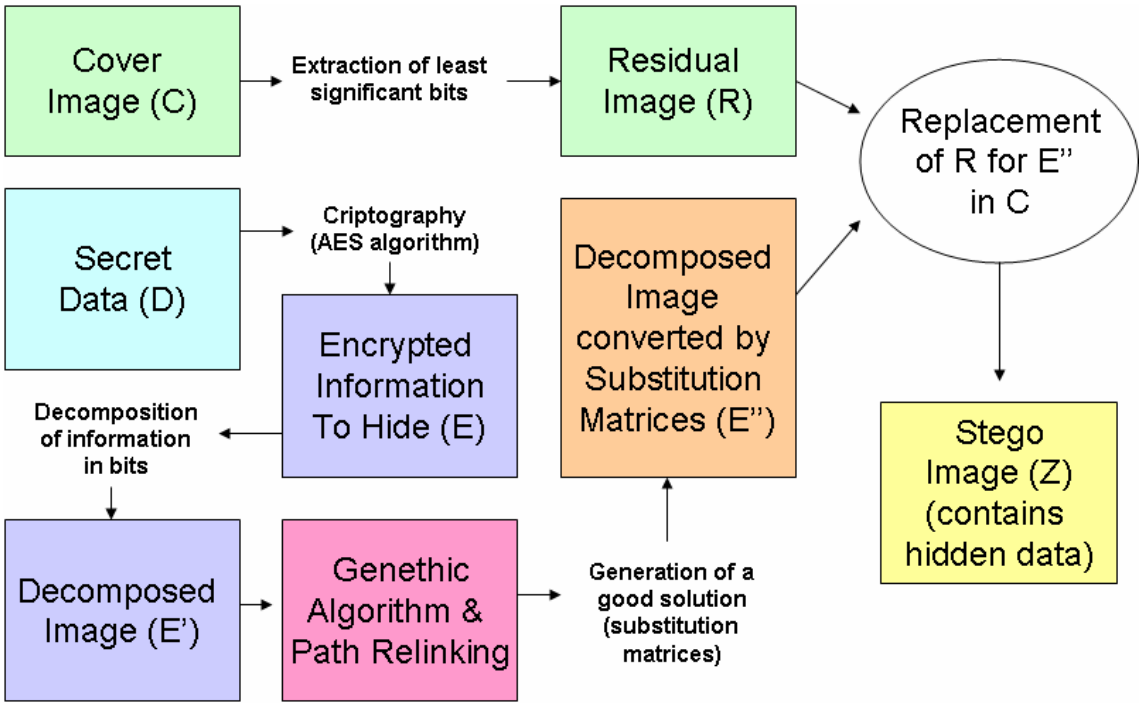


Figure 23 – The complete image steganography process

The complete image steganography process can be subdivided in three main phases, totalizing seven steps:

STEP 1 – Extraction of the significant bits from the Cover Image (C), generating the Residual Image (R);

STEP 2 – Cryptography of the Secret data (D) using the AES algorithm;

STEP 3 – Decomposition of the Encrypted Secret data (D) in bits, producing the Decomposed Image (E');

STEP 4 – Use of genetic algorithm to search for a good solution to convert the Decomposed Image (E');

STEP 5 – Use of path relinking refinement to combine the best individuals (solutions) found by each generation of genetic algorithm (step 4), in order to search for a better solution;

STEP 6 – Conversion of the Decomposed Image (E') by the best solution found in step 5, generating the Decomposed Image Converted by Substitution Matrices (E'');

STEP 7 – Replacement of Residual Image (R) by Decomposed Image Converted by Substitution Matrices (E'') inside the Cover Image (C), producing the Stego image (Z).

3.1 Cover image preparation

The first phase is responsible to prepare the cover image to receive the hidden data, by extracting from the cover image the bytes that will be replaced. This phase is represented by the green boxes in Figure 23 (two top boxes), and comprehends the STEP 1 from the complete process.

The complete image steganography process starts by extraction of the least significant bits of the cover image (C), in order to produce the residual image (R) (STEP 1). Before this, a calculation of how many bits are necessary to hide the information (I) inside the cover image (C) must be done. This process is detailed in sub-section 2.2.1, steps 1 and 2.

3.2 Secret data preparation

The second phase treats the data to be hidden, encrypting it to secure the information against possible intruders and arranging the encrypted data in a format it can easily be hidden inside the cover image.

It is necessary to protect the secret data (D) by the use of AES cryptosystem (STEP 2). The secret data (D) can be any digital medium: an image, a text file, a compressed file, etcetera. This information is copied to an array of bytes, the information vector (V). This vector is submitted to the AES cryptosystem together with the cipher key, resulting in the encrypted secret data (D), an array of the same size than (V). The cipher key size can be 128, 192 or 256 bits.

The encrypted secret data (D) must be decomposed to an image of the same depth than the residual image (R) (STEP 3), producing the decomposed image (E'). This is similar to the procedure outlined in sub-section 2.2.1, step 3.

3.3 Search for a good solution

The third phase of the process comprehends the use of several optimization methods to search for a better solution to convert the data to be hidden, so it reduces the degradation level of the stego image.

It is necessary to use a genetic algorithm (STEP 4) to find a good solution, so the decomposed image (E') can be converted in order to improve the stego image (Z) quality. The use of a genetic algorithm can be seen in details in section 2.4.

The parameters adopted for the genetic algorithm were the same suggested by [16], considering a total of 8 generations. The initial population is composed of 10 individuals generated randomly. The crossover process works by selecting copies of 10 pairs of individuals chosen randomly among the initial population of the current generation, producing 10 crossed individuals. The mutation process operates on copies of the individuals from initial population of the current generation with a mutation rate of 0.1, checking a 10% of chance of each gene of each individual suffering a random gene swapping, generating 10 mutated individuals. Finally, a selection of the 10 best individuals among the 10 individuals from initial generation population, the 20 produced by crossing and the 10 ones from mutation is done to compose the next generation initial population. The selection process evaluates the quality of solutions by PSNR (peak-to-signal noise ratio) comparison between the cover image (C) and a temporary stego image obtained after applying each solution over a copy of the cover image (C). The solutions presenting the higher PSNR values stay in elite group.

In order to improve yet more the search process, the elite group (10 best individuals found in each generation of the genetic algorithm) is recombined by the path relinking process (STEP 5) to produce superior individuals. The path relinking starts by selecting randomly a pair of individuals among the elite group, designing one of them as the initial solution (S1) and the other as the guiding solution (S2). The path relinking then slightly starts to alter the genes of solution (S1) to transform it into the solution (S2). Each possible gene alteration in order to change (S1) into (S2) is considered an individual step of the process, and produces several intermediate solutions. These intermediate solutions found in each single step are compared between themselves to find the best intermediate solution (higher PSNR) among them, which is designated as (S1'). This solution (S1') is then compared to the best solution found until now (S*). If solution (S1') is better than solution (S*), then (S*) is updated and (S1') is incorporated into the elite group for future combining with other elite group solutions. The path relinking

process then continues, this time starting from solution (S1') as the initial solution to achieve (S2) and producing another best intermediate solution (S1'') for this second step. This process ends only when (S1) is turned into (S2) and after all solutions from elite group are selected. The 10 best individuals will form the initial population for the next generation of the genetic algorithm. See section 2.5 for explanation about path relinking and section 4 for details concerning the use of path relinking in image steganography.

3.4 Data hiding

After obtaining the solution (S*) which is the best solution found after the genetic algorithm and path relinking processes, the decomposed image (E') must be converted by using (S*) (STEP 6). This produces the decomposed image converted by substitution matrices (E''). This procedure is explained in section 2.3.

Finally, a replacement of the residual image (R) by the decomposed image converted by substitution matrices (E'') inside the cover image (C), producing the stego image (Z) (STEP 7). The process is done by replacing the least significant bits of the cover image (C) by those contained in (E'') for all pixels. This is considered in section 2.3, item 4.

4. The novel use of path relinking refinement in image steganography

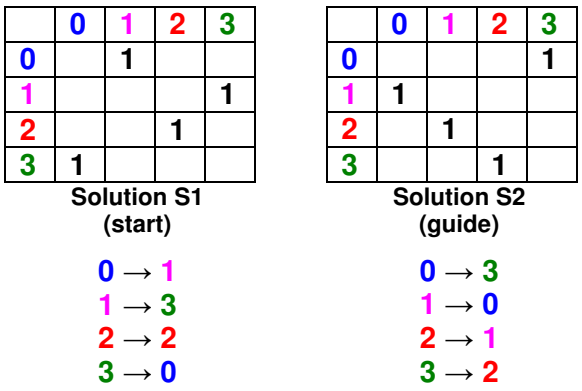
In most computational optimization applications, the path relinking can be applied to improve the results obtained, combining the best solutions already found by another meta-heuristic in a search for better ones.

The path relinking use described here is adapted to image steganography and was applied after the end of each generation of the genetic algorithm described in section 2.4 and in step 4 of section 3. It could be easily used together also with several other computational optimization methods, like GRASP [27], Tabu Search [28, 29], or any other, in order to improve the quality of the solutions obtained by them.

4.1 Path relinking use in grayscale image steganography

In grayscale steganography, each individual in genetic algorithm and path relinking processes is composed by a single substitution matrix, constructed to replace the grayscale intensities present in each pixel of the decomposed image (E') (see Figure 23).

After each execution of the genetic algorithm, it returns a selection of the 10 best individuals found. The path relinking groups these individuals randomly in pairs. Inside each pair of individuals, one of them is designated as a starting solution (S1) and the other as a guiding solution (S2). The representation row → column is used to indicate the color conversions present in each individual and facilitate the understanding of the path relinking process. To illustrate the process, let us suppose a simple grayscale steganography process, where the solutions S1 and S2 below, each one composed by a single 2x2 substitution matrix, have being chosen for path relinking:

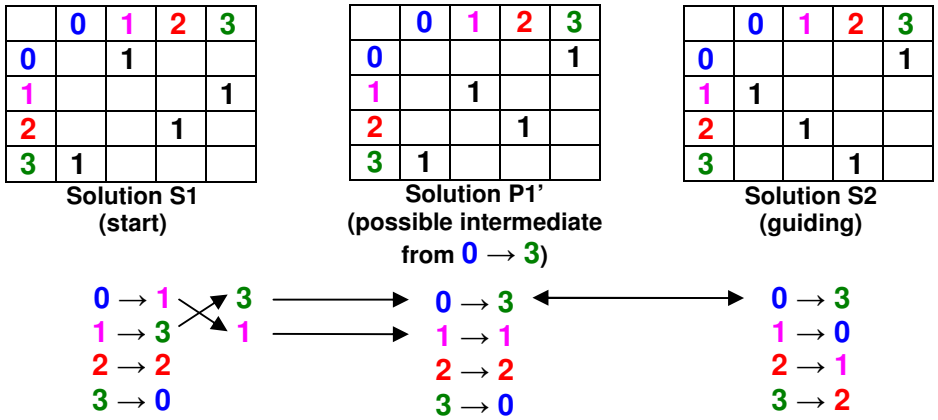


Now let us focus on how the start solution (S1) can be transformed into the guiding solution (S2). The first step is to look at the guiding solution (S2) and identify which parts in S2 are different from S1 solution. In this case, none of the conversions in S2 (**0** → **3**, **1** → **0**, **2** → **1** and **3** → **2**) exist in S1, so it is possible to start transforming S1 into S2 by changing any of these parts. In fact, all alternatives must be tried to see which one is the best. Starting with **0** → **3**, for example: in order to transform S1 so it incorporates the **0** → **3** conversion, it is necessary to find inside S1 the following conversions:

The conversion having **0** as original color: there is **0** → **1** in S1;

The conversion having **3** as resulting color: there is **1** → **3** in S1.

So, to achieve the conversion **0** → **3** in S1, the resulting colors present in the conversions **0** → **1** and **1** → **3** must be swapped:



Now, there is a possible intermediate solution P1'. This process must be repeated to every other possible first step to turn the solution S1 into solution S2. The other possible first steps

are: 1 → 0, 2 → 1 and 3 → 2. By trying the three other possible first steps, a total of four possible intermediate solutions P1', P2', P3' and P4' will be available:

	0	1	2	3
0				1
1		1		
2			1	
3	1			

Solution P1' (possible intermediate from 0 → 3)

0 → 3
1 → 1
2 → 2
3 → 0

	0	1	2	3
0		1		
1	1			
2			1	
3				1

Solution P2' (possible intermediate from 1 → 0)

0 → 1
1 → 0
2 → 2
3 → 3

	0	1	2	3
0			1	
1				1
2		1		
3	1			

Solution P3' (possible intermediate from 2 → 1)

0 → 2
1 → 3
2 → 1
3 → 0

	0	1	2	3
0		1		
1				1
2	1			
3			1	

Solution P4' (possible intermediate from 3 → 2)

0 → 1
1 → 3
2 → 0
3 → 2

Now, it is necessary to elect one of them to be the definitive intermediate solution S1'. This can be done by calculating the fitness value (PSNR) for each one of the four possible solutions and then choosing the one presenting the highest PSNR value as definitive intermediate solution S1', so we can continue the process. Supposing the solution showing the highest PSNR was P1', so S1' (definitive intermediate solution) = P1' (possible intermediate solution from 0 → 3).

The next step is to identify the possibilities available into turning the intermediate solution S1' into S2. For this next step, the options 1 → 0, 2 → 1 and 3 → 2 can be tried.

	0	1	2	3
0				1
1		1		
2			1	
3	1			

Solution S1' (intermediate)

0 → 3
1 → 1
2 → 2
3 → 0

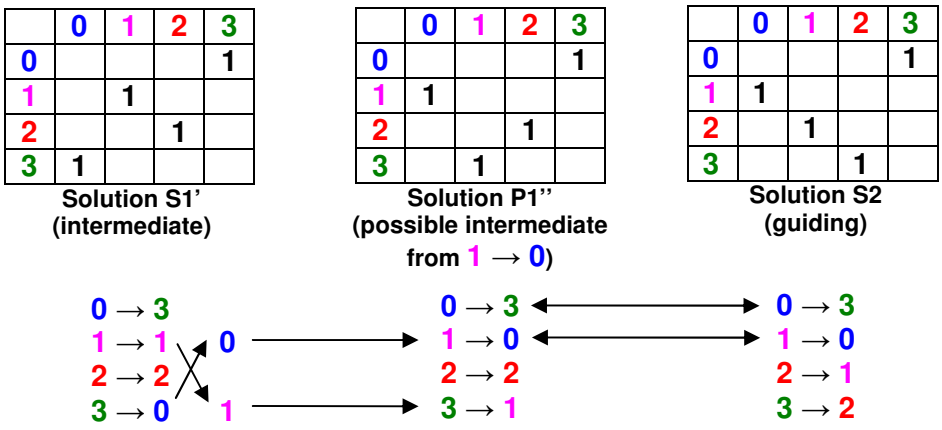
	0	1	2	3
0				1
1	1			
2		1		
3			1	

Solution S2 (guide)

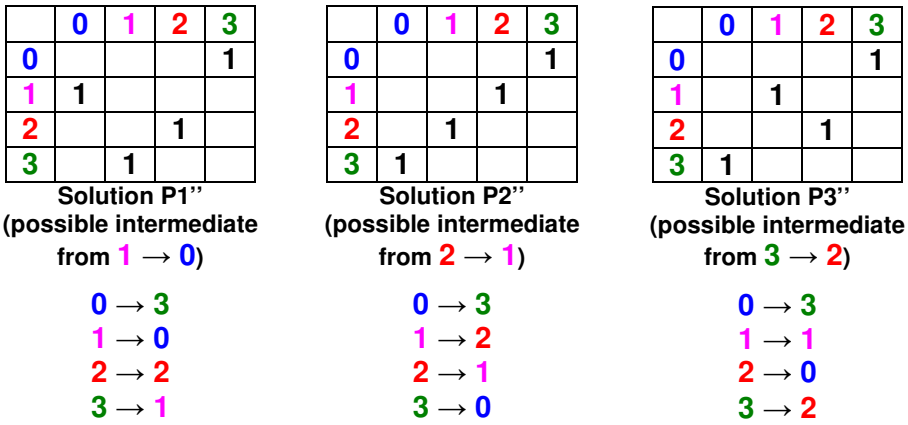
0 → 3
1 → 0
2 → 1
3 → 2

Analyzing the first try (1 → 0), in order to transform the intermediate solution S1' so it incorporates the (1 → 0) conversion, it is necessary to identify inside S1' the conversion having

1 as original color (there is 1 → 1) and the conversion having 0 as resulting color (there is 3 → 0). To achieve the desired result, the values of the resulting colors must then be swapped.

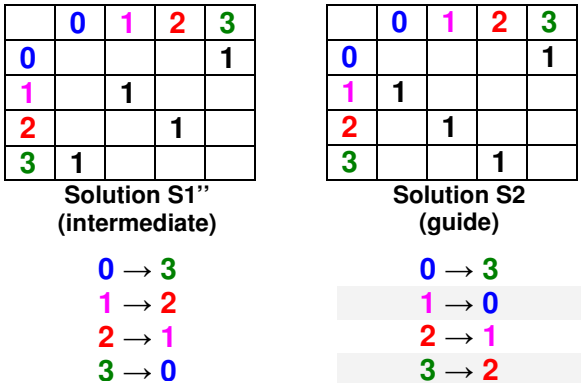


The other two possibilities must also be tested: 2 → 1 and 3 → 2. As result, three possible intermediate solutions arise: P1'', P2'' and P3''.

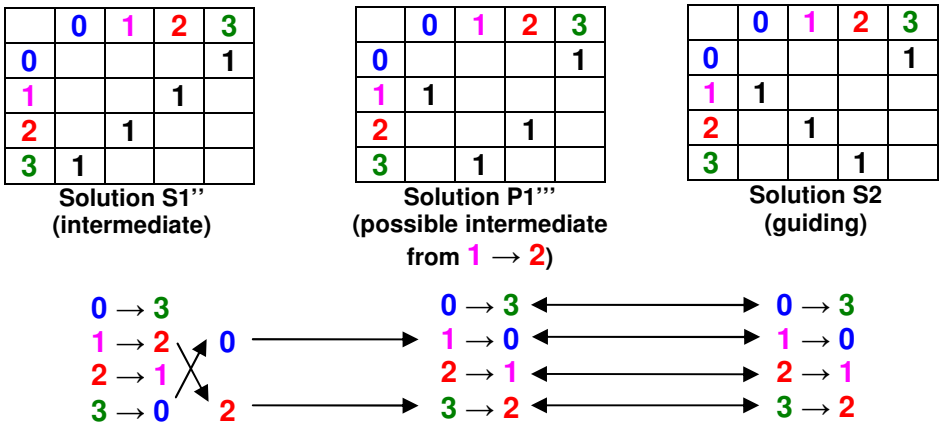


Now, it is necessary to elect one of them to be the definitive intermediate solution S1''. This can be done by calculating the fitness value (PSNR) for each one of the three possible solutions and then choosing the one presenting the highest PSNR value as definitive intermediate solution S1'', so we can continue the process. Supposing the solution showing the highest PSNR was P2'', so S1'' (definitive intermediate solution) = P2'' (possible intermediate solution from 2 → 1).

The next step is to identify the possibilities available into turning the intermediate solution S1'' into S2. For this next step, there are two remaining options: 1 → 0 and 3 → 2.



Analyzing the first try (1 → 0), in order to transform the intermediate solution S1'' so it incorporates the (1 → 0) conversion, it is necessary to identify inside S1'' the conversion having 1 as original color (we have 1 → 2) and the conversion with 0 as resulting color (we found 3 → 0). To achieve the desired result, the values of the resulting colors must then be swapped.



After the substitutions, the possible intermediate solution P1''' found is exactly the S2 solution, indicating the end of this path relinking. In this example, the path relinking method showed two new intermediate solutions: S1' and S1''. If one of these solutions present fitness value (PSNR) equal or higher than the best solution present in the best solutions list, this solution will be added to the best solutions list, so it may be paired and then combined with another solution afterwards by using the same method. The whole process ends when there will be no more solutions on the list of best solutions to be combined.

Interesting results also were found by applying the path relinking process in a reversal way (reverse path relinking). After finishing the normal path relinking, from start solution S1 to guiding solution S2, some experiments applying another path relinking starting from solution S2 to the guiding solution S1 were tried, thus obtaining more intermediate solutions, and several times a better final result than only when applying a normal path relinking alone.

4.2 Path relinking use in color image steganography

Differing from grayscale images, which only use 8-bits for representing gray intensities inside each pixel, the 32-bits color images pixels are composed of three color intensities of 8-bits each: red, green and blue.

In color image steganography, each new individual or specific solution generated by genetic algorithm or path relinking is composed by a total of three color substitution matrices, each one for its respective RGB color channel (red, green and blue substitution matrices). Each step of the path relinking process must then be executed three times, once for each 8-bits color channel (red, green and blue), corresponding to the three color substitution matrices that compose each solution. After these three executions, a step is concluded, producing a new intermediate solution to be evaluated for aptitude, just like in grayscale path relinking. Aptitude of each individual is measured by average PSNR values obtained by comparing each cover image (H) color channel (red, green and blue) with the channels present in stego image (Z).

Regarding to time cost, using the path relinking in color images is three times slower than grayscale path relinking, in average.

4.2.1 The RGB cube: an alternative structure for constructing solutions

As an alternative approach for color image steganography, we also present a different way to construct the solutions used in genetic algorithms and path relinking. Instead constructing the solution based on three substitution matrices (reg, green and blue), it is possible to construct only one tri-dimensional substitution matrix. Each dimension in this substitution matrix represents one of the RGB color channels. The Figure 24 helps to illustrate this concept.

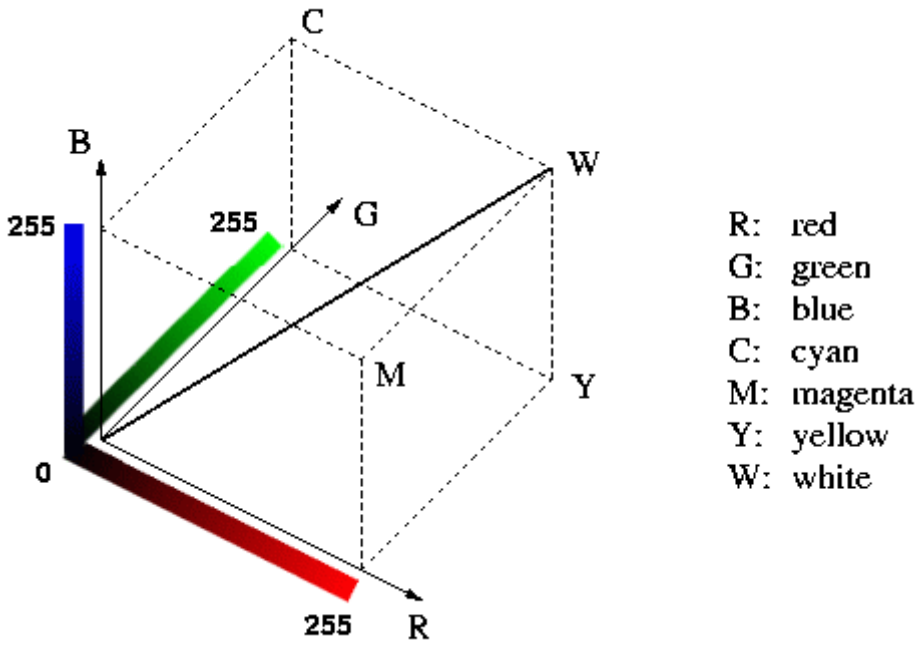


Figure 24 – The RGB cube, representing the color channels of the tri-dimensional substitution matrix

When using the tri-dimensional substitution matrix, each element of this matrix is an initial RGB color represented by matrix [R, G, B] where R, G and B are the red, green and blue intensities of this color. Inside each element of the matrix are the values representing the converted color, a structure called RGBQuad. The RGBQuad is an array of four values: the red intensity, the green intensity, the blue intensity and the opacity intensity (alpha channel). The opacity intensity is not used in this case, since we are working only on color conversion.

Let us suppose there is a pure red color to be converted into a pure green color. By using the tri-dimensional matrix, it would have the following element inside: [255][0][0] = [0,255, 0].

The use of tri-dimensional matrices has following advantages and drawbacks when compared to the traditional method:

- With this approach, the number of possibilities and each solution complexity increases significantly, slowing the performance of the genetic algorithms and path relinking processes, but improving the quality results;
- The space required to store the tri-dimensional substitution matrix inside the cover image is greater than traditional method, leaving a little less space to save secret data.

5. Implementation

5.1 The CxImage Class

CxImage is a C++ class that can load, save, display and transform images in a very simple and fast way. It is open source and licensed under zlib. With more than 200 functions, and with comprehensive working demos, CxImage offers all the tools to build simple image processing applications on a fast learning way. Supported file formats are: BMP, GIF, ICO, CUR, JBG, JPG, JPC, JP2, PCX, PGX, PNG, PNM, RAS, SKA, TGA, TIF, WBMP, WMF, RAW, CRW, NEF, CR2, DNG, ORF, ARW, ERF, 3FR, DCR, X3F, MEF, RAF, MRW, PEF, SR2. It is highly portable and has been tested with Visual C++ 6 / 2008, C++ Builder 3 / 6, MinGW on Windows, and with gcc 3.3.2 on Linux. The library can be linked statically, or through a DLL or an activex component. It is currently available for download in http://www.xdp.it/cximage.htm.

The main reason for choosing CxImage as a base for development was the availability of several functions that could easily convert all the pixel colors contained inside the image into a matrix of bytes, facilitating further processing and conversion of that information to complete the other steps of image steganography, including the genetic algorithm and path relinking. It also presents an easy to use interface and good compatibility among several image formats. It also allows the update and visualization of the image after the steganography and saving of stego image (Z) into a PNG or TIFF file. The other image formats were not used for saving the results after image steganography because they work with compression of image data, which risks loosing the information embedded inside the image.

The CxImage is currently in version 6.00. The version adopted for image steganography was 5.93c, the last available when this project started.

5.2 The CxImageHider software

Several additions were included in CxImage project to achieve image steganography. Therefore, after all the mods, it was denominated CxImageHider. Figure 25 shows a picture of CxImageHider software under execution, with an open image of Lena.

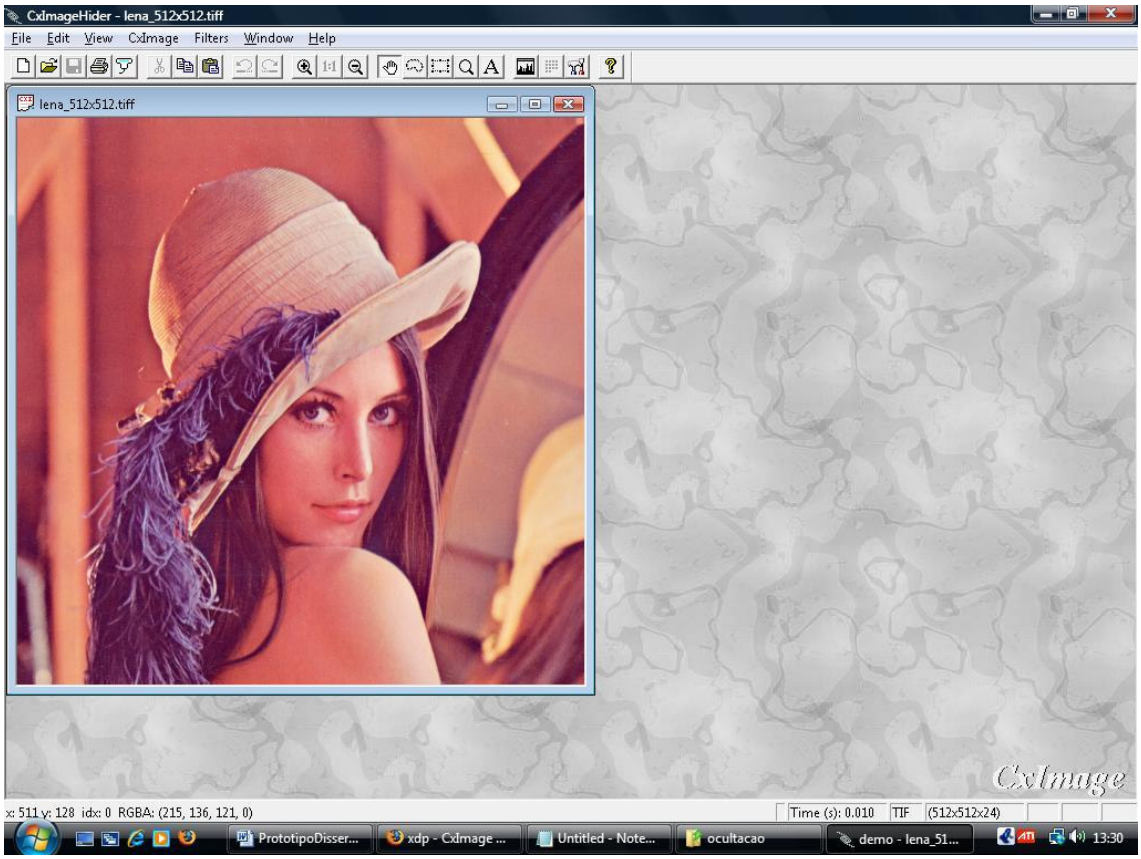


Figure 25 – CxImageHider software with an open image of Lena

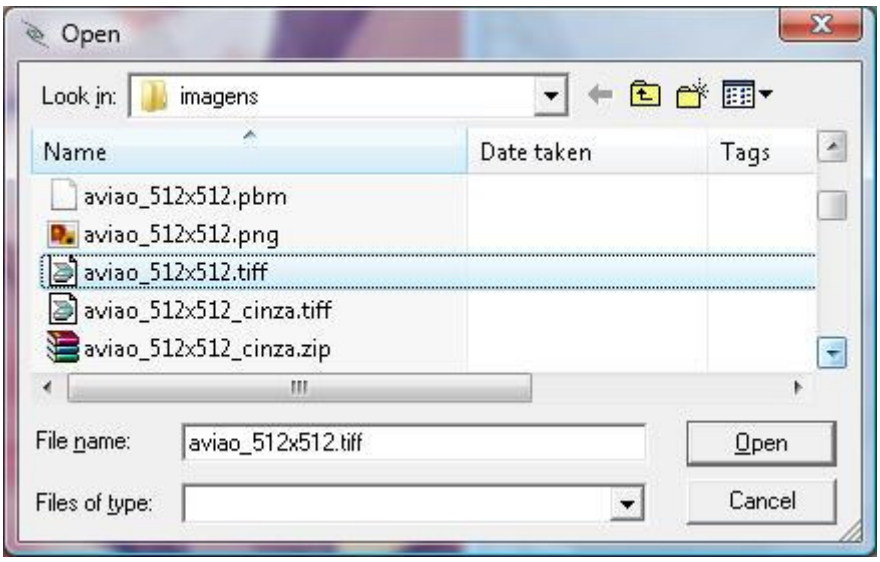


Figure 26 – Dialog window asking for a file to be embedded inside the cover image

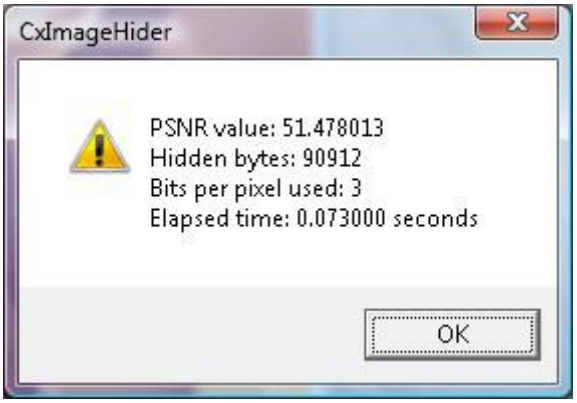


Figure 27 – Dialog showing PSNR value, bits per pixel used, total of bytes hidden and time elapsed in seconds after the image steganography

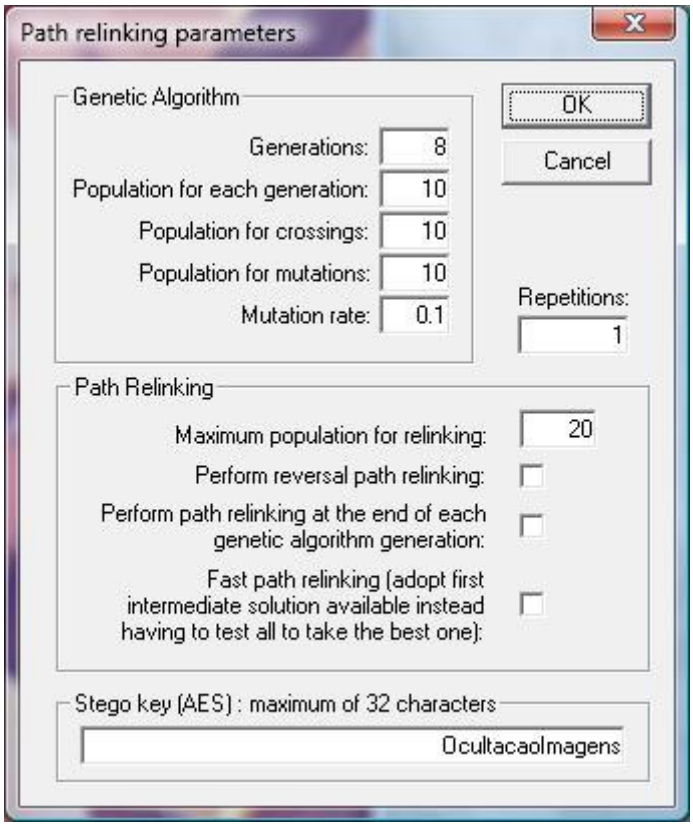


Figure 28 – Dialog window asking for genetic algorithm and path relinking parameters, and a stego key

Inside CxImage menu, five new options were added. They only become available after the user opens an image within the software, which is considered to be the cover image. The options are:

- Steganography – LSB Substitution: When this option is selected, it opens a dialog window as the one shown on the Figure 26, asking for a file to be embedded inside the cover image. After the file is selected and the Open button is pressed,

the CxImageHider applies the simple LSB substitution method to hide the file inside the image. It also shows the PSNR value, bits per pixel used, total of bytes hidden and time elapsed in seconds after concluding the process (see Figure 27). Any kind of file can be chosen to be embedded into the cover image, including text, image, compressed file or any other. The only limitation is the space available inside the cover image, which limits the size of file being chosen.

- Steganography – Genetic Algorithm: Similar to the option "Steganography – LSB Substitution", after being selected, the CxImageHider software runs a genetic algorithm in order to search for a good substitution matrix to improve the stego image quality. The parameters for this genetic algorithm are the same suggested by [16]: 8 generations, 10 random individuals as initial population, at each generation a copy of the 10 individuals, plus 10 individuals resulting from crossing and 10 resulting from mutation at 0.1 rate. At the end of each generation, the selection takes only the best 10 individuals among these all to continue to the next generation. This process is detailed in section 2.4 and section 3 under step 4. It also shows the PSNR value of the best solution found by the genetic algorithm, bits per pixel used, total of bytes hidden and the time elapsed in seconds (see Figure 27), updating the cover image with the embedded information.
- Steganography – Path Relinking: Upon selection, a dialog window opens (see Figure 28) asking for the several parameters. The genetic algorithm parameters are: number of generations, population selected for each generation, population generated by crossover, population generated by mutation and mutation rate. The path relinking parameters are: maximum population generated by path relinking, use of reversal path relinking, use of path relinking at the end of each genetic algorithm generation, use of rapid reconnections (takes the first option that excel the best solution instead of testing all options to see whichever is the best one). The other two parameters are the stego key, whose length is 128, 192 or 256 bits

and the number of executions or repetitions, so the software runs several times and calculate and returns an average value from the final results of each execution afterwards. After supplying these parameters and pressing the OK button, it opens a second dialog window, as the one shown on the Figure 26, asking for the file to be embedded inside the cover image. After the file is selected and the Open button is pressed, the CxImageHider software applies the steganography over the cover image, encrypting the file with the given stego key, running the genetic algorithm and path relinking procedures and finally embedding the encrypted file inside the cover image. It also shows the PSNR value of the best solution found by the path relinking, bits per pixel used, total of bytes hidden, the time elapsed in seconds and the size of the largest reconnection elite group (see Figures 29 and 30), updating the cover image with the embedded information.

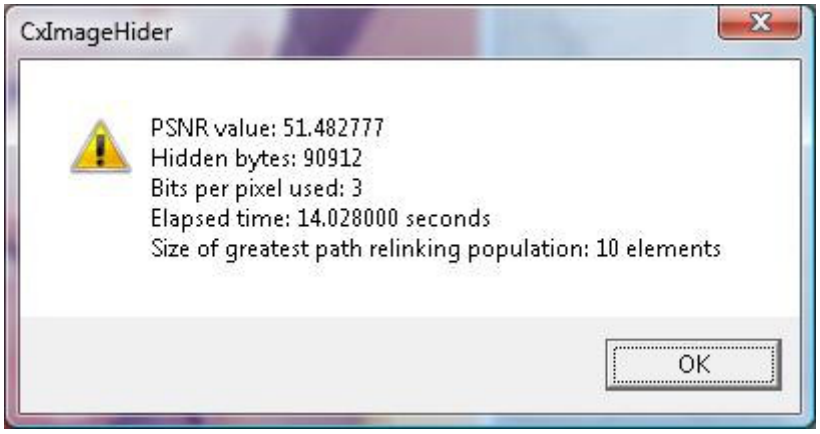


Figure 29 – Dialog window showing results after image steganography with path relinking at each genetic algorithm generation

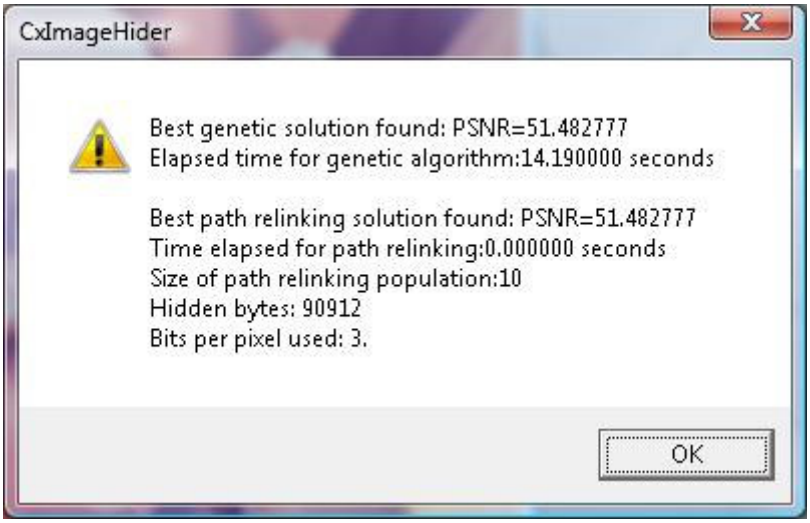


Figure 30 - Dialog window showing results after image steganography with path relinking after the end of genetic algorithm

- Undo Steganography: After selected, that option brings a dialog window as the one shown on Figure 31, asking for a stego key to undo the steganography process on the image and a path to restore the embedded file. After pressing the OK button, the software will start to undo the steganography by reconstructing the embedded information and decrypting it with the use of a given stego key afterwards. If everything is ok, a dialog window like the one presented in Figure 32 is shown, and the embedded file will be restored and saved in the specified path.

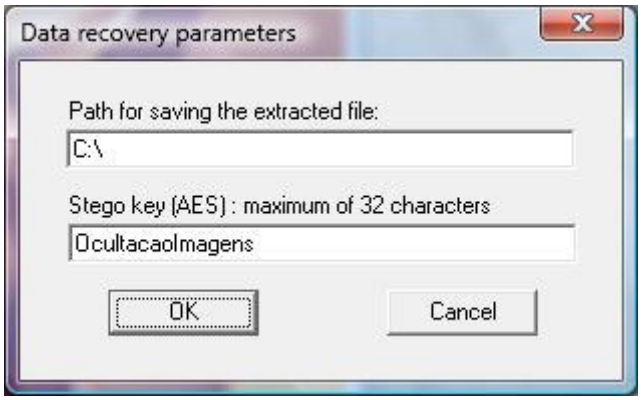


Figure 31 – Dialog window asking for parameters to undo image steganography

Figure 32 – Dialog window showing undo steganography success

- MSE between images: This option is used to compare 2 similar images and see its diferences in terms of mean square error (MSE). After selected, it opens a dialog window as the one shown on the Figure 26, asking for a second image file to compare to the image already open within the software. After an image file is selected and the Open button is pressed, the CxImageHider calculates and shows the MSE value between the two images, as shown on Figure 33.

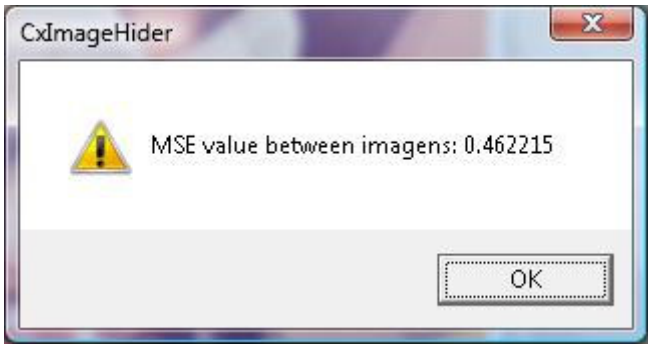


Figure 33 – Dialog window presenting MSE value between the two compared images

To implement those five new options, two classes were created: Hiding and CryptographyAES.

The Hiding class incorporates all methods necessary for image steganography. The most important ones are:

- Hiding: The construction method, responsible for creating a matrix of bytes with the same size of the cover image (C) and copying all the pixel values to this matrix. It also reads the information to be hidden converting it into an information array (I) of bytes and calls the Encrypt method of the CryptographyAES class to encrypt the information inside this array.

- MSE: The method for calculating the mean square error between two given images.
- LSBSubstitution: This method implements the least significant bits substitution, embedding the information contained inside the encrypted information array directly into the cover image (C).
- GeneticAlgorithm: This method implements the genetic algorithm to search for a good substitution matrix to convert the encrypted information array (I) before it is embedded inside the cover image (C). Afterwards, it calls the updateCoverImage procedure to embed the converted information array into the cover image (C).
- PathRelinking: A method implementing the path relinking procedure by combining the solutions already found after the end of the genetic algorithm execution, in order to search for better solutions. Afterwards, it calls the updateCoverImage procedure to embed the converted information array into the cover image (C).
- GeneticRelinking: A method implementing the path relinking procedure by combining the solutions already found at each generation of the genetic algorithm execution, in order to search for better solutions. At the end of execution, it calls the updateCoverImage procedure to convert the encrypted information array into the cover image (C).
- UndoSteganography: This method is responsible for undoing the image steganography. It reads the encrypted information from the cover image (C), calls the Decrypt method of CryptographyAES class to decrypt the information and saves the information on the path indicated, recovering the original hidden file.
- PSNR: This function compares two images and returns the peak-to-signal noise ratio value between them, acting as measure of quality and aptitude for the genetic algorithm and the path relinking selections of best individuals.

The CryptographyAES class is an implementation of the Advanced Encryption Standard (AES) algorithm. Its most important methods are:

- Encrypt: This method process the information array (I) and encrypts all the information by using the given cipher key of 128, 192 or 256 bits, generating the encrypted information array.
- Decrypt: This method process the encrypted information array and decrypts all the information by using the given cipher key of 128, 192 or 256 bits, reproducing the information array (I).

6. Experimental results

Experimental results of the implemented schemes are presented in this section. The tests with grayscale cover images ran on a first personal computer, equipped with an 850 MHz clock Pentium III processor and 512 megabytes of RAM memory (DIMM) under the Microsoft Windows 2000 operational system. The tests with color cover images ran on a second personal computer, equipped with an AMD Athlon X2 5200+ (2 x 2.6 GHz) processor and 4 gigabytes of RAM memory (DDR2-800Mhz) under the Microsoft Windows Vista 64 Business operational system. The CxImageHider program was written in C++ language as described in chapter 5. The images used for the tests are Figures 34-60. Figures 34-36, 43-45 and 50-59 are composed of 512 x 512 pixels. Six of them (Figures 34-36 and 43-45) are 8-bit grayscale and the other ones are 32-bits color. The Figures 37-42 and 46-50 are 8-bits grayscale, containing 256 x 512 pixels. The Figures 34-39, 41-51 and 53-59 were obtained in [44-45]. The Figure 40 was sent to us by authors of [16] and the Figure 52 was part of a print screen containing a black text and white background edited inside a text editor. Figure 60, the Pacman3D, is a 32-bits color image of 640x480 pixels, obtained from [46]. All figures used are in TIFF file format.

In order to recover the data hidden by steganography process afterwards, the stego image (Z) must be saved into PNG or TIFF file formats. Other file formats, like JPEG and PCX, are not suitable for saving the results because they work with compression of image data, which risks loosing the information embedded inside the image.

Nine works were selected for comparison of results [13-14, 16, 17, 20, 37-40], each one showing a different strategy for image hiding. The comparisons are most similar as possible, utilizing images of the same size and aspect than the ones presented by those papers. The results presented by our method are average values of five executions for each compared instance of secret data and cover image.

6.1 Previous works

In [16], the main idea is to apply a genetic algorithm to speed up the search for a near optimal solution, instead using simple LSB substitution. It has many benefits, allowing the work with larger numbers of least significant bits ($k \geq 3$), otherwise it would take too long to find an optimal solution among all possibilities.

In [17], a DES-like cryptosystem is used in conjunction with modulus operations to incorporate secret data into the cover image, improving the simple LSB substitution method. It analyzes the hidden data security and is presented as an alternative image hiding strategy.

In [13], a modulus operation concept is used to accelerate the search for an optimal solution to LSB substitution. It shows mathematical analysis of the pixel visual distortion range and is presented as an optimization in quality and speed over [16], replacing the genetic algorithm.

In [14], a dynamic programming strategy was used to optimize the calculations of the MSE values from all possible solutions and then find an optimal solution for LSB substitution. It presents a speed and quality optimization over [16] and replaces the genetic algorithm as well.

In [20], a color quantization process is adopted in order to generate a codebook and use the Vector Quantization to compress the image, reducing the number of colors inside the image to be embedded to 256. It also uses the DES (Data Encryption Standard) cryptography algorithm to encrypt the image data.

In [37], the proposal is a lossless data embedding scheme that exploits the difference expansion of the pixels to conceal large amount of message data in a digital image. The proposed scheme takes into consideration the correlation between the pixel and its surrounding pixels to determine the degree of the difference expansion for message data embedding.

In [38], a reversible steganography method is presented, which can reconstruct an original image effectively after extracting the embedded secret data. The proposed reversible hiding method aims at BTC (block truncation coding)-compressed color images, allowing steganography on a great number of three and four bits secret patterns inside the cover images.

It also uses a genetic algorithm to find the best blocks to be used in the block truncation coding process.

In [39], a data hiding technique for color images using a BSP (Binary Space Partitioning) tree is proposed. The RGB values at each pixel are treated as a three-dimensional (3D) virtual point in the XYZ coordinates and a bounding volume is employed to enclose them. A BSP tree is constructed by recursively decomposing this bounding volume into voxels containing one or more 3D virtual points. These voxels are categorized into eight subspaces, each represented as three-digit binary characters, furtherly used to embed the information accordingly, helping to reduce the degradation level of the stego image.

In [40], a palette modification scheme is presented, which can iteratively embed one message bit into each pixel in a palette-based image. In each iteration, both the cost of removing an entry color in a palette and the benefit of generating a new one to replace it are calculated. If the maximal benefit exceeds the minimal cost, the entry color is replaced, improving the quality of stego images when compared to the LSB substitution method. Unfortunately comparisons with this work were not possible because the images used inside [40] could not be found on internet and we contacted the authors by e-mail, asking for the pictures “fruit” and “swimmer”, but did not receive any answers.

We also found some other papers with superior quality results, like [15]. However, this kind of approach only uses grayscale images, and never could hide critical data, like a secret message or an executable file, since some information is always lost during the hiding process. Its results were obtained by applying lossy compression of secret data and by creation of a dictionary of common terms, which is a strategy that allows the hiding of much more data, even more than one entire image of the same size as the cover image. Our work is much more flexible, allowing a lossless steganography of any digital media. So, a comparison of results would be pointless in that case.

Figure 34 – grayscale Lena 512x512 pixels

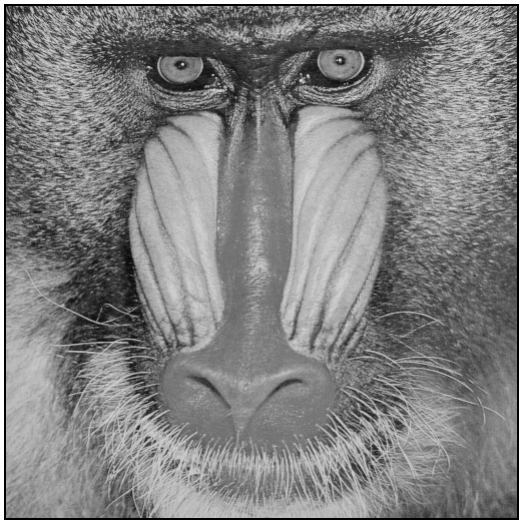
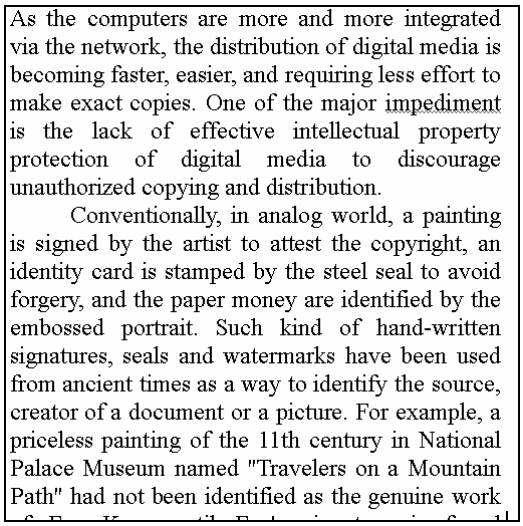


Figure 35 – grayscale Baboon 512x512 pixels


As the computers are more and more integrated via the network, the distribution of digital media is becoming faster, easier, and requiring less effort to make exact copies. One of the major impediment is the lack of effective intellectual property protection of digital media to discourage unauthorized copying and distribution.

Conventionally, in analog world, a painting is signed by the artist to attest the copyright, an identity card is stamped by the steel seal to avoid forgery, and the paper money are identified by the embossed portrait. Such kind of hand-written signatures, seals and watermarks have been used from ancient times as a way to identify the source, creator of a document or a picture. For example, a priceless painting of the 11th century in National Palace Museum named "Travelers on a Mountain Path" had not been identified as the genuine work

Figure 36 – grayscale Text 512x512 pixels

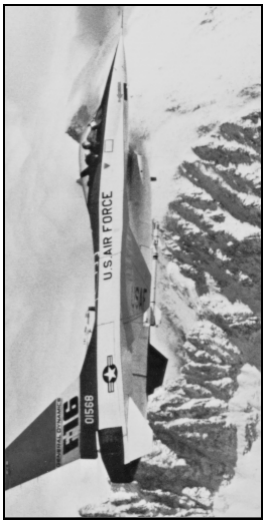


Figure 37 – grayscale Jet 256x512 pixels

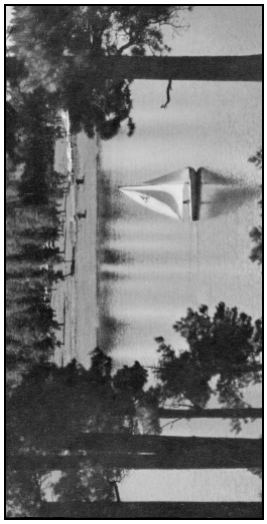


Figure 38 – grayscale Sailboat 256x512 pixels

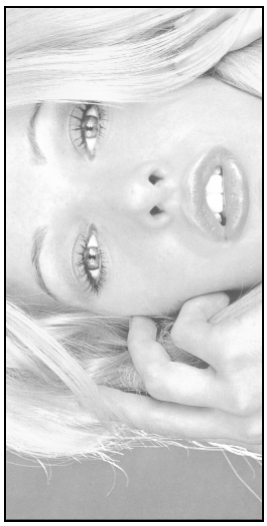


Figure 39 – grayscale Tiffany 256x512 pixels

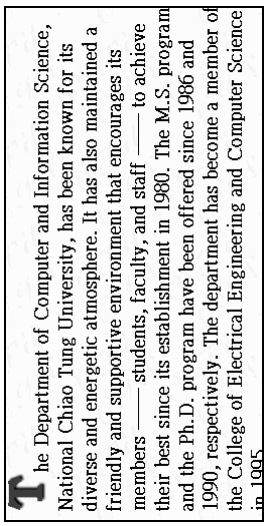

The Department of Computer and Information Science, National Chiao Tung University, has been known for its diverse and energetic atmosphere. It has also maintained a friendly and supportive environment that encourages its members —— students, faculty, and staff —— to achieve their best since its establishment in 1980. The M.S. program and the Ph.D. program have been offered since 1986 and 1990, respectively. The department has become a member of the College of Electrical Engineering and Computer Science in 1995

Figure 40 – grayscale Text 256x512 pixels

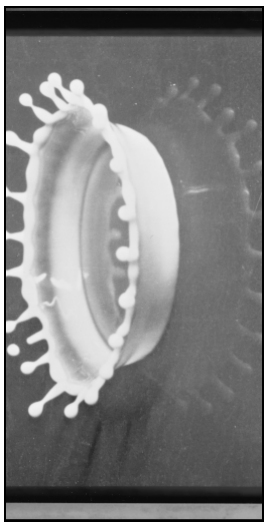


Figure 41 – grayscale Splash 256x512 pixels

Figure 42 – grayscale Fishing Boat 256x512 pixels

Figure 43 – grayscale Peepers 512x512 pixels

Figure 44 – grayscale Barbara 512x512 pixels

Figure 45 – grayscale Jet 512x512 pixels

Figure 46 - grayscale Jet 256x512 pixels (squeezed)

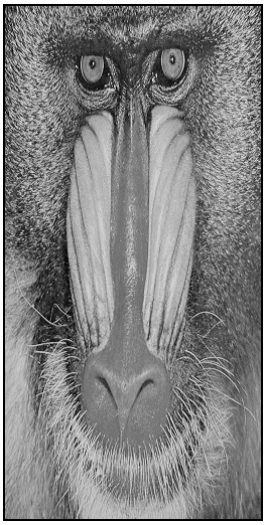


Figure 47 - grayscale Baboon 256x512 pixels (squeezed)

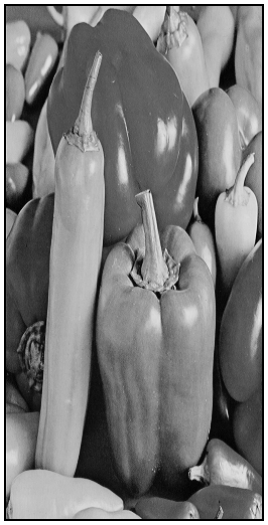
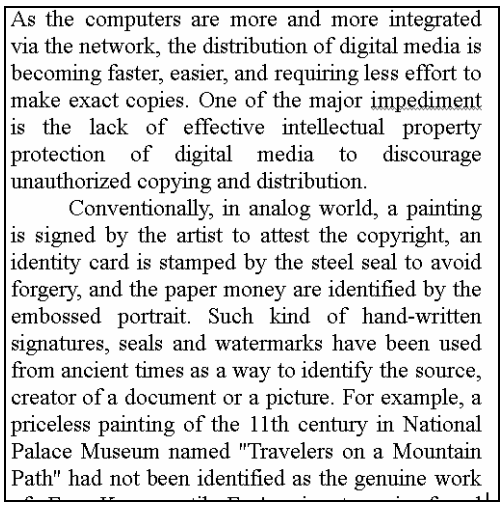


Figure 48 - grayscale Peepers 256x512 pixels (squeezed)

Figure 49 - grayscale Lena 256x512 pixels (squeezed)

Figure 50 – color Lena 512x512 pixels

Figure 51 – color Baboon 512x512 pixels



As the computers are more and more integrated via the network, the distribution of digital media is becoming faster, easier, and requiring less effort to make exact copies. One of the major impediment is the lack of effective intellectual property protection of digital media to discourage unauthorized copying and distribution.

Conventionally, in analog world, a painting is signed by the artist to attest the copyright, an identity card is stamped by the steel seal to avoid forgery, and the paper money are identified by the embossed portrait. Such kind of hand-written signatures, seals and watermarks have been used from ancient times as a way to identify the source, creator of a document or a picture. For example, a priceless painting of the 11th century in National Palace Museum named "Travelers on a Mountain Path" had not been identified as the genuine work

Figure 52 – color Text 512x512 pixels

Figure 53 – color Peepers 512x512 pixels

Figure 54 – color Barbara 512x512 pixels

Figure 55 – color Jet 512x512 pixels

Figure 56 – color House 512x512 pixels

Figure 57 – color Sailboat 512x512 pixels

Figure 58 – color Zelda 512x512 pixels

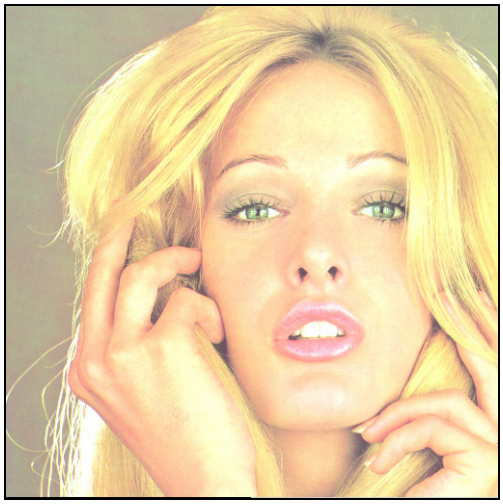


Figure 59 – color Tiffany 512x512 pixels

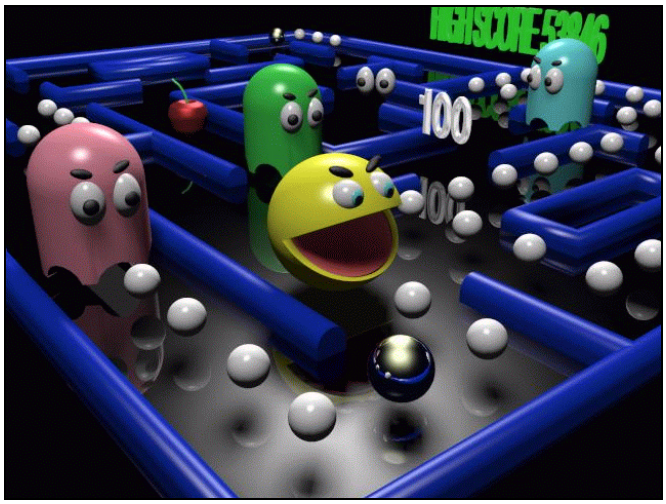


Figure 60 – color Pacman3D 640x480 pixels

6.2 The BI (billions of instructions) performance measurement concept

Considering facilitates future time comparisons between our results and others, we also present a **performance measurement concept**, called BI, or "Billions of Instructions". The main idea consists into measuring the CPU speed in terms of instructions executed per second, estimating how many BI (billions of instructions) will be required to execute an image hiding task and then using this value to compare the performance among diverse computer systems. To achieve this, a very popular benchmarking program called Dhrystone [41] was used.

The Dhrystone is a mixed set of instructions containing several operations involving integer numbers, which the main purpose is to measure the CPU performance. This program runs on a machine for some time (ex. several seconds) and the total number of instructions executed by the CPU during this time is counted. This total is divided by the time elapsed and is possible to calculate how many millions of instructions the CPU was capable to execute within a second, which is called MIPS (millions of instructions per second). This allows us to measure the CPU performance in terms of millions of instructions executed per second. The target computer system can be compared with other computer systems by using MIPS results, so it is possible to conclude about the capability of executing more instructions per second when running the Dhrystone program.

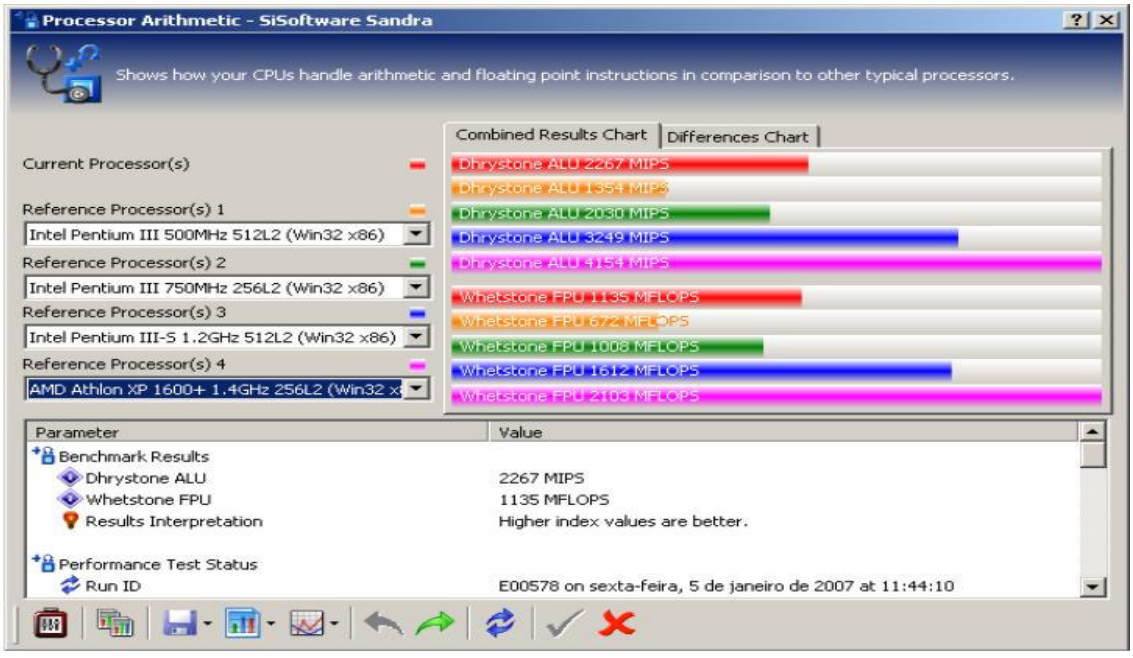


Figure 61 – Dhrystones Benchmark of first computer system using SiSoftware Sandra Lite XI b

To calculate how many billions of instructions will be required for executing an image hiding task, the following steps must be executed:

Step 1: Obtain the Dhrystones benchmark result for the testing computer system: To calculate this, very popular benchmarking software was used, the System Analyser, Diagnostic and Reporting Assistant (SANDRA) [42] Version Lite XI b. After running the Processor Arithmetic Test on our first computer system (Pentium III, 850 Mhz) (see Figure 61), it reported a value of 2267 MIPS (millions of instructions per second) on the Dhrystones benchmark;

Step 2: Obtain the result of the image hiding task in seconds: To calculate this, simply run your image hiding process and measure the total time elapsed in seconds. For this example, the task of embedding the Jet image (Figure 37) into the Lena cover image (Figure 34) will be used, applying only the genetic algorithm. In that case the value obtained for this task was 33.308 seconds;

Step 3: Calculate the total BI (billions of instructions): by multiplying the task result time (in seconds) for the Dhrystone benchmark value in MIPS (millions of instructions per second) and dividing by 1000. The final value is 33.308 seconds * 2267 MIPS / 1000, resulting in 75.51 BI (billions of instructions) executed for this task.

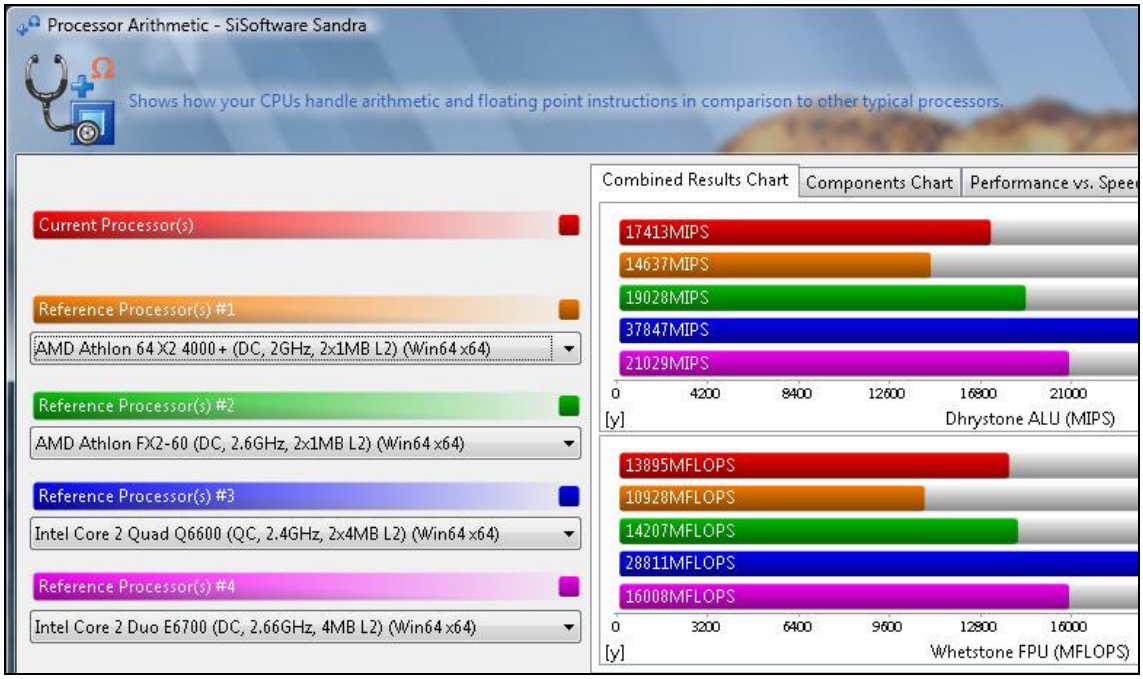


Figure 62 – Dhrystones Benchmark of second computer system using SiSoftware Sandra Lite XII SP1

The Dhrystones benchmark result was also obtained for the second testing computer system (AMD 5200+ processor): a score of 17413 MIPS in SANDRA Lite Version XII SP1 (See figure 62).

The higher the BI value obtained for a task, harder will be that specific image hiding task, since it always will use more CPU time to execute a bigger number of instructions. So the faster tasks are those presenting the lower BI values.

6.3 Comparison with simple LSB substitution and the genetic algorithm results

In this section, our results are compared with those presented by [16]. The tests used the same cover images and secret data than [16]. The cover images are the 8-bits grayscale 512x512 pixels images of Lena, Baboon and Text (Figures 34-36) and the 32-bits color versions of those images (Figures 50-52). The secret data are the 8-bits grayscale 256x512 pixels images of Jet, Sailboat, Tiffany and Text (Figures 37-40). The obtained and reported values for LSB [16] and GEN [16] are the results presented with simple LSB substitution and the genetic algorithm approaches, with exception of some strange results reported for cover image Text (Figure 36). The authors of [16] were contacted and sent us updated correct data for these results.

LSB2 and GEN2 are the results achieved by our implementation of least significant bits substitution and genetic algorithm proposed by [16]. GA+PR is our new proposal, an implementation of the genetic algorithm with the path relinking approach, applied after the end of each genetic algorithm generation, by combining the 10 better solutions found until then (elite group). The program chooses random pairs of solutions among these ones to be combined, taking one as the initial solution (S1) and other as the guiding solution (S2). After the normal path relinking process, a reverse path relinking also is applied, considering solution S1 as the guiding one and S2 as initial. During this process, all intermediate solutions found in the path relinking process are immediately evaluated by the fitness function (PSNR) and the ones presenting equal or better values than the best solution are added to the elite group so they can be combined with the others not yet chosen solutions from the elite group too, until a maximum of 10 new intermediate solutions for each genetic algorithm generation. Each solution is

selected for combining only once. GA+PR' is a modified version of GA+PR, using a simpler and faster path relinking approach, by changing always the first different part between solutions S1 and S2, instead to testing all the possible changes to see which one is the best. GA+PR' is also less selective, since it accepts any intermediate solution found that is better than the worse solution present inside the population for the next generation (10 best individuals) until a maximum of 30 new individuals, instead of accepting only intermediate solutions equal or better the best solution among the next generation's population, as GA+PR does.

Quality comparisons are presented on table 1. The arrow symbol (→) was used to indicate that the image before the symbol is being embedded inside the image after the symbol. The word "zipped", appearing before the arrow symbol, indicates the embedding image was submitted to WinZip [43] compression application version 11.1, generating a lossless compressed file to be hidden inside the cover image. The word "3x", appearing before the arrow symbol, indicates three identical embedded images, instead of only one, were compressed by the WinZip application version 11.1, which also produces a lossless compressed file, of greater size. The words "gray" and "color", appearing after the arrow symbol, represent the use of 8-bits grayscale or 32-bits color versions of the cover image. The time results of our executions are shown on table 2, with cryptography time included for all tests with color cover images. The AES cryptography was not applied on tests with grayscale cover images. Time comparisons were not possible given the differences between the computer systems used in [16] and here. Table 3 shows the time results converted to the BI unit values (billions of instructions), for future comparisons. Each result of our implementations is an average value for five executions with the same cover image and secret data.

The path relinking results (GA+PR and GA+PR') presented when dealing with "zipped 3x" secret data being hidden inside color cover images were obtained by using the method described in section 4.2 while all others color image steganography results used the section 4.2.1 alternative approach, which leads to better results when secret data is around one third or less of cover image hiding capacity.

Secret data → cover image	Bytes hidden	Method (average PSNR result)					
		LSB [16]	GEN [16]	LSB2	GEN2	GA+PR	GA+PR'
Jet → gray Lena	99888	32.04	32.71	32.08	32.69	**32.79**	32.71
Jet → color Lena	99888	-	-	48.64	48.95	**49.03**	48.99
zipped Jet → color Lena	97280	-	-	51.18	51.20	**51.21**	**51.21**
zipped 3x Jet → color Lena	291728	-	-	37.94	37.97	**37.99**	37.97
Sailboat → gray Lena	120064	32.10	32.55	31.93	32.47	**32.58**	32.57
Sailboat → color Lena	120064	-	-	47.62	48.15	**48.22**	48.20
zipped Sailboat → color Lena	116128	-	-	47.99	48.02	**48.03**	48.02
zipped 3x Sailboat → color Lena	348256	-	-	33.18	33.20	**33.21**	33.20
Tiffany → gray Lena	92336	31.21	32.90	31.96	32.71	**32.90**	32.87
Tiffany → color Lena	92336	-	-	51.40	51.43	**51.44**	**51.44**
zipped Tiffany → color Lena	90912	-	-	51.49	**51.50**	**51.50**	**51.50**
zipped 3x Tiffany → color Lena	272608	-	-	38.25	38.26	**38.27**	38.26
Text → gray Lena	32560	29.51	34.27	29.21	**34.61**	**34.61**	34.36
Text → color Lena	32560	-	-	55.94	55.94	55.94	55.94
zipped Text → color Lena	32528	-	-	55.96	55.96	55.96	55.96
zipped 3x Text → color Lena	97536	-	-	51.18	**51.19**	**51.19**	**51.19**
Jet → gray Baboon	99888	32.11	32.79	32.16	32.63	**32.89**	32.83
Jet → color Baboon	99888	-	-	48.44	48.99	**49.02**	49.00
zipped Jet → color Baboon	97280	-	-	51.18	**51.20**	**51.20**	**51.20**
Zipped 3x Jet → color Baboon	291728	-	-	37.95	37.98	**37.99**	37.98
Sailboat → gray Baboon	120064	32.13	32.50	32.01	32.51	**32.68**	32.60
Sailboat → color Baboon	120064	-	-	47.94	**48.21**	**48.21**	48.20
zipped Sailboat → color Baboon	116128	-	-	47.98	48.02	**48.03**	48.02
zipped 3x Sailboat → color Baboon	348256	-	-	33.23	33.25	**33.26**	33.25
Tiffany → gray Baboon	92336	31.31	32.95	32.06	32.82	**33.02**	33.01
Tiffany → color Baboon	92336	-	-	51.42	51.42	51.42	51.42
zipped Tiffany → color Baboon	90912	-	-	51.48	**51.49**	**51.49**	**51.49**
zipped 3x Tiffany → color Baboon	272608	-	-	38.25	**38.27**	**38.27**	38.26
Text → gray Baboon	32560	29.60	34.38	29.25	34.83	**34.83**	34.76
Text → color Baboon	32560	-	-	55.95	55.95	55.95	55.95
zipped Text → color Baboon	32528	-	-	55.94	**55.95**	**55.95**	**55.95**
zipped 3x Text → color Baboon	97536	-	-	51.18	**51.19**	**51.19**	**51.19**
Jet → gray Text	99888	30.51	30.87	29.64	30.55	**30.77**	30.66
Jet → color Text	99888	-	-	46.85	48.25	**48.39**	48.33
zipped Jet → color Text	97280	-	-	51.24	51.24	51.24	51.24
zipped 3x Jet → color Text	291728	-	-	35.74	35.76	**35.77**	35.76
Sailboat → gray Text	120064	29.07	30.35	29.25	30.08	**30.49**	30.42
Sailboat → color Text	120064	-	-	46.40	47.61	**47.82**	47.76
zipped Sailboat → color Text	116128	-	-	47.26	47.27	**47.28**	**47.28**
zipped 3x Sailboat → color Text	348256	-	-	31.21	31.23	**31.24**	31.23
Tiffany → gray Text	92336	30.75	30.90	29.84	30.87	**30.95**	30.74
Tiffany → color Text	92336	-	-	51.10	51.75	**51.76**	51.74
zipped Tiffany → color Text	90912	-	-	51.51	**51.52**	**51.52**	**51.52**
zipped 3x Tiffany → color Text	272608	-	-	36.62	36.64	**36.65**	36.64
Text → gray Text	99888	30.81	33.43	28.52	33.78	**33.91**	33.85
Text → color Text	99888	-	-	55.83	**56.06**	**56.06**	**56.06**
zipped Text → color Text	97280	-	-	55.93	55.95	55.95	**55.96**
zipped 3x Text → color Text	291728	-	-	51.15	**51.19**	**51.19**	**51.19**
Average quality	**135492**	**30.93**	**32.55**	**43.13**	**43.72**	**43.78**	**43.75**

Table 1 – Quality comparison (PSNR values) with our results and those presented on reference [16]

Secret data → cover image	Method (average time in seconds)			
	LSB2	**GEN2**	**GA+PR**	**GA+PR'**
Jet → gray Lena	**1.592**	33.308	474.26	88.257
Jet → color Lena	**0.062**	17.127	134.057	13.014
zipped Jet → color Lena	**0.087**	17.021	47.020	26.017
zipped 3x Jet → color Lena	**0.109**	20.467	178.542	20.514
Sailboat → gray Lena	**1.542**	34.439	468.84	90.830
Sailboat → color Lena	**0.125**	27.115	164.582	58.580
zipped Sailboat → color Lena	**0.141**	26.552	118.998	31.656
Zipped 3x Sailboat → color Lena	**0.105**	21.837	360.842	21.859
Tiffany → gray Lena	**1.592**	34.129	468.75	65.284
Tiffany → color Lena	**0.086**	16.501	51.754	25.142
zipped Tiffany → color Lena	**0.079**	16.344	65.692	42.803
zipped 3x Tiffany → color Lena	**0.141**	29.061	139.952	29.302
Text → gray Lena	**1.622**	33.408	1295.94	96.278
Text → color Lena	**0.093**	17.474	22.025	20.451
zipped Text → color Lena	**0.094**	17.536	22.071	20.467
zipped 3x Text → color Lena	**0.124**	26.187	72.108	40.418
Jet → gray Baboon	**1.573**	33.298	611.34	64.512
Jet → color Baboon	**0.088**	16.716	108.046	43.060
zipped Jet → color Baboon	**0.083**	16.531	41.445	24.896
Zipped 3x Jet → color Baboon	**0.114**	21.523	257.541	21.434
Sailboat → gray Baboon	**1.602**	34.159	376.67	77.631
Sailboat → color Baboon	**0.094**	17.815	111.650	34.398
zipped Sailboat → color Baboon	**0.094**	17.628	141.538	28.345
zipped 3x Sailboat → color Baboon	**0.109**	21.091	80.075	21.498
Tiffany → gray Baboon	**1.622**	33.898	504.84	81.657
Tiffany → color Baboon	**0.078**	13.681	26.426	13.588
zipped Tiffany → color Baboon	**0.062**	13.385	22.448	13.478
zipped 3x Tiffany → color Baboon	**0.101**	20.093	62.376	20.339
Text → gray Baboon	**1.603**	43.412	849.51	77.612
Text → color Baboon	**0.067**	12.584	15.997	13.739
zipped Text → color Baboon	**0.071**	12.927	15.831	13.690
zipped 3x Text → color Baboon	**0.087**	16.595	43.470	24.561
Jet → gray Text	**1.542**	36.630	536.39	80.816
Jet → color Text	**0.083**	16.050	118.456	33.159
zipped Jet → color Text	**0.086**	16.608	33.330	24.790
zipped 3x Jet → color Text	**0.105**	21.191	206.953	21.425
Sailboat → gray Text	**1.572**	34.129	552.13	83.611
Sailboat → color Text	**0.091**	17.583	96.795	34.407
zipped Sailboat → color Text	**0.088**	17.334	120.811	40.094
zipped 3x Sailboat → color Text	**0.109**	22.326	242.431	22.360
Tiffany → gray Text	**1.542**	33.979	384.85	93.174
Tiffany → color Text	**0.085**	16.173	39.039	23.690
zipped Tiffany → color Text	**0.083**	16.336	36.276	23.573
zipped 3x Tiffany → color Text	**0.103**	20.529	114.652	20.635
Text → gray Text	**1.552**	33.187	403.10	81.437
Text → color Text	**0.071**	12.861	14.913	13.946
zipped Text → color Text	**0.072**	13.569	16.965	14.841
zipped 3x Text → color Text	**0.084**	16.783	39.555	31.246
Average time in seconds	**1.580**	**34.831**	**577,22**	**81.758**

Table 2 – Time spent in seconds with our tests

Secret data → cover image	Method (billions of instructions value)			
	LSB2	GEN2	GA+PR	GA+PR'
Jet → gray Lena	**3.61**	75.51	1075.15	200.08
Jet → color Lena	**1.08**	298.23	2334.33	226.61
zipped Jet → color Lena	**1.51**	296.39	818.76	453.03
zipped 3x Jet → color Lena	**1.90**	356.39	3108.95	357.21
Sailboat → gray Lena	**3.50**	78.07	1062.86	205.91
Sailboat → color Lena	**2.18**	472.15	2865.87	1020.05
zipped Sailboat → color Lena	**2.46**	462.35	2072.11	551.23
Zipped 3x Sailboat → color Lena	**1.83**	380.25	6283.34	380.63
Tiffany → gray Lena	**3.61**	77.37	1062.66	148.00
Tiffany → color Lena	**1.50**	287.33	901.19	437.80
zipped Tiffany → color Lena	**1.38**	284.60	1143.89	745.33
zipped 3x Tiffany → color Lena	**2.46**	506.04	2436.98	510.24
Text → gray Lena	**3.68**	75.74	2937.90	218.26
Text → color Lena	**1.62**	304.27	383.52	356.11
zipped Text → color Lena	**1.64**	305.35	384.32	356.39
zipped 3x Text → color Lena	**2.16**	455.99	1255.62	703.80
Jet → gray Baboon	**3.57**	75.49	1385.91	146.25
Jet → color Baboon	**1.53**	291.08	1881.40	749.80
zipped Jet → color Baboon	**1.45**	287.85	721.68	433.51
Zipped 3x Jet → color Baboon	**1.99**	374.78	4484.56	373.23
Sailboat → gray Baboon	**3.63**	77.44	853.91	175.99
Sailboat → color Baboon	**1.64**	310.21	1944.16	598.97
zipped Sailboat → color Baboon	**1.64**	306.96	2464.60	493.57
zipped 3x Sailboat → color Baboon	**1.90**	367.26	1394.35	374.34
Tiffany → gray Baboon	**3.68**	76.85	1144.47	185.12
Tiffany → color Baboon	**1.36**	238.23	460.16	236.61
zipped Tiffany → color Baboon	**1.08**	233.07	390.89	234.69
zipped 3x Tiffany → color Baboon	**1.76**	349.88	1086.15	354.16
Text → gray Baboon	**3.63**	98.42	1925.84	175.95
Text → color Baboon	**1.17**	219.13	278.56	239.24
zipped Text → color Baboon	**1.24**	225.10	275.67	238.38
zipped 3x Text → color Baboon	**1.51**	288.97	756.94	427.68
Jet → gray Text	**3.50**	83.04	1216.00	183.21
Jet → color Text	**1.45**	279.48	2062.67	577.40
zipped Jet → color Text	**1.50**	289.20	580.38	431.67
zipped 3x Jet → color Text	**1.83**	369.00	3603.67	373.07
Sailboat → gray Text	**3.56**	77.37	1251.68	189.55
Sailboat → color Text	**1.58**	306.17	1685.49	599.13
zipped Sailboat → color Text	**1.53**	301.84	2103.68	698.16
zipped 3x Sailboat → color Text	**1.90**	388.76	4221.45	389.35
Tiffany → gray Text	**3.50**	77.03	872.45	211.23
Tiffany → color Text	**1.48**	281.62	679.79	412.51
zipped Tiffany → color Text	**1.45**	284.46	631.67	410.48
zipped 3x Tiffany → color Text	**1.79**	357.47	1996.44	359.32
Text → gray Text	**3.52**	75.23	913.83	184.62
Text → color Text	**1.24**	223.95	259.68	242.84
zipped Text → color Text	**1.25**	236.28	295.41	258.43
zipped 3x Text → color Text	**1.46**	292.24	688.77	544.09
Average billions of instructions	**2.11**	**259.58**	**1555.00**	**382.78**

Table 3 – Time spent in BI units (billions of instructions) with our tests

In Figures 63 and 64 some visual results of image hiding are shown for comparison: The Lena (Figure 34) was used as cover image and the Text (Figure 40) was used as secret data. The Figure 63 shows the final results using the LSB substitution, while Figure 64 presents the results applying the path relinking approach. By examining Figure 63 carefully, it is possible to

perceive some text “written” in the background of the picture, showing the lack of quality of the LSB method. It does not happen in Figure 64. These differences are better seen in Figures 65 and 66, where two negative image difference maps are shown, each one presenting the differences between the original Lena cover image (Figure 34) and the other Lena stego images (Figures 63 and 64). Those image difference maps were scaled to 25x (all differences multiplied by 25), with an offset of -150 (-150 on all intensities, to focus on the bigger differences) and turned to negative, to show the differences in black (improving visual perception).

Figure 63 – Lena stego image produced by image hiding with LSB substitution (some text appears in background)

Figure 64 – Lena stego image obtained by image hiding with path relinking (no background text and appears smoother)

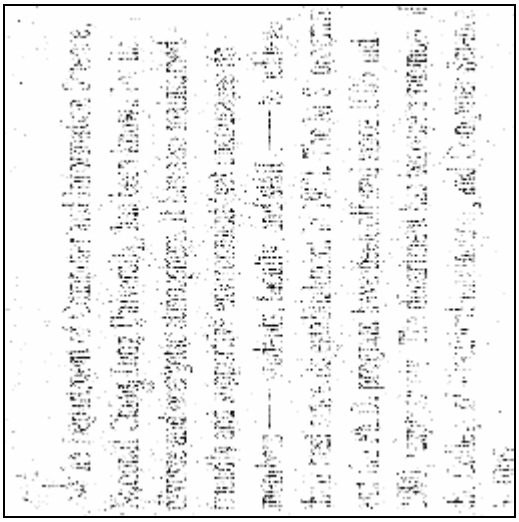


Figure 65 - The negative difference map between Figures 34 and 63 (scale 25x with an offset of -150). Note the text inside this difference map.

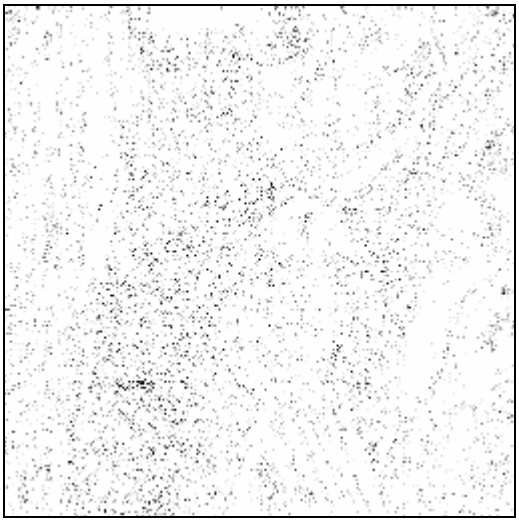


Figure 66 - The negative difference map between Figures 34 and 64 (scale 25x with an offset of -150)

6.4 Comparison with modulus functions results

In this section, our results are compared with those presented by [13], which use a different strategy on image hiding process. The tests were applied with the same images than [13]. The cover images are the 8-bits grayscale 512x512 pixels images of Lena and Baboon (Figures 34-36) and the 32-bits color versions of those images (Figures 50-51). The secret data are the 8-bits grayscale 256x512 pixels images of Jet, Sailboat, Tiffany and Splash (Figures 37-39 and 41). LSB [13] and GEN [13] are the results shown by [13] for simple LSB substitution and genetic algorithm. MOD [13] represents the results presented by [13] when using the modulus function approach. LSB2 and GEN2 are the results achieved by our implementation of least significant bits substitution and genetic algorithm, originally proposed by [16]. GA+PR is our new proposal, an implementation of the genetic algorithm with the path relinking approach, applied after the end of each genetic algorithm generation, by combining the 10 better solutions found until then (elite group). The GA+PR process is detailed in the section 6.3. GA+PR' is a modified version of GA+PR, using a simpler and faster path relinking approach, also detailed in the section 6.3.

Quality comparisons are presented on table 4. The arrow symbol (→) was used to indicate that the image before the symbol is being embedded inside the image after the symbol. The word "zipped", appearing before the arrow symbol, indicates the embedding image was submitted to WinZip compression application version 11.1, generating a lossless compressed file to be hidden inside the cover image. The word "3x", appearing before the arrow symbol, indicates three identical embedded images, instead of only one, were compressed by the WinZip application version 11.1, which also produces a lossless compressed file, of greater size. The words "gray" and "color", appearing after the arrow symbol, represent the use of 8-bits grayscale or 32-bits color versions of the cover image. The time results of our executions are shown on table 5, with cryptography time included for all tests with color cover images. The AES cryptography was not applied on tests with grayscale cover images. Time comparisons were not possible due to the differences between our computer system and the one used in [13].

Table 6 shows the time results converted to the BI unit values (billions of instructions), for future comparisons. Each result of our implementations is an average value for five executions with the same cover image and secret data.

The path relinking results (GA+PR and GA+PR') presented when dealing with "zipped 3x" secret data being hidden inside color cover images were obtained by using the method described in section 4.2 while all others color image steganography results used the section 4.2.1 alternative approach, which leads to better results when secret data is around one third or less of cover image hiding capacity.

Secret data → cover image	Bytes hidden	Method (PSNR result)						
		LSB [13]	LSB2	GEN [13]	GEN2	MOD [13]	GA+PR	GA+PR'
Jet → gray Lena	99888	32.02	32.08	34.16	32.61	**34.76**	32.79	32.71
Jet → color Lena	99888	-	48.64	-	48.95	-	**49.03**	48.99
zipped Jet → color Lena	97280	-	51.18	-	51.20	-	**51.21**	**51.21**
zipped 3x Jet → color Lena	291728	-	37.94	-	37.97	-	**37.99**	37.97
Sailboat → gray Lena	120064	32.28	31.93	34.08	32.43	**34.79**	32.58	32.57
Sailboat → color Lena	120064	-	47.62	-	48.15	-	**48.22**	48.20
zipped Sailboat → color Lena	116128	-	47.99	-	48.02	-	**48.03**	48.02
zipped 3x Sailboat → color Lena	348256	-	33.18	-	33.20	-	**33.21**	33.20
Tiffany → gray Lena	92336	31.30	31.96	32.90	32.82	**34.80**	32.90	32.87
Tiffany → color Lena	92336	-	51.40	-	51.43	-	**51.44**	**51.44**
zipped Tiffany → color Lena	90912	-	51.49	-	**51.50**	-	**51.50**	**51.50**
zipped 3x Tiffany → color Lena	272608	-	38.25	-	38.26	-	**38.27**	38.26
Splash → gray Lena	82256	32.27	32.02	32.44	32.45	**34.83**	32.52	32.43
Splash → color Lena	82256	-	51.92	-	**51.93**	-	**51.93**	**51.93**
zipped Splash → color Lena	81456	-	51.97	-	51.97	-	51.97	51.97
zipped 3x Splash → color Lena	244240	-	39.48	-	**39.51**	-	**39.51**	**39.51**
Jet → gray Baboon	99888	32.08	32.16	34.28	32.63	**34.81**	32.89	32.83
Jet → color Baboon	99888	-	48.44	-	48.99	-	**49.02**	49.00
zipped Jet → color Baboon	97280	-	51.18	-	**51.20**	-	**51.20**	**51.20**
zipped 3x Jet → color Baboon	291728	-	37.95	-	37.98	-	**37.99**	37.98
Sailboat → gray Baboon	120064	32.31	32.01	33.99	32.51	**34.82**	32.68	32.60
Sailboat → color Baboon	120064	-	47.94	-	**48.21**	-	**48.21**	48.20
zipped Sailboat → color Baboon	116128	-	47.98	-	48.02	-	**48.03**	48.02
zip 3x Sailboat → color Baboon	348256	-	33.23	-	33.25	-	**33.26**	33.25
Tiffany → gray Baboon	92336	31.30	32.06	32.97	32.82	**34.82**	33.02	33.01
Tiffany → color Baboon	92336	-	51.10	-	51.75	-	**51.76**	51.74
zipped Tiffany → color Baboon	90912	-	51.51	-	**51.52**	-	**51.52**	**51.52**
zip 3x Tiffany → color Baboon	272608	-	36.62	-	36.64	-	**36.65**	36.64
Splash → gray Baboon	82256	32.27	32.11	32.42	32.48	**34.79**	32.57	32.52
Splash → color Baboon	82256	-	51.91	-	**51.93**	-	**51.93**	**51.93**
zipped Splash → color Baboon	81456	-	51.97	-	51.97	-	51.97	51.97
zip 3x Splash → color Baboon	244240	-	39.48	-	39.51	-	**39.52**	39.51
Average quality	**145731**	**31.98**	**42.40**	**33.41**	**42.62**	**34.80**	**42.67**	**42.65**

Table 4 – Quality comparison (PSNR values) with our results and those presented on reference [13]

Secret data → cover image	Method (average time in seconds)			
	LSB2	GEN2	GA+PR	GA+PR'
Jet → gray Lena	**1.592**	33.308	474.26	88.257
Jet → color Lena	**0.062**	17.127	134.057	13.014
zipped Jet → color Lena	**0.087**	17.021	47.020	26.017
zipped 3x Jet → color Lena	**0.109**	20.467	178.542	20.514
Sailboat → gray Lena	**1.542**	34.439	468.84	90.830
Sailboat → color Lena	**0.125**	27.115	164.582	58.580
zipped Sailboat → color Lena	**0.141**	26.552	118.998	31.656
Zipped 3x Sailboat → color Lena	**0.105**	21.837	360.842	21.859
Tiffany → gray Lena	**1.592**	34.129	468.75	65.284
Tiffany → color Lena	**0.086**	16.501	51.754	25.142
zipped Tiffany → color Lena	**0.079**	16.344	65.692	42.803
zipped 3x Tiffany → color Lena	**0.141**	29.061	133.273	22.872
Splash → gray Lena	**1.632**	34.199	383.08	59.063
Splash → color Lena	**0.109**	24.146	74.157	40.730
zipped Splash → color Lena	**0.109**	23.936	76.059	35.172
zipped 3x Splash → color Lena	**0.140**	28.193	137.874	28.152
Jet → gray Baboon	**1.573**	33.298	611.34	64.512
Jet → color Baboon	**0.088**	16.716	108.046	43.060
zipped Jet → color Baboon	**0.083**	16.531	41.445	24.896
Zipped 3x Jet → color Baboon	**0.114**	21.523	257.541	21.434
Sailboat → gray Baboon	**1.602**	34.159	376.67	77.631
Sailboat → color Baboon	**0.094**	17.815	111.650	34.398
zipped Sailboat → color Baboon	**0.094**	17.628	141.538	28.345
zipped 3x Sailboat → color Baboon	**0.109**	21.091	80.075	21.498
Tiffany → gray Baboon	**1.622**	33.898	504.84	81.657
Tiffany → color Baboon	**0.078**	13.681	26.426	13.588
zipped Tiffany → color Baboon	**0.062**	13.385	22.448	13.478
zipped 3x Tiffany → color Baboon	**0.101**	20.093	62.376	20.339
Splash → gray Baboon	**1.572**	34.049	316.800	72.424
Splash → color Baboon	**0.099**	20.041	36.964	19.988
zipped Splash → color Baboon	**0.112**	19.914	38.831	19.566
zip 3x Splash → color Baboon	**0.151**	28.132	213.346	29.756
Average time in seconds	**0.475**	**23.948**	**196.504**	**39.266**

Table 5 – Time spent in seconds with our tests

In Figures 67 and 68 some visual results of image hiding are shown for illustration: The Baboon (Figure 35) was used as cover image and the Tiffany (Figure 39) was used as secret data. The Figure 67 shows the stego image produced after steganography with genetic algorithm, while Figure 68 presents the stego image obtained after steganography applying the path relinking approach. The two pictures are very similar and is very hard to detect any visual difference, given the small difference in PSNR quality values between both methods. This difference is better seen in Figures 69 and 70, where two negative image difference maps are presented, each one presenting the differences between the cover image Baboon (Figure 35) and the other Baboon stego images (Figures 67 and 68). Those image difference maps were scaled to 25x (all differences multiplied by 25), with an offset of -100 (-100 on all intensities, to focus on the bigger differences) and turned to negative, to show the differences in black (improving

visual perception). In fact, when the PSNR values differ only less than a single point between the results obtained (32.82 for genetic algorithm only and 33.02 for genetic algorithm with path relinking; in this case), it is really hard to perceive any visual difference without the help of difference image maps.

Secret data → cover image	Method (billions of instructions value)			
	LSB2	GEN2	GA+PR	GA+PR'
Jet → gray Lena	**3.61**	75.51	1075.15	200.08
Jet → color Lena	**1.08**	298.23	2334.33	226.61
zipped Jet → color Lena	**1.51**	296.39	818.76	453.03
zipped 3x Jet → color Lena	**1.90**	356.39	3108.95	357.21
Sailboat → gray Lena	**3.50**	78.07	1062.86	205.91
Sailboat → color Lena	**2.18**	472.15	2865.87	1020.05
zipped Sailboat → color Lena	**2.46**	462.35	2072.11	551.23
Zipped 3x Sailboat → color Lena	**1.83**	380.25	6283.34	380.63
Tiffany → gray Lena	**3.61**	77.37	1062.66	148.00
Tiffany → color Lena	**1.50**	287.33	901.19	437.80
zipped Tiffany → color Lena	**1.38**	284.60	1143.89	745.33
zipped 3x Tiffany → color Lena	**2.46**	506.04	2320.68	398.27
Splash → gray Lena	**3.70**	77.53	868.44	133.90
Splash → color Lena	**1.90**	420.45	1291.30	709.23
zipped Splash → color Lena	**1.90**	416.80	1324.42	612.45
zipped 3x Splash → color Lena	**2.44**	490.92	2400.80	490.21
Jet → gray Baboon	**3.57**	75.49	1385.91	146.25
Jet → color Baboon	**1.53**	291.08	1881.40	749.80
zipped Jet → color Baboon	**1.45**	287.85	721.68	433.51
Zipped 3x Jet → color Baboon	**1.99**	374.78	4484.56	373.23
Sailboat → gray Baboon	**3.63**	77.44	853.91	175.99
Sailboat → color Baboon	**1.64**	310.21	1944.16	598.97
zipped Sailboat → color Baboon	**1.64**	306.96	2464.60	493.57
zipped 3x Sailboat → color Baboon	**1.90**	367.26	1394.35	374.34
Tiffany → gray Baboon	**3.68**	76.85	1144.47	185.12
Tiffany → color Baboon	**1.36**	238.23	460.16	236.61
zipped Tiffany → color Baboon	**1.08**	233.07	390.89	234.69
zipped 3x Tiffany → color Baboon	**1.76**	349.88	1086.15	354.16
Splash → gray Baboon	**3.56**	77.19	718.19	164.19
Splash → color Baboon	**1.72**	348.97	643.65	348.05
zipped Splash → color Baboon	**1.95**	346.76	676.16	340.70
zip 3x Splash → color Baboon	**2.63**	489.86	3714.99	518.14
Average billions of instructions	**2.25**	**288.51**	**1715.62**	**399.91**

Table 6 – Time spent in BI units (billions of instructions) with our tests

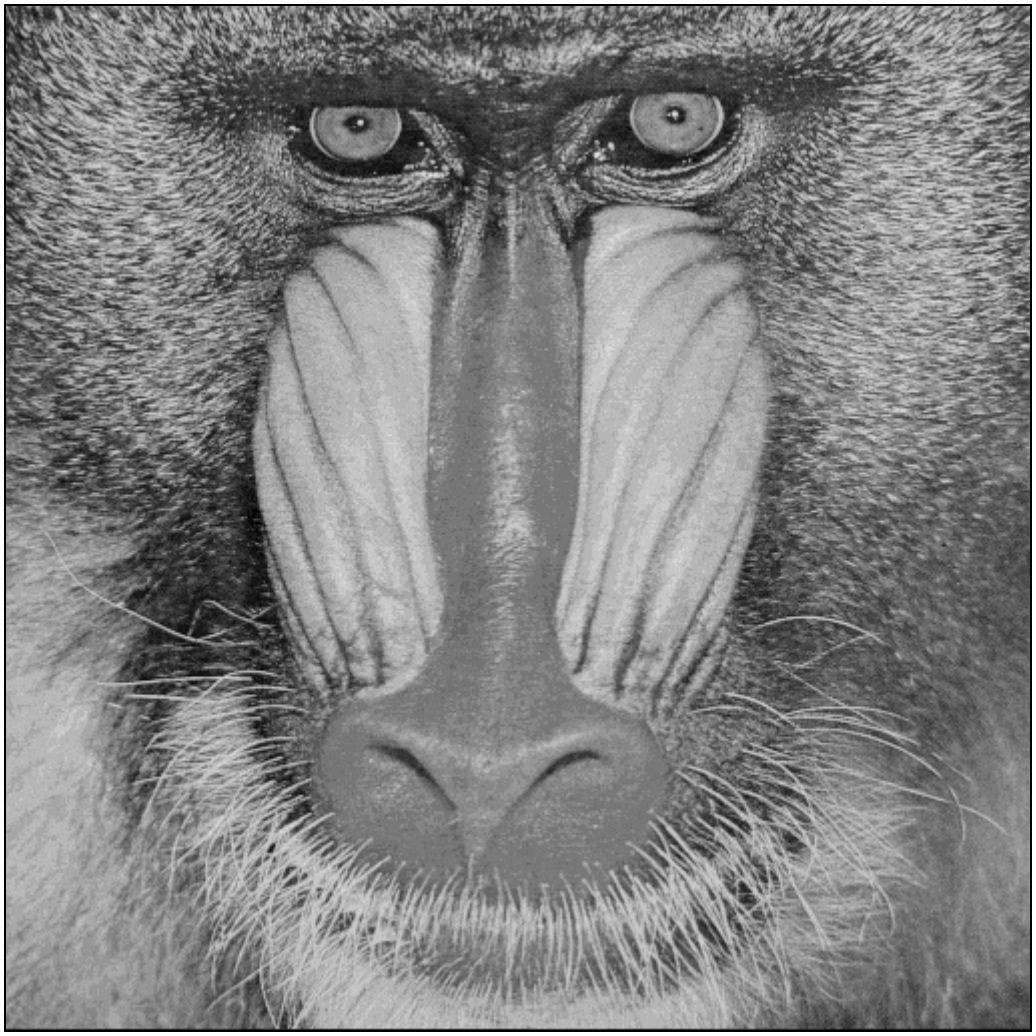


Figure 67 – Baboon stego image produced by image hiding with genetic algorithm

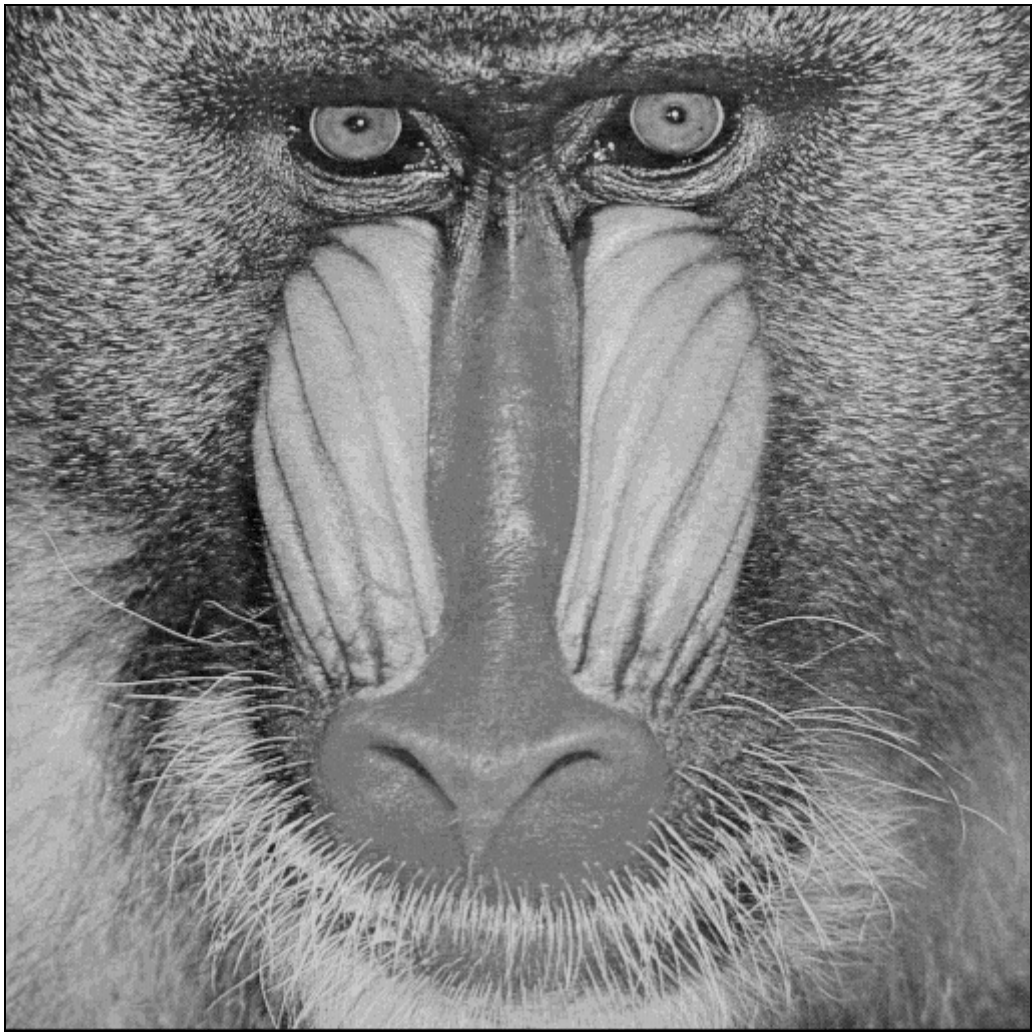


Figure 68 – Baboon stego image obtained by image hiding with path relinking (not much visually different from Figure 43)

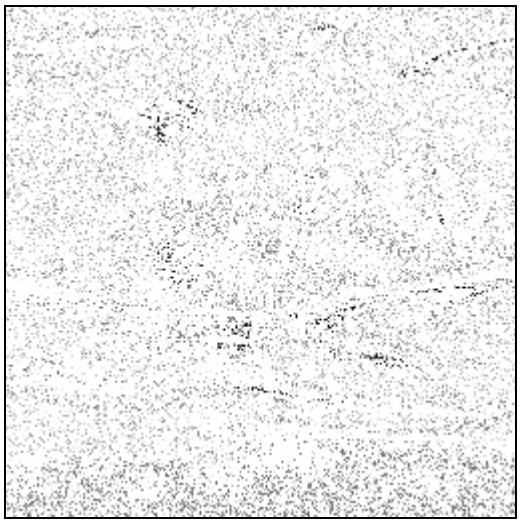


Figure 69 - The negative difference map between Figures 35 and 67 (scale 25x with an offset of -100). Note this map is slightly "noisy" than Figure 70.

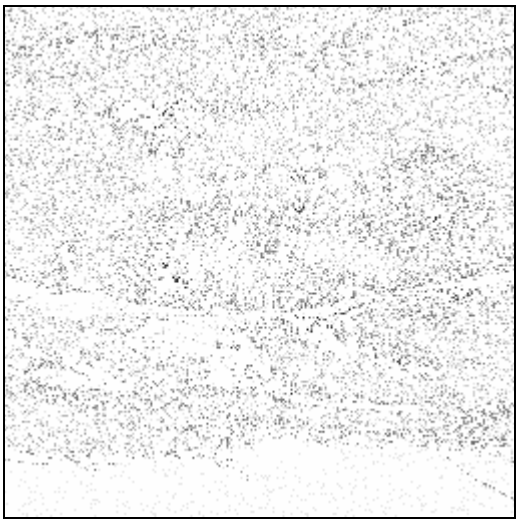


Figure 70 - The negative difference map between Figures 35 and 68 (scale 25x with an offset of -100)

6.5 Comparison with dynamic programming strategy results

In this section our results are also compared with those presented on [14], where another approach to optimize the search for the best LSB substitution possible on image hiding process is used. The tests were applied with the same images than [14]. The cover images are the 8-bits grayscale 512x512 pixels images of Lena, Peppers and Barbara (Figures 34 and 43-44) and the 32-bits color versions of those images (Figures 50 and 53-54). The secret data are the 8-bits grayscale 256x512 pixels images of Jet, Tiffany and Fishing Boat (Figures 37, 39 and 42). The Fishing Boat image was not shown in [14] with its resized 256 x 512 pixels format. We contacted the authors of [14] asking about this and they oriented us to use the upper half of the 512 x 512 Fishing Boat cover image from [44] to produce the 256 x 512 image (Figure 42). GEN[14] are the results shown by [14] with genetic algorithm and DYN[14] are the results presented by [14] using the optimal LSB substitution with dynamic programming strategy. LSB2 and GEN2 are the results achieved by our implementation of least significant bits substitution and genetic algorithm, originally proposed by [16]. GA+PR is our new proposal, an implementation of the genetic algorithm with the path relinking approach, applied after the end of each genetic algorithm generation, by combining the 10 better solutions found until then (elite group). The GA+PR process is detailed in the section 6.3. GA+PR' is a modified version of GA+PR, using a simpler and faster path relinking approach, also detailed in the section 6.3.

Quality comparisons are presented on table 7. The results presented on [14] were converted from MSE to PSNR to allow correct comparisons with our results. Again, the arrow symbol (→) was used to indicate that the image before the symbol is being embedded inside the image after the symbol. The word "zipped", appearing before the arrow symbol, indicates the embedding image was submitted to WinZip compression application version 11.1, generating a lossless compressed file to be hidden inside the cover image. The word "3x", appearing before the arrow symbol, indicates three identical embedded images, instead of only one, were compressed by the WinZip application version 11.1, which also produces a lossless compressed file, of greater size. The words "gray" and "color", appearing after the arrow symbol, represent

the use of 8-bits grayscale or 32-bits color versions of the cover image. The time results of our executions are shown on table 8, with cryptography time included for all tests with color cover images. The AES cryptography was not applied on tests with grayscale cover images. Time comparisons were not possible due to the differences between the computer systems used in [14] and ours. Table 9 shows the time results converted to the BI unit values (billions of instructions), for future comparisons. Each result of our implementations is an average value for five executions with the same cover image and secret data.

Secret data → cover image	Bytes hidden	Method (PSNR result)					
		LSB2	GEN [14]	GEN2	DYN [14]	GA+PR	GA+PR'
Jet → gray Lena	99888	32.08	32.79	32.63	**32.90**	32.76	32.71
Jet → color Lena	99888	48.64	-	48.95	-	**49.03**	48.99
zipped Jet → color Lena	97280	51.18	-	51.20	-	**51.21**	**51.21**
zipped 3x Jet → color Lena	291728	37.94	-	37.97	-	**37.99**	37.97
Tiffany → gray Lena	92336	31.96	32.73	32.84	32.92	**32.93**	32.87
Tiffany → color Lena	92336	51.40	-	51.43	-	**51.44**	**51.44**
zipped Tiffany → color Lena	90912	51.49	-	**51.50**	-	**51.50**	**51.50**
zipped 3x Tiffany → color Lena	272608	38.25	-	38.26	-	**38.27**	38.26
Fishing Boat → gray Lena	50736	31.83	32.55	33.78	32.71	**34.08**	33.66
Fishing Boat → color Lena	50736	54.00	-	**54.03**	-	**54.03**	**54.03**
zipped Fishing Boat → color Lena	50656	54.02	-	**54.04**	-	**54.04**	**54.04**
zipped 3x Fish Boat → color Lena	151856	45.81	-	**45.83**	-	**45.83**	**45.83**
Jet → gray Peepers	99888	32.17	32.72	32.76	32.91	**32.95**	32.89
Jet → color Peepers	99888	48.47	-	48.98	-	**49.03**	49.00
zipped Jet → color Peepers	97280	51.18	-	51.20	-	**51.21**	**51.21**
zipped 3x Jet → color Peepers	291728	37.85	-	**37.86**	-	**37.86**	**37.86**
Tiffany → gray Peepers	92336	32.09	32.75	32.96	32.93	**33.08**	33.01
Tiffany → color Peepers	92336	51.44	-	**51.45**	-	**51.45**	**51.45**
zipped Tiffany → color Peepers	90912	51.48	-	51.49	-	**51.50**	**51.50**
zipped 3x Tiffany → color Peepers	272608	38.13	-	38.14	-	**38.15**	**38.15**
Fishing Boat → gray Peepers	50736	32.02	32.50	33.67	32.72	**34.25**	33.51
Fishing Boat → color Peepers	50736	54.04	-	**54.05**	-	**54.05**	**54.05**
zipped Fish Boat → color Peepers	50656	54.02	-	**54.04**	-	**54.04**	**54.04**
zipped 3x FishBoat→ color Peepers	151856	45.75	-	45.76	-	**45.77**	**45.77**
Jet → gray Barbara	99888	32.10	32.73	32.63	**32.90**	32.81	32.73
Jet → color Barbara	99888	48.44	-	48.99	-	**49.02**	**49.02**
zipped Jet → color Barbara	97280	51.19	-	**51.20**	-	**51.20**	**51.20**
zipped 3x Jet → color Barbara	291728	37.96	-	37.98	-	**37.99**	37.98
Tiffany → gray Barbara	92336	31.99	32.67	32.81	32.92	**32.96**	32.88
Tiffany → color Barbara	92336	51.41	-	**51.43**	-	**51.43**	**51.43**
zipped Tiffany → color Barbara	90912	51.48	-	51.49	-	**51.50**	**51.50**
zipped 3x Tiffany → color Barbara	272608	38.25	-	38.27	-	**38.28**	38.27
Fishing Boat → gray Barbara	50736	31.94	32.52	33.92	32.71	**34.15**	33.68
Fishing Boat → color Barbara	50736	54.02	-	**54.03**	-	**54.03**	**54.03**
zipped Fishing Boat → color Barbara	50656	54.03	-	54.03	-	54.03	54.03
zipped 3x FishBoat→ color Barbara	151856	45.83	-	45.83	-	45.83	45.83
Average quality	**120080**	**44.05**	**32.66**	**44.37**	**32.85**	**44.44**	**44.38**

Table 7 – Quality comparison (PSNR values) with our results and those presented on reference [14]

The path relinking results (GA+PR and GA+PR') presented when dealing with "zipped 3x" secret data being hidden inside color cover images were obtained by using the method

described in section 4.2 while all others color image steganography results used the section 4.2.1 alternative approach, which leads to better results when secret data is around one third or less of cover image hiding capacity.

Secret data → cover image	Method (average time in seconds)			
	LSB2	**GEN2**	**GA+PR**	**GA+PR'**
Jet → gray Lena	**1.572**	33.728	367.809	88.257
Jet → color Lena	**0.062**	17.127	134.057	13.014
zipped Jet → color Lena	**0.087**	17.021	47.020	26.017
zipped 3x Jet → color Lena	**0.109**	20.467	178.542	20.514
Tiffany → gray Lena	**1.672**	42.751	492.278	65.284
Tiffany → color Lena	**0.086**	16.501	51.754	25.142
zipped Tiffany → color Lena	**0.079**	16.344	65.692	42.803
zipped 3x Tiffany → color Lena	**0.141**	29.061	133.273	22.872
Fishing Boat → gray Lena	**1.732**	33.779	298.149	86.975
Fishing Boat → color Lena	**0.093**	19.856	37.282	27.993
zipped Fishing Boat → color Lena	**0.094**	19.904	36.580	26.932
zipped 3x Fishing Boat → color Lena	**0.125**	24.780	63.373	24.545
Jet → gray Peepers	**1.593**	33.829	319.650	87.976
Jet → color Peepers	**0.125**	24.483	181.327	62.216
zipped Jet → color Peepers	**0.125**	26.126	81.410	43.005
zipped 3x Jet → color Peepers	**0.156**	31.272	357.352	32.665
Tiffany → gray Peepers	**1.613**	33.568	457.257	66.917
Tiffany → color Peepers	**0.125**	26.563	80.487	44.256
zipped Tiffany → color Peepers	**0.141**	26.359	74.761	45.227
zipped 3x Tiffany → color Peepers	**0.141**	29.629	259.171	30.521
Fishing Boat → gray Peepers	**1.572**	34.359	479.059	70.712
Fishing Boat → color Peepers	**0.110**	19.927	36.090	27.320
zipped Fishing Boat → color Peepers	**0.094**	19.907	39.328	27.594
zipped 3x Fishing Boat→ color Peepers	**0.125**	24.360	77.953	24.625
Jet → gray Barbara	**1.572**	33.919	378.224	67.667
Jet → color Barbara	**0.125**	24.253	158.733	56.576
zipped Jet → color Barbara	**0.140**	26.029	73.675	40.883
zipped 3x Jet → color Barbara	**0.141**	30.517	114.074	30.595
Tiffany → gray Barbara	**1.582**	34.399	465.469	75.648
Tiffany → color Barbara	**0.125**	25.343	80.206	38.311
zipped Tiffany → color Barbara	**0.125**	25.375	74.412	42.171
zipped 3x Tiffany → color Barbara	**0.141**	29.145	156.610	28.666
Fishing Boat → gray Barbara	**1.573**	33.759	431.120	72.955
Fishing Boat → color Barbara	**0.092**	19.768	37.593	27.016
zipped Fishing Boat → color Barbara	**0.093**	19.852	36.666	27.102
zipped 3x Fishing Boat→ color Barbara	**0.124**	24.423	75.652	24.361
Average time in seconds	**0.489**	**26.347**	**178.669**	**43.481**

Table 8 – Time spent in seconds with our tests

Secret data → cover image	Method (billions of instructions value)			
	LSB2	GEN2	GA+PR	GA+PR'
Jet → gray Lena	**3.56**	76.46	833.82	200.08
Jet → color Lena	**1.08**	298.23	2334.33	226.61
zipped Jet → color Lena	**1.51**	296.39	818.76	453.03
zipped 3x Jet → color Lena	**1.90**	356.39	3108.95	357.21
Tiffany → gray Lena	**3.79**	96.92	1115.99	148.00
Tiffany → color Lena	**1.50**	287.33	901.19	437.80
zipped Tiffany → color Lena	**1.38**	284.60	1143.89	745.33
zipped 3x Tiffany → color Lena	**2.46**	506.04	2320.68	398.27
Fishing Boat → gray Lena	**3.93**	76.58	675.90	197.17
Fishing Boat → color Lena	**1.62**	345.75	649.19	487.44
zipped Fishing Boat → color Lena	**1.64**	346.59	636.97	468.97
zipped 3x Fishing Boat → color Lena	**2.18**	431.49	1103.51	427.40
Jet → gray Peepers	**3.61**	76.69	724.65	199.44
Jet → color Peepers	**2.18**	426.32	3157.45	1083.37
zipped Jet → color Peepers	**2.18**	454.93	1417.59	748.85
zipped 3x Jet → color Peepers	**2.72**	544.54	6222.57	568.80
Tiffany → gray Peepers	**3.66**	76.10	1036.60	151.70
Tiffany → color Peepers	**2.18**	462.54	1401.52	770.63
zipped Tiffany → color Peepers	**2.46**	458.99	1301.81	787.54
zipped 3x Tiffany → color Peepers	**2.46**	515.93	4512.94	531.46
Fishing Boat → gray Peepers	**3.56**	77.89	1086.03	160.30
Fishing Boat → color Peepers	**1.92**	346.99	628.44	475.72
zipped Fishing Boat → color Peepers	**1.64**	346.64	684.82	480.49
zipped 3x Fishing Boat→ color Peepers	**2.18**	424.18	1357.40	428.80
Jet → gray Barbara	**3.56**	76.89	857.43	153.40
Jet → color Barbara	**2.18**	422.32	2764.02	985.16
zipped Jet → color Barbara	**2.44**	453.24	1282.90	711.90
zipped 3x Jet → color Barbara	**2.46**	531.39	1986.37	532.75
Tiffany → gray Barbara	**3.59**	77.98	1055.22	171.49
Tiffany → color Barbara	**2.18**	441.30	1396.63	667.11
zipped Tiffany → color Barbara	**2.18**	441.85	1295.74	734.32
zipped 3x Tiffany → color Barbara	**2.46**	507.50	2727.05	499.16
Fishing Boat → gray Barbara	**3.57**	76.53	977.35	165.39
Fishing Boat → color Barbara	**1.60**	344.22	654.61	470.43
zipped Fishing Boat → color Barbara	**1.62**	345.68	638.47	471.93
zipped 3x Fishing Boat→ color Barbara	**2.16**	425.28	1317.33	424.20
Average billions of instructions	**2.42**	**326.63**	**1559.11**	**470.05**

Table 9 – Time spent in BI units (billions of instructions) with our tests

6.6 Comparison with cryptosystem and modulus operations results

In this section, our results are compared with the ones presented on [17], that applies a cryptosystem with modulus operations on steganography process. The tests were done with the same images than [17]. The cover images are the 8-bits grayscale 512x512 pixels images of Lena, Baboon, Peppers and Jet (Figures 34-35, 43 and 45) and the 32-bits color versions of those images (Figures 50-51, 53 and 55). The secret data are the 8-bits grayscale squeezed 256x512 pixels images of Jet, Baboon, Peepers and Lena (Figures 46-49). Note that the Figure 46 (squeezed Jet) is slightly different from the Figure 37 (Jet). Both are 256x512 pixels images, derived from the original 512x512 pixels Jet image (Figure 45). The main difference is that in

[17], all the four images used as secret data were squeezed to attain the 256x512 size, while in [13, 14, 16] those images were obtained by cutting a 256x512 part of the original picture. STE[17] are the results presented by [17] with the cryptosystem and modulus operations method. LSB2 and GEN2 are the results achieved by our implementation of least significant bits substitution and genetic algorithm, originally proposed by [16]. GA+PR is our new proposal, an implementation of the genetic algorithm with the path relinking approach, applied after the end of each genetic algorithm generation, by combining the 10 better solutions found until then (elite group). The GA+PR process is detailed in the section 6.3. GA+PR' is a modified version of GA+PR, using a simpler and faster path relinking approach, also detailed in the section 6.3.

Quality comparisons are presented on table 10. The arrow symbol (→) was used to indicate that the image before the symbol is being embedded inside the image after the symbol. The word "zipped", appearing before the arrow symbol, indicates the embedding image was submitted to WinZip compression application version 11.1, generating a lossless compressed file to be hidden inside the cover image. The word "3x", appearing before the arrow symbol, indicates three identical embedded images, instead of only one, were compressed by the WinZip application version 11.1, which also produces a lossless compressed file, of greater size. The words "gray" and "color", appearing after the arrow symbol, represent the use of 8-bits grayscale or 32-bits color versions of the cover image. The time results of our executions are shown on table 11, with cryptography time included for all tests with color cover images. The AES cryptography was not applied on tests with grayscale cover images. Time comparisons were not possible because the authors only commented some details about individual mathematical operations time falling into microseconds and milliseconds range, but apparently have not shown any total time results, so we will be presenting only ours. Table 12 shows the time results converted to the BI unit values (billions of instructions), for future comparisons. Each result of our implementations is an average value for five executions with the same cover image and secret data.

The path relinking results (GA+PR and GA+PR') presented when dealing with "zipped 3x" secret data being hidden inside color cover images were obtained by using the method

described in section 4.2 while all others color image steganography results used the section 4.2.1 alternative approach, which leads to better results when secret data is around one third or less of cover image hiding capacity.

Secret data → cover image	Bytes hidden	Method (PSNR result)				
		LSB2	GEN2	STE [17]	GA+PR	GA+PR'
squeezed Jet → gray Lena	112432	31.98	32.67	31.05	**32.74**	32.66
squeezed Jet → color Lena	112432	47.93	48.46	-	**48.52**	48.48
squeezed Baboon → gray Lena	160944	31.77	32.36	31.28	**32.49**	32.44
squeezed Baboon → color Lena	160944	45.57	**45.58**	-	**45.58**	**45.58**
squeezed Peepers → gray Lena	119952	31.96	32.25	31.33	**32.40**	32.29
squeezed Peepers → color Lena	119952	47.63	48.21	-	**48.24**	48.22
squeezed Lena → gray Lena	125920	31.97	32.86	31.27	**33.13**	32.88
squeezed Lena → color Lena	125920	47.39	48.01	-	**48.04**	**48.04**
squeezed Jet → gray Baboon	112432	32.03	32.72	31.09	**32.87**	32.74
squeezed Jet → color Baboon	112432	47.93	48.49	-	**48.52**	48.50
squeezed Baboon → gray Baboon	160944	31.90	32.42	31.24	**32.69**	32.59
squeezed Baboon → color Baboon	160944	45.57	45.59	-	**45.59**	45.58
squeezed Peepers → gray Baboon	119952	32.02	32.36	31.28	**32.45**	32.39
squeezed Peepers → color Baboon	119952	47.63	48.17	-	**48.24**	48.20
squeezed Lena → gray Baboon	125920	31.99	32.38	31.25	**32.52**	32.51
squeezed Lena → color Baboon	125920	47.40	48.03	-	**48.05**	48.04
squeezed Jet → gray Peepers	112432	32.11	32.74	31.42	32.77	**32.85**
squeezed Jet → color Peepers	112432	47.97	48.48	-	**48.52**	48.49
squeezed Baboon → gray Peepers	160944	31.91	32.50	31.60	**32.66**	32.58
squeezed Baboon → color Peepers	160944	45.52	**45.53**	-	**45.53**	**45.53**
squeezed Peepers → gray Peepers	119952	32.19	32.50	31.65	**32.60**	32.58
squeezed Peepers → color Peepers	119952	47.66	48.20	-	**48.24**	48.22
squeezed Lena → gray Peepers	125920	32.02	32.44	31.61	**32.58**	32.50
squeezed Lena → color Peepers	125920	47.43	48.01	-	**48.05**	48.01
squeezed Jet → gray Jet	112432	32.04	33.18	30.24	**33.84**	33.59
squeezed Jet → color Jet	112432	47.94	48.44	-	**48.52**	48.49
squeezed Baboon → gray Jet	160944	31.84	32.38	30.23	**32.66**	32.60
squeezed Baboon → color Jet	160944	45.55	**45.56**	-	**45.56**	**45.56**
squeezed Peepers → gray Jet	119952	31.98	32.38	30.27	**32.56**	32.46
squeezed Peepers → color Jet	119952	47.64	48.18	-	**48.24**	48.22
squeezed Lena → gray Jet	125920	31.94	32.45	30.29	**32.59**	32.54
squeezed Lena → color Jet	125920	47.42	48.04	-	**48.05**	48.01
Average quality	**129812**	**39.56**	**40.05**	**31.07**	**40.16**	**40.11**

Table 10 – Quality comparison (PSNR values) with our results and those presented on reference [17]

Secret data → cover image	Method (average time in seconds)			
	LSB2	GEN2	GA+PR	GA+PR'
squeezed Jet → gray Lena	**1.552**	34.219	477.32	88.627
squeezed Jet → color Lena	**0.093**	16.489	86.222	31.403
squeezed Baboon → gray Lena	**1.613**	36.182	333.95	73.766
squeezed Baboon → color Lena	**0.078**	16.879	44.288	16.848
squeezed Peepers → gray Lena	**1.572**	35.151	420.10	63.302
squeezed Peepers → color Lena	**0.078**	16.801	103.577	30.435
squeezed Lena → gray Lena	**1.583**	39.898	408.02	55.790
squeezed Lena → color Lena	**0.094**	17.191	133.492	32.900
squeezed Jet → gray Baboon	**1.562**	34.170	378.40	73.676
squeezed Jet → color Baboon	**0.093**	16.318	108.576	36.520
squeezed Baboon → gray Baboon	**1.583**	34.149	382.15	99.102
squeezed Baboon → color Baboon	**0.094**	16.739	45.271	19.079
squeezed Peepers → gray Baboon	**1.563**	33.989	401.17	65.725
squeezed Peepers → color Baboon	**0.094**	16.988	104.186	30.904
squeezed Lena → gray Baboon	**1.662**	34.160	440.00	87.375
squeezed Lena → color Baboon	**0.093**	17.269	96.892	32.807
squeezed Jet → gray Peepers	**1.763**	33.989	552.68	72.144
squeezed Jet → color Peepers	**0.093**	16.427	109.527	33.337
squeezed Baboon → gray Peepers	**1.542**	34.109	434.10	78.523
squeezed Baboon → color Peepers	**0.078**	16.895	44.944	16.879
squeezed Peepers → gray Peepers	**1.612**	34.099	367.75	80.015
squeezed Peepers → color Peepers	**0.093**	17.051	114.862	33.868
squeezed Lena → gray Peepers	**1.562**	34.660	452.02	57.883
squeezed Lena → color Peepers	**0.078**	17.129	86.424	32.604
squeezed Jet → gray Jet	**1.562**	33.658	401.36	84.352
squeezed Jet → color Jet	**0.094**	16.349	111.369	32.136
squeezed Baboon → gray Jet	**1.552**	36.783	480.21	70.321
squeezed Baboon → color Jet	**0.094**	16.739	45.739	16.723
squeezed Peepers → gray Jet	**1.572**	34.419	414.84	67.046
squeezed Peepers → color Jet	**0.078**	16.895	86.612	32.963
squeezed Lena → gray Jet	**1.613**	34.049	449.23	98.682
squeezed Lena → color Jet	**0.094**	17.191	102.024	27.908
Average time	**0.840**	**25.845**	**256.791**	**52.301**

Table 11 – Time spent in seconds with our tests

Secret data → cover image	Method (billions of instructions value)			
	LSB2	GEN2	GA+PR	GA+PR'
squeezed Jet → gray Lena	**3.52**	77.57	1082.08	200.92
squeezed Jet → color Lena	**1.62**	287.12	1501.38	546.82
squeezed Baboon → gray Lena	**3.66**	82.02	757.06	167.23
squeezed Baboon → color Lena	**1.36**	293.91	771.19	293.37
squeezed Peepers → gray Lena	**3.56**	79.69	952.37	143.51
squeezed Peepers → color Lena	**1.36**	292.56	1803.59	529.96
squeezed Lena → gray Lena	**3.59**	90.45	924.98	126.48
squeezed Lena → color Lena	**1.64**	299.35	2324.50	572.89
squeezed Jet → gray Baboon	**3.54**	77.46	857.83	167.02
squeezed Jet → color Baboon	**1.62**	284.15	1890.63	635.92
Squeezed Baboon → gray Baboon	**3.59**	77.42	866.33	224.66
Squeezed Baboon → color Baboon	**1.64**	291.48	788.30	332.22
Squeezed Peepers → gray Baboon	**3.54**	77.05	909.45	149.00
Squeezed Peepers → color Baboon	**1.64**	295.81	1814.19	538.13
squeezed Lena → gray Baboon	**3.77**	77.44	997.48	198.08
squeezed Lena → color Baboon	**1.62**	300.71	1687.18	571.27
squeezed Jet → gray Peepers	**4.00**	77.05	1252.93	163.55
squeezed Jet → color Peepers	**1.62**	286.04	1907.19	580.50
Squeezed Baboon → gray Peepers	**3.50**	77.33	984.10	178.01
Squeezed Baboon → color Peepers	**1.36**	294.19	782.61	293.91
Squeezed Peepers → gray Peepers	**3.65**	77.30	833.69	181.39
squeezed Peepers → color Peepers	**1.62**	296.91	2000.09	589.74
squeezed Lena → gray Peepers	**3.54**	78.57	1024.73	131.22
squeezed Lena → color Peepers	**1.36**	298.27	1504.90	567.73
squeezed Jet → gray Jet	**3.54**	76.30	909.88	191.23
squeezed Jet → color Jet	**1.64**	284.69	1939.27	559.58
squeezed Baboon → gray Jet	**3.52**	83.39	1088.64	159.42
squeezed Baboon → color Jet	**1.64**	291.48	796.45	291.20
squeezed Peepers → gray Jet	**3.56**	78.03	940.44	151.99
squeezed Peepers → color Jet	**1.36**	294.19	1508.17	573.98
squeezed Lena → gray Jet	**3.66**	77.19	1018.40	223.71
squeezed Lena → color Jet	**1.64**	299.35	1776.54	485.96
Average billions of instructions	**2.58**	**186.08**	**1256.14**	**335.02**

Table 12 – Time spent in BI units (billions of instructions) with our tests

6.7 Comparison with color quantization and DES cryptosystem

In this section, our results are compared with the ones presented on Table 1 from [20], that apply a color quantization process, reducing the number of colors inside the image to be embedded to 256, using the DES cryptosystem thereafter. The tests were done with the same images than [20] as cover images, with no exception. These cover images are the 32-bits color, 512x512 pixels images of Lena, Baboon, Peppers, Jet, House and Sailboat (Figures 50, 51, 54 and 55-57). The same figures were also used as embedded images, after being color quantized to 256 colors. The 256-color quantization images were obtained by decreasing the bits per pixel value to 8, reducing the maximum number of colors to 256 and converting the images files to the portable network graphics format (png). CQ[20] are the results presented by [20] with the color quantization and DES cryptosystem. LSB2 and GEN2 are the results achieved by our

implementation of least significant bits substitution and genetic algorithm, originally proposed by [16]. GA+PR is our new proposal, an implementation of the genetic algorithm with the path relinking approach, applied after the end of each genetic algorithm generation, by combining the 10 better solutions found until then (elite group). The GA+PR process is detailed in the section 6.3. GA+PR' is a modified version of GA+PR, using a simpler and faster path relinking approach, also detailed in the section 6.3.

Quality comparisons are presented on table 13. The arrow symbol (→) was used to indicate that the image before the symbol is being embedded inside the image after the symbol. The word "zipped", appearing before the arrow symbol, indicates the embedding image was submitted to WinZip compression application version 11.1, generating a lossless compressed file to be hidden inside the cover image. The word "2x", appearing before the arrow symbol, indicates two identical embedded images, instead of only one, were compressed by the WinZip application version 11.1, which also produces a lossless compressed file, of greater size. The time results of our executions are shown on table 14, with cryptography time included for all tests. Time comparisons were not possible because the authors not commented details about time results, so we will be presenting only ours. Table 15 shows the time results converted to the BI unit values (billions of instructions), for future comparisons. Each result of our implementations is an average value for five executions with the same cover image and secret data.

The path relinking results (GA+PR and GA+PR') presented when dealing with quantized color Jet or quantized color House secret data being hidden inside cover images were obtained by using the alternative approach described in section 4.2.1, which leads to better results when secret data is around one third or less of cover image hiding capacity. The other results were obtained by the method described on section 4.2.

Secret data → cover image	Bytes hidden	Method (PSNR result)				
		LSB2	GEN2	CQ[20]	GA+PR	GA+PR'
quantized color Jet → color Jet	110704	48.15	48.26	41.84	**48.28**	48.27
zipped quantized color Jet 2x → color Jet	221680	41.09	41.10	-	**41.11**	**41.11**
quantized color Baboon → color Jet	212640	41.27	41.30	**41.86**	41.30	41.29
quantized color House → color Jet	119584	47.79	47.94	41.88	**47.95**	**47.95**
quantized color Lena → color Jet	179280	44.52	**44.53**	41.86	**44.53**	**44.53**
quantized color Peppers → color Jet	148400	**45.91**	**45.91**	41.86	**45.91**	**45.91**
quantized color Sailboat → color Jet	171072	44.72	**44.73**	41.87	**44.73**	**44.73**
quantized color Jet → color Baboon	117472	48.16	48.27	41.87	**48.28**	48.26
quantized color Baboon → color Baboon	212640	41.32	41.34	**41.86**	41.34	41.34
quantized color House → color Baboon	119584	47.80	47.93	41.87	**47.96**	47.95
quantized color Lena → color Baboon	156592	45.68	**45.70**	41.87	**45.70**	**45.70**
quantized color Peppers → color Baboon	132560	46.42	**46.43**	41.88	**46.43**	**46.43**
quantized color Sailboat → color Baboon	153472	45.78	**45.79**	41.87	**45.79**	**45.79**
quantized color Jet → color House	110704	48.15	48.26	41.83	**48.28**	48.27
quantized color Baboon → color House	186192	44.37	**44.38**	41.86	**44.38**	**44.38**
quantized color House → color House	119600	47.79	47.94	41.87	**47.96**	47.95
quantized color Lena → color House	156592	45.69	**45.70**	41.88	**45.70**	**45.70**
quantized color Peppers → color House	132560	46.39	**46.41**	41.88	**46.41**	**46.41**
quantized color Sailboat → color House	153472	**45.78**	**45.78**	41.86	**45.78**	**45.78**
quantized color Jet → color Lena	110704	48.13	48.25	41.87	**48.28**	48.27
quantized color Baboon → color Lena	186192	44.37	**44.39**	41.87	**44.39**	**44.39**
quantized color House → color Lena	119600	47.79	47.94	41.88	**47.95**	**47.95**
quantized color Lena → color Lena	156592	45.69	**45.70**	41.89	**45.70**	**45.70**
quantized color Peppers → color Lena	132560	46.41	**46.42**	41.87	**46.42**	**46.42**
quantized color Sailboat → color Lena	153472	45.78	**45.79**	41.87	**45.79**	**45.79**
quantized color Jet → color Peppers	110704	48.15	48.26	41.58	**48.28**	48.27
quantized color Baboon → color Peppers	186192	44.30	**44.31**	41.62	**44.31**	**44.31**
quantized color House → color Peppers	119600	47.81	47.94	41.60	**47.95**	**47.95**
quantized color Lena → color Peppers	156592	45.62	**45.64**	41.62	**45.64**	**45.64**
quantized color Peppers → color Peppers	132560	46.33	**46.37**	41.61	**46.37**	**46.37**
quantized color Sailboat → color Peppers	153472	45.71	**45.73**	41.61	**45.73**	**45.73**
quantized color Jet → color Sailboat	110704	48.14	48.25	41.88	**48.27**	48.26
quantized color Baboon → color Sailboat	186192	44.34	**44.36**	41.87	**44.36**	**44.36**
quantized color House → color Sailboat	119600	47.80	47.93	41.89	**47.95**	47.94
quantized color Lena → color Sailboat	156592	45.66	**45.67**	41.87	**45.67**	**45.67**
quantized color Peppers → color Sailboat	132560	46.38	**46.41**	41.86	**46.41**	**46.41**
quantized color Sailboat → color Sailboat	153472	45.75	**45.77**	41.87	**45.77**	**45.77**
Average quality	**148437**	**45.97**	**46.02**	**41.83**	**46.03**	**46.03**

Table 13 – Quality comparison (PSNR values) with our results and those presented on reference [20]

Secret data → cover image	Method (average time in seconds)			
	LSB2	GEN2	GA+PR	GA+PR'
quantized color Jet → color Jet	**0.126**	25.762	157.225	52.805
zipped quantized color Jet 2x → color Jet	**0.125**	28.136	100.193	28.340
quantized color Baboon → color Jet	**0.125**	27.147	109.683	27.199
quantized color House → color Jet	**0.144**	27.620	182.548	55.480
quantized color Lena → color Jet	**0.125**	26.031	73.676	25.204
quantized color Peppers → color Jet	**0.123**	23.738	63.262	24.951
quantized color Sailboat → color Jet	**0.126**	25.397	78.729	25.083
quantized color Jet → color Baboon	**0.125**	26.040	243.445	61.723
quantized color Baboon → color Baboon	**0.140**	27.192	100.170	27.195
quantized color House → color Baboon	**0.125**	26.979	188.981	51.917
quantized color Lena → color Baboon	**0.125**	25.017	64.841	24.944
quantized color Peppers → color Baboon	**0.078**	15.117	43.165	15.163
quantized color Sailboat → color Baboon	**0.083**	16.836	46.035	16.786
quantized color Jet → color House	**0.083**	16.830	103.392	34.173
quantized color Baboon → color House	**0.088**	18.019	55.057	18.115
quantized color House → color House	**0.088**	17.199	82.111	31.322
quantized color Lena → color House	**0.083**	17.023	44.371	18.024
quantized color Peppers → color House	**0.080**	15.430	35.024	15.942
quantized color Sailboat → color House	**0.082**	16.738	56.227	16.731
quantized color Jet → color Lena	**0.085**	16.715	104.952	33.322
quantized color Baboon → color Lena	**0.087**	17.968	56.011	18.101
quantized color House → color Lena	**0.089**	17.291	91.430	33.250
quantized color Lena → color Lena	**0.082**	16.928	44.909	18.363
Quantized color Peppers → color Lena	**0.078**	15.450	39.999	15.420
quantized color Sailboat → color Lena	**0.083**	17.021	43.706	17.018
quantized color Jet → color Peppers	**0.088**	16.684	110.545	36.122
quantized color Baboon → color Peppers	**0.089**	18.040	56.343	18.042
quantized color House → color Peppers	**0.090**	17.297	117.379	37.663
Quantized color Lena → color Peppers	**0.082**	16.972	47.016	16.676
quantized color Peppers → color Peppers	**0.078**	15.163	40.185	16.645
quantized color Sailboat → color Peppers	**0.078**	16.411	54.078	16.380
quantized color Jet → color Sailboat	**0.094**	16.333	73.227	37.315
quantized color Baboon → color Sailboat	**0.094**	17.737	55.286	17.722
quantized color House → color Sailboat	**0.078**	16.941	116.204	32.572
quantized color Lena → color Sailboat	**0.078**	16.660	52.151	16.848
quantized color Peppers → color Sailboat	**0.062**	15.038	36.410	15.038
quantized color Sailboat → color Sailboat	**0.094**	16.365	35.678	16.317
Average time	**0.097**	**19.548**	**81.180**	**26.592**

Table 14 – Time spent in seconds with our tests

Secret data → cover image	Method (billions of instructions value)			
	LSB2	GEN2	GA+PR	GA+PR'
quantized color Jet → color Jet	**2.19**	448.59	2737.76	919.49
zipped quantized color Jet 2x → color Jet	**2.18**	489.93	1744.66	493.48
quantized color Baboon → color Jet	**2.18**	472.71	1909.91	473.62
quantized color House → color Jet	**2.51**	480.95	3178.71	966.07
quantized color Lena → color Jet	**2.18**	453.28	1282.92	438.88
quantized color Peppers → color Jet	**2.14**	413.35	1101.58	434.47
quantized color Sailboat → color Jet	**2.19**	442.24	1370.91	436.77
quantized color Jet → color Baboon	**2.18**	453.43	4239.11	1074.78
quantized color Baboon → color Baboon	**2.44**	473.49	1744.26	473.55
quantized color House → color Baboon	**2.18**	469.79	3290.73	904.03
Quantized color Lena → color Baboon	**2.18**	435.62	1129.08	434.35
quantized color Peppers → color Baboon	**1.36**	263.23	751.63	264.03
quantized color Sailboat → color Baboon	**1.45**	293.17	801.61	292.29
quantized color Jet → color House	**1.45**	293.06	1800.36	595.05
quantized color Baboon → color House	**1.53**	313.76	958.71	315.44
Quantized color House → color House	**1.53**	299.49	1429.80	545.41
quantized color Lena → color House	**1.45**	296.42	772.63	313.85
quantized color Peppers → color House	**1.39**	268.68	609.87	277.60
quantized color Sailboat → color House	**1.43**	291.46	979.08	291.34
quantized color Jet → color Lena	**1.48**	291.06	1827.53	580.24
Quantized color Baboon → color Lena	**1.51**	312.88	975.32	315.19
quantized color House → color Lena	**1.55**	301.09	1592.07	578.98
quantized color Lena → color Lena	**1.43**	294.77	782.00	319.75
Quantized color Peppers → color Lena	**1.36**	269.03	696.50	268.51
quantized color Sailboat → color Lena	**1.45**	296.39	761.05	296.33
quantized color Jet → color Peppers	**1.53**	290.52	1924.92	628.99
quantized color Baboon → color Peppers	**1.55**	314.13	981.10	314.17
quantized color House → color Peppers	**1.57**	301.19	2043.92	655.83
Quantized color Lena → color Peppers	**1.43**	295.53	818.69	290.38
quantized color Peppers → color Peppers	**1.36**	264.03	699.74	289.84
quantized color Sailboat → color Peppers	**1.36**	285.76	941.66	285.22
quantized color Jet → color Sailboat	**1.64**	284.41	1275.10	649.77
quantized color Baboon → color Sailboat	**1.64**	308.85	962.70	308.59
quantized color House → color Sailboat	**1.36**	294.99	2023.46	567.18
quantized color Lena → color Sailboat	**1.36**	290.10	908.11	293.37
quantized color Peppers → color Sailboat	**1.08**	261.86	634.01	261.86
quantized color Sailboat → color Sailboat	**1.64**	284.96	621.26	284.13
Average billions of instructions	**1.69**	**340.38**	**1413.58**	**463.05**

Table 15 – Time spent in BI units (billions of instructions) with our tests

6.8 Comparison with difference expansion

In this section, our results are compared with the ones presented on section 4.1 from [37], that introduces a lossless data embedding scheme that exploits the difference expansion of the pixels to conceal large amount of message data in a digital image. The tests were done using the 32-bits color versions of the Lena, Baboon and Zelda images presented by [37] as cover images of 512x512 pixels (Figures 50, 51 and 58). The secret data used were text messages extracted from beginning of [47] and saved on unicode format using Windows notepad application, with the same size in bits specified by [37]. The text message size on [47] is around 100 kilobytes, so

it was repeated sometimes until achieve the desired amount of bytes hidden for each test. The other nine images results unfortunately could not be compared with ours because the paper was not clear about the exact amount of bits hidden inside them, referencing only a generic term called payload, relative to the cover image size, which was not shown inside the article. DEX[37] are the results presented by [37] with the difference expansion strategy. LSB2 and GEN2 are the results achieved by our implementation of least significant bits substitution and genetic algorithm, originally proposed by [16]. GA+PR is our new proposal, an implementation of the genetic algorithm with the path relinking approach, applied after the end of each genetic algorithm generation, by combining the 10 better solutions found until then (elite group). The GA+PR process is detailed in the section 6.3. GA+PR' is a modified version of GA+PR, using a simpler and faster path relinking approach, also detailed in the section 6.3.

Quality comparisons are presented on table 16. The arrow symbol (→) was used to indicate that the image before the symbol is being embedded inside the image after the symbol. The time results of our executions are shown on table 17, with cryptography time included for all tests. Time comparisons were not possible because the authors not commented any details about time results, so we will be presenting only ours. Table 18 shows the time results converted to the BI unit values (billions of instructions), for future comparisons. Each result of our implementations is an average value for five executions with the same cover image and secret data.

The path relinking results (GA+PR and GA+PR') presented here were obtained by using the method described in section 4.2.

Secret data → cover image	**Bytes hidden**	**Method (PSNR result)**				
		LSB2	**GEN2**	**DEX[37]**	**GA+PR**	**GA+PR'**
message → color Lena	224552	41.07	**41.09**	34.79	**41.09**	**41.09**
message → color Lena	252059	39.35	39.36	30.52	**39.37**	39.36
message → color Baboon	141238	46.14	**46.15**	32.64	**46.15**	**46.15**
message → color Baboon	231657	39.72	39.74	24.91	**39.75**	39.74
message → color Zelda	255570	39.30	**39.32**	33.31	**39.32**	**39.32**
zipped message 5x → color Zelda	264720	38.38	38.40	-	**38.41**	38.40
Average quality	**221015**	**41.12**	**41.13**	**31.23**	**41.14**	**41.13**

Table 16 – Quality comparison (PSNR values) with our results and those presented on reference [37]

Secret data → cover image	Method (average time in seconds)			
	LSB2	GEN2	GA+PR	GA+PR'
message → color Lena	**0.141**	28.315	109.745	28.299
message → color Lena	**0.140**	29.084	141.930	28.877
message → color Baboon	**0.109**	23.301	62.777	23.640
message → color Baboon	**0.141**	27.143	109.199	27.238
message → color Zelda	**0.140**	29.199	124.698	24.468
zipped message 5x → color Zelda	**0.141**	28.533	119.286	28.517
Average time	**0.134**	**27.408**	**109.670**	**26.504**

Table 17 – Time spent in seconds with our tests

Secret data → cover image	Method (billions of instructions value)			
	LSB2	GEN2	GA+PR	GA+PR'
message → color Lena	**2.46**	493.05	1910.99	492.77
message → color Lena	**2.44**	506.44	2471.43	502.84
message → color Baboon	**1.90**	405.74	1093.14	411.64
message → color Baboon	**2.46**	472.64	1901.48	474.30
message → color Zelda	**2.44**	508.44	2171.37	426.06
zipped message 5x → color Zelda	**2.46**	496.85	2077.13	496.57
Average billions of instructions	**2.36**	**480.53**	**1937.59**	**467.36**

Table 18 – Time spent in BI units (billions of instructions) with our tests

6.9 Comparison with lossless block truncation coding

In this section, our results are compared with the ones presented on Table 2 from [38], that details a reversible hiding method that aims at block truncation coding (BTC) compressed color images. In order to improve the compression rate, a genetic algorithm (GA) is applied. The tests were done using the same six classical images presented by [38] as cover images, which are the 32-bits color, 512x512 pixels images of Jet, Baboon, Lena, Peppers, Tifanny and Sailboat (Figures 50-51, 53, 55, 57 and 59). The secret data used were text messages extracted from beginning of [47] and saved on unicode format using Windows notepad application, with the same size in bits specified by [38]. The text message size on [47] is around 100 kilobytes, so it was capped sometimes until achieve the desired amount of bytes hidden for each test. The other four common bitmaps images presented by [38], called Scene, Pillar, Snow and Plate unfortunately could not be found in public domain inside internet, turning unviable the comparisons with these ones. Time comparison was also not possible because the authors did not specified the computational system used to obtain the time results reported by them. LBT[38] are the results presented by [38] with the lossless block truncation coding strategy. LSB2 and GEN2 are the results achieved by our implementation of least significant bits substitution and genetic algorithm, originally proposed by [16]. GA+PR is our new proposal, an

implementation of the genetic algorithm with the path relinking approach, applied after the end of each genetic algorithm generation, by combining the 10 better solutions found until then (elite group). The GA+PR process is detailed in the section 6.3. GA+PR' is a modified version of GA+PR, using a simpler and faster path relinking approach, also detailed in the section 6.3.

Quality comparisons are presented on table 19. The arrow symbol (→) was used to indicate that the image before the symbol is being embedded inside the image after the symbol. The results reported by [38] have a separate value for red, green and blue color channels. They are presented here as an average value for these three channels. The time results of our executions are shown on table 20, with cryptography time included for all tests. Table 21 shows the time results converted to the BI unit values (billions of instructions), for future comparisons. Each result of our implementations is an average value for five executions with the same cover image and secret data.

All path relinking results (GA+PR and GA+PR') presented here used the section 4.2.1 alternative approach, which leads to better results when secret data is around one third or less of cover image hiding capacity.

Secret data → cover image	**Bytes hidden**	**Method (PSNR result)**				
		LSB2	**GEN2**	**LBT [38]**	**GA+PR**	**GA+PR'**
message → color Jet	58572	**53.41**	**53.41**	31.87	**53.41**	**53.41**
message → color Baboon	58868	**53.39**	**53.39**	24.89	**53.39**	**53.39**
message → color Peepers	60044	53.36	**53.37**	29.49	**53.37**	**53.37**
message → color Lena	59984	53.26	**53.31**	31.97	**53.31**	**53.31**
message → color Tiffany	37170	54.65	**56.23**	29.66	**56.23**	**56.23**
message → color Sailboat	59938	53.31	**53.31**	27.59	**53.31**	**53.31**
Average quality	**55763**	**53.56**	**53.84**	**29.25**	**53.84**	**53.84**

Table 19 – Quality comparison (PSNR values) with our results and those presented on reference [38]

Secret data → cover image	**Method (average time in seconds)**			
	LSB2	**GEN2**	**GA+PR**	**GA+PR'**
message → color Jet	**0.074**	14.150	26.042	19.942
message → Color Baboon	**0.072**	14.166	23.678	17.805
message → Color Peepers	**0.072**	14.181	26.475	18.229
message → color Lena	**0.072**	14.126	25.687	18.418
message → Color Tiffany	**0.066**	12.378	22.549	16.339
message → Color Sailboat	**0.074**	14.220	25.782	18.410
Average Time	**0.072**	**13.870**	**25.036**	**18.191**

Table 20 – Time spent in seconds with our tests

Secret data → cover image	Method (billions of instructions value)			
	LSB2	GEN2	GA+PR	GA+PR'
message → color Jet	**1.29**	246.39	453.47	347.25
message → color Baboon	**1.25**	246.67	412.31	310.04
message → color Peepers	**1.25**	246.93	461.01	317.42
message → color Lena	**1.25**	245.98	447.29	320.71
message → color Tiffany	**1.15**	215.54	392.65	284.51
message → color Sailboat	**1.29**	247.61	448.94	320.57
Average billions of instructions	**1.25**	**241.52**	**435.94**	**316.75**

Table 21 – Time spent in BI units (billions of instructions) with our tests

6.10 Comparison with binary space partitioning tree

In this section, our results are compared with the ones presented on Tables 2, 3 and 4 from [39], which explain a hiding technique for color images using a binary space partitioning (BSP) tree. The tests were done using the same four images presented by [39] as cover images, which are the 32-bits color, 512x512 pixels images of Jet, Baboon, Lena and Peppers (Figures 50-51, 53 and 57). The secret data used were text messages extracted from beginning of [47] and saved on unicode format using Windows notepad application, with the same size in bits specified by [39], always a sequence of 512 x 512 x 3 bits. The text message size on [47] is around 100 kilobytes, so it was capped sometimes until achieve the desired amount of bytes hidden for each test. BSP[39] are the results presented by [39] with the binary space partitioning strategy. LSB2 and GEN2 are the results achieved by our implementation of least significant bits substitution and genetic algorithm, originally proposed by [16]. GA+PR is our new proposal, an implementation of the genetic algorithm with the path relinking approach, applied after the end of each genetic algorithm generation, by combining the 10 better solutions found until then (elite group). The GA+PR process is detailed in the section 6.3. GA+PR' is a modified version of GA+PR, using a simpler and faster path relinking approach, also detailed in the section 6.3.

Quality comparisons are presented on table 22. The arrow symbol (→) was used to indicate that the image before the symbol is being embedded inside the image after the symbol. The word "zipped", appearing before the arrow symbol, indicates the embedding image was submitted to WinZip compression application version 11.1, generating a lossless compressed file to be hidden inside the cover image. Our time results are shown on table 23, with

cryptography time included for all tests. Table 24 shows a comparison of time results converted to the BI unit values (billions of instructions), for future comparisons. According to [39], a Pentium IV 2.4 GHz was used to process the results, which achieves a Drystone benchmark result of 5234 MIPS in [47]. Each result of our implementations is an average value for five executions with the same cover image and secret data.

All path relinking results (GA+PR and GA+PR') presented on Table 22 used the section 4.2.1 alternative approach, which leads to better results when secret data is around one third or less of cover image hiding capacity.

Secret data → cover image	Bytes hidden	Method (PSNR result)				
		LSB2	GEN2	BSP[39]	GA+PR	GA+PR'
message → color Jet	98304	51.16	**51.17**	51.14	**51.17**	**51.17**
zipped message → color Jet	20432	57.95	**57.99**	-	**57.99**	**57.99**
message → color Baboon	98304	51.14	51.17	51.14	**51.18**	**51.18**
zipped message → color Baboon	20432	57.99	57.99	-	57.99	57.99
message → color Peepers	98304	51.24	**51.25**	51.14	**51.25**	**51.25**
zipped message → color Peepers	20432	57.96	**57.98**	-	**57.98**	**57.98**
message → color Lena	98304	51.10	**51.19**	51.14	**51.19**	**51.19**
zipped message → color Lena	20432	57.98	57.98	-	57.98	57.98
Average quality	**98336**	**54.57**	**54.59**	**51.14**	**54.59**	**54.59**

Table 22 – Quality comparison (PSNR values) with our results and those presented on reference [39]

Secret data → cover image	Method (average time in seconds)			
	LSB2	GEN2	GA+PR	GA+PR'
message → color Jet	**0.082**	16.773	43.218	24.430
zipped message → color Jet	**0.060**	11.727	13.576	12.703
message → color Baboon	**0.085**	16.289	54.540	25.762
zipped message → color Baboon	**0.060**	11.667	14.378	12.530
message → color Peepers	**0.086**	16.375	44.345	24.003
zipped message → color Peepers	**0.064**	11.723	13.066	12.681
message → color Lena	**0.083**	16.082	52.697	25.342
zipped message → color Lena	**0.061**	11.766	13.954	12.676
Average time	**0.073**	**14.050**	**31.222**	**18.766**

Table 23 – Time comparisons in seconds between our results and those presented on reference [39]

Secret data → cover image	Method (billions of instructions value)				
	LSB2	GEN2	BSP[39]	GA+PR	GA+PR'
message → color Jet	**1.43**	292.07	4.33	752.56	425.40
zipped message → color Jet	**1.04**	204.20	-	236.40	221.20
message → color Baboon	**1.48**	283.64	7.77	949.71	448.59
zipped message → color Baboon	**1.04**	203.16	-	250.36	218.18
message → color Peepers	**1.50**	285.14	3.68	772.18	417.96
zipped message → color Peepers	**1.11**	204.13	-	227.52	220.81
message → color Lena	**1.45**	280.04	5.56	917.61	441.28
zipped message → color Lena	**1.06**	204.88	-	242.98	220.73
Average billions of instructions	**1.26**	**244.66**	**5.34**	**543.66**	**326.77**

Table 24 – Time spent in BI units (billions of instructions) with our tests

6.11 Steganography of an executable file

In this section a different steganography act was tried: The hiding of an executable file inside the cover image. The secret data, in that case, was a 3D version of a very popular game, the famous Pacman, an executable file named PacMan3D.exe, obtained from [48]. The cover image used was Figure 60, Pacman3D, 32-bits color with 640x480 pixels, obtained from [46].

Quality results are presented on table 25. The arrow symbol (→) was used to indicate that the image before the symbol is being embedded inside the image after the symbol. The word "zipped", appearing before the arrow symbol, indicates the embedding image was submitted to WinZip compression application version 11.1, generating a lossless compressed file to be hidden inside the cover image. The word "5x", appearing before the arrow symbol, indicates five identical executable files, instead of only one, were compressed by the WinZip application version 11.1, which also produces a lossless compressed file, of greater size. The time results are shown on table 26, with cryptography time included for all tests. Table 27 shows the time results converted to the BI unit values (billions of instructions), for future comparisons. Each result of our implementations is an average value for five executions with the same cover image and secret data.

The path relinking results (GA+PR and GA+PR') presented here used the section 4.2.1 alternative approach, which leads to better results when secret data is around one third or less of cover image hiding capacity.

Secret data → cover image	Bytes hidden	Method (PSNR result)			
		LSB2	GEN2	GA+PR	GA+PR'
Pacman3D game → Pacman3D	94224	51.99	**52.04**	**52.04**	**52.04**
zipped Pacman3D game 5x → Pacman3D	213424	44.47	**44.49**	**44.49**	**44.49**

Table 25 – Quality results (PSNR values), after hiding the executable file Pacman3D.exe inside Figure 60

Secret data → cover image	Method (average time in seconds)			
	LSB2	GEN2	GA+PR	GA+PR'
Pacman3D game → Pacman3D	**0.080**	15.190	27.664	15.099
zipped Pacman3D game 5x → Pacman3D	**0.141**	30.454	83.004	30.470

Table 26 – Time results in seconds, after hiding the executable file Pacman3D.exe inside Figure 60

Secret data → cover image	Method (billions of instructions value)			
	LSB2	GEN2	GA+PR	GA+PR'
Pacman3D game → Pacman3D	**1.39**	264.50	481.71	262.92
zipped Pacman3D game 5x → Pacman3D	**2.46**	530.30	1445.35	530.57

Table 27 – Time spent in BI units (billions of instructions) with our tests

7. Conclusions and future works

After having concluded the comparisons among this work and many others analyzed, we achieved a good degree of satisfaction. This work is the most flexible of them, being a complete solution able to hide anything of considerable size inside the cover image with good quality results.

An executable file, a zip file, an excel spreadsheet, a powerpoint presentation, a mp3 music, … any of these and any other one can be hidden safely inside a color image. A custom stego key is provided to protect the hidden information by encrypting it with a reliable and high security cryptography algorithm. After the encryption, the information is submitted to an optimization path relinking routine to improve the final quality of the hiding process, turning the information invisible to the human eyes. At the end, the image containing the hidden information can be sent via electronic mail, cell phone or any other means to its destiny, without worry. Afterwards, the receptor of the message activates the image decryption routine, providing the stego key to quickly recover the critical information contained inside the image. That is good.

7.1 Conclusions

In this work, the use of path relinking as an improvement over the existing method has been shown itself relatively expensive when we look at the additional time elapsed in execution compared to the increase on final image quality. Nevertheless, in an image hiding process, the main objective is the concealment of the information, making it “invisible” to intruders, so the quality gain is the main focus not only in this work.

The change from grayscale to color images on steganography process shows a great difference on final image quality and expands the embedding capacity to a factor of three or more times when compared to the original.

The addition of the AES cryptography algorithm incorporates a new security level over the hidden information, opening paths for several futures uses of this application on the World

Wide Web. The encryption and decryption execution times are also very cheap when compared to the total time spent on steganography.

Several other configurations of the genetic algorithm implementation combined with the path relinking refinement were also tried. For example, executing the path relinking refinement only after the end of all genetic algorithm processing produced faster hiding but worse quality results. Increasing the maximum number of generations in genetic algorithm to ten, instead eight, resulted in a slower processing time, but showed noticeable enhancements on final results. When doubling the population size at each generation of genetic algorithm, the processing achieved slowest time but slightly better quality results. Applying the reversal path relinking after the standard path relinking resulted in a two times slower processing but few noticeable quality gains. The best configuration was obtained by executing the genetic algorithm with a total of eight generations and applying only the standard path relinking refinement at the end of each genetic algorithm generation, over the best individuals chosen to form the next generation population, with a limit of forty individuals generated after the path relinking process.

Considering all the improvements, after the tests, although still somewhat experimental, the method shows itself more effective when discovering better solutions in the middle of the best ones already existing, especially when the search space and the number of possibilities are not too vast, presenting best time and quality performances when the hidden data occupies one third of its maximum capacity. An interesting aspect of this method is its flexibility, allowing its combination with several other nice approaches. In fact, the final quality of solutions obtained through path relinking refinement is strongly influenced by the main method capacity to generate several diverse and good solutions.

Another interesting aspect of this research is the development of a machine independent time comparison unit, the BI, or billions of instructions value, in an expectance to allow reasonable comparisons among diverse computers systems in the future.

7.2 Future works

When working toward a good, or an objective, new ideas always arise. It could be compared to a journey to a great horizon. The more steps you walk, more you see and discover that needs to be completed. Here are some ideas that came up on the developing of this work:

1. Replace the genetic algorithm by another meta-heuristic, like GRASP or Tabu Search, which may perform better;

2. Use other less costly local search methods instead of path relinking, such as Variable Neighborhood Search (VNS) or Iterated Local Search, among others;

3. Create different time performance measures, possibly based on Standard Performance Evaluation Corporation (SpecInt) benchmark or another one, in order to achieve better precision of results for future comparisons;

4. Develop a similar color image steganography application optimized for use with cell phones or on the Internet. It could also incorporate some video steganography features, for use within multimedia messages.

References

[1] Jinn-Ke Jan and Yuh-Min Tseng , On the security of image encryption method, *Information Processing Letters* 60 (1996) 261-265.

[2] N. G. Bourbakis and C. Alexopoulos, Picture data encryption using scan patterns, *Pattern Recognition* 25 (1992) 567-581.

[3] H. J. Highland, Data encryption: a non-mathematical approach, *Computers and Security* 16 (1997) 369-386.

[4] Man Young Rhee, Cryptography and secure communication (1994) ISBN 0071125027, *McGraw-Hill, Inc-New York.*

[5] W. Bender, D. Gruhl, N. Morimoto and A. Lu, Techniques for data hiding, *IBM Systems Journal* 35 (1996) 313-336.

[6] Z. Duric, M. Jacobs and S. Jajoolia, Information hiding: steganography and steganalysis, *Handbook of Statistics* 24 (2005) 171-187.

[7] F. Glover, M. Laguna and R. Martí, Fundamentals of scatter search and path relinking, *Control and Cybernetics* 29 (2000) 653-684.

[8] F. Glover, A Template for scatter search and path relinking in *Lecture Notes in Computer Science* 1363 (1998) 1-51 ISBN 978-3-540-64169-8, Springer-Berlin/Heidelberg.

[9] G. C. Onwubolu and B. V. Babu, Scatter search and path relinking: foundations and advanced designs in New Optimization Techniques in Engineering (2004) 87-99 ISBN 354020167X, *Springer-Verlag*.

[10] M. Laguna, Scatter search and path relinking: methodology and applications. Available at: http://leeds-faculty.colorado.edu/laguna/ presentations/puebla.ppt.

[11] B. Norman, Secret warfare, the battle of codes and ciphers (1980) ISBN 0-87491-600-3, *Acropolis Books*- Washington D.C.

[12] A. Rocha and S. Goldenstein, Steganography and steganalysis in digital multimedia: hype or hallelujah?, *Revista de Informática Teórica e Aplicada* (2008).

[13] Chih-Ching Thien and Ja-Chen Lin, A simple and high-hiding capacity method for hiding digit-by-digit data in images based on modulus function, *Pattern Recognition* 36 (2003) 2875-2881.

[14] Chin-Chen Chang, Ju-Yuan Hsiao and Chi-Shiang Chan, Finding optimal least-significant-bit substitution in image hiding by dynamic programming strategy, *Pattern Recognition* 36 (2003) 1583-1595.

[15] Yu-Chen Hu, High-capacity image hiding scheme based on vector quantization, *Pattern Recognition* 39 (2006) 1715-1724.

[16] Ran-Zan Wang, Chi-Fang Lin, Ja-Chen Lin, Image hiding by optimal LSB substitution and genetic algorithm, *Pattern Recognition* 34 (2001) 671-683.

[17] Shiuh-Jeng Wang, Steganography of capacity required using modulo operator for embedding secret image, *Applied Mathematics and Computation* 164 (2005) 99-116.

[18] Chi-Yuan Lina and Chin-Hsing Chenb, An invisible hybrid color image system using spread vector quantization neural networks with penalized FCM, *Pattern Recognition* 40 (2007) 1685-1694.

[19] Yu-Chen Hu, Min-Hui Lin and Ji-Han Jiang, A Novel Color Image Hiding Scheme Using block truncation coding, *Fundamenta Informaticae* 70 (2006) 317–331.

[20] Yuan-Hui Yu, Chin-Chen Chang and Iuon-Chang Lin , A new steganographic method for color and grayscale image hiding, *Computer Vision and Image Understanding* 107 (2007) 183-194.

[21] Wen-Yuan Chen, Color image steganography scheme using DFT, SPIHT codec, and modified differential phase-shift keying techniques, *Applied Mathematics and Computation*, 196 (2008) 40-54.

[22] Ran-Zan Wang and Yao-De Tsai , An image-hiding method with high hiding capacity based on best-block matching and k-means clustering, *Pattern Recognition* 40 (2007) 398-409.

[23] KokSheik Wong, Xiaojun Qi and Kiyoshi Tanaka, A DCT-based mod4 steganographic method, *Signal Processing* 87 (2007) 1251-1263.

[24] P. Campisi, D. Kundur, D. Hatzinakos and A. Neri, Compressive data hiding: an unconventional approach for improved color image coding, *EURASIP Journal on Applied Signal Processing* 2 (2002) 152–163.

[25] Wen-Yuan Chen, Color image steganography scheme using set partitioning in hierarchical trees coding, digital fourier transform and adaptive phase modulation, *Applied Mathematics and Computation* 185 (2007) 432-448.

[26] M. Ashourian, P. Moallem and Yo-Sung Ho, A robust method for data hiding in color images, *Lecture Notes in Computer Science* 3768 (2005) 258-269 ISBN 3-540-30027-9, Springer-Berlin.

[27] L. Pitsoulis and M.G.C. Resende, Greedy randomized adaptive search procedures, *Handbook of Applied Optimization* (2002) 168-181 ISBN 0195125940, Oxford University Press.

[28] F. Glover, Tabu search - Part I, *ORSA Journal on Computing* 1 (1989) 190-206.

[29] F. Glover, Tabu search - Part II, *ORSA Journal on Computing* 2 (1990) 4-32.

[30] Announcing the advanced encryption standard (AES), *Federal Information Processing Standards Publication* 197 (2001). Available at: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[31] J. Daemen and V. Rijmen, AES proposal: Rijndael, *AES Algorithm Submission* (1999). Available at: http://www.gel.ulaval.ca/~klein/maitrise/aes/rijndael.pdf.

[32] Advanced encryption standard, *Wikipedia, the free encyclopedia.* Available at: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

[33] N. Mladenovic, and P. Hansen, Variable neighbourhood search. *Computers and Operations Research* 24 (1997) 1097-1100.

[34] N. Mladenovic, and P. Hansen, Variable neighbourhood search: Principles and Applications, *European Journal of Operational Research* 130 (2001) 449-467.

[35] Y. Rochat, É. D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, *Journal of Heuristics* 1 (1995) 147-167. Available at: http://ina2.eivd.ch/collaborateurs/etd/articles.dir/crt95_13.pdf

[36] C. R. Reeves, Genetic algorithms, path relinking and the flow shop sequencing problem, *Evolutionary Computation Journal*, 6 (1998) 230-234.

[37] Chin-Chen Chang and Tzu-Chuen Lu, A difference expansion oriented data hiding scheme for restoring the original host images, *The Journal of Systems and Software* 79 (2006) 1754-1766.

[38] Chin-Chen Changa, Chih-Yang Linb and Yi-Hsuan Fanb, Lossless data hiding for color images based on block truncation coding, *Pattern Recognition*, in press, corrected proof. Available at www.sciencedirect.com.

[39] Yuan-Yu Tsai and Chung-Ming Wang, A novel data hiding scheme for color images using a BSP tree, *The Journal of Systems and Software* 80 (2007) 429–437.

[40] Mei-Yi Wu, Yu-Kun Ho and Jia-Hong Lee, An iterative method of palette-based image steganography, *Pattern Recognition Letters* 25 (2004) 301–309.

[41] R. P. Weicker, Dhrystone: a synthetic systems programming benchmark, *Communications of the Association for Computing Machinery* 27 (1984) 1013-1030.

[42] System Analyser, Diagnostic and Reporting Assistant (SANDRA), *SiSoftware Corporation*, available at http://www.sisoftware.co.uk/.

[43] WinZip - The compression utility for windows, *Corel Corporation*, available at www.winzip.com.

[44] Image database – volume 3: miscellaneous, *Signal and Image Processing Institute of University of Southern California (USC).* Available at: http://sipi.usc.edu/database/database.cgi?volume=misc

[45] Programming, video codecs and image processing resources. Available at: http://www.hlevkin.com/

[46] The Trustees of Princeton University, Pacman simulator - an automantic synthesized simulator for cycle-accurate multiprocessor simulation. Available at: http://www.ece.neu.edu/~xzhu/pacman.html

[47] N. Provos and P. Honeyman, Hide and seek: an introduction to steganography in *IEEE Security and Privacy* 3 (2003) 32-44. Available at: http://www.citi.umich.edu/u/provos/papers/practical.pdf.

[48] Brian Postma, Pacman 3D game. Available at: http://brianpostma.tweakdsl.nl/files/PacMan3D.zip.