URECA Research Paper.docx

by SPMS LE YANZHI

Submission date: 04-Jun-2024 10:57AM (UTC+0800)
Submission ID: 2390730391
File name: URECA_Research_Paper.docx (560.03K)
Word count: 4616
Character count: 21382

Incorporating Gale-Berlekamp Switch Game in Image Steganography

Le Yanzhi
School of Physical and Mathematical Sciences

Asst Prof Kiah Han Mao
School of Physical and Mathematical Sciences

Abstract **-** Image steganography is the process of hiding information within a digital image. One technique is LSB Substitution, where the least significant bit (LSB) of each pixel value is altered such that it matches the parity of the bit sequence representing the information to hide.

To enhance the reliability of LSB Substitution, we propose the utilization of Gale-Berlekamp Switch Game to minimize the number of changes to the LSBs. In this game, a sequence of binaries is arranged in a square matrix, and each row and column are controlled by a switch that toggles the parity of bits across the corresponding row or column. The aim is to find the combination of toggled switches that result in minimal '1's within the matrix.

In the context of image steganography, bit sequence representing the information to be hidden is first XORed with the LSBs of the image, to identify mismatches in the form of '1's. Gale-Berlekamp Switch Game is subsequently applied to reduce the number of '1's, before embedding the hidden information into the image. Overall results show promise, images with information hidden in this way had lesser alternations to the LSBs, thus offering improved security than the original counterpart, without compromising the integrity of the hidden information. However, the high complexity of finding an optimal solution to the Gale-Berlekamp Switch Game, an integer linear programming problem, particularly for larger datasets, indicates a trade-off between steganographic security and computational efficiency.

Keywords – Image Steganography; LSB Substitution; Gale-Berlekamp Switching Game; Integer Programming; Optimization.

1 INTRODUCTION

1.1 IMAGE STEGANOGRAPHY

A digital image is an image comprised of pixels, arranged in a two-dimensional matrix structure. For example, a digital image with resolution of 640×480 would mean that the digital image has a width of 640 pixels, and a height of 480 pixels. For simplicity, the term "image" will be used to refer to all digital images hereafter unless otherwise specified.

A pixel is a small square of colour, where the colour is represented by numbers depending on the type of colour mode used. If the image is in greyscale, or commonly known as black and white, each pixel is represented by a single integer between 0 and 255. A value of 0 would mean that the pixel is white, and 255 means black. Any value in between represents the varying shade of grey, with a value closer to 0 being darker and a value closer to 255 being lighter. If the image is in colour, one common colour mode is RGB, where each pixel is represented by 3 integers, each between 0 and 255, in a 3-tuple (R, G, B). The value of 'R' represents the intensity of red colour, with higher value representing higher intensity. Similarly, the value of 'G' and 'B' represents the intensity of green and blue colour respectively. The combination of these 3 colours can create most colours that can be perceived by the human eyes [1].

Each number has 256 possible values because both greyscale and RGB usually allocate 8 bits for each number, and $2^8 = 256$.

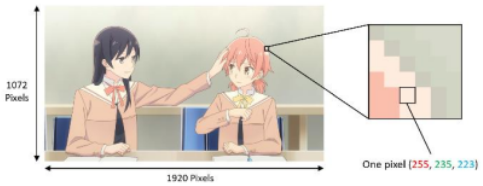

Figure 1. An image in RGB colour mode with width of 1920 pixels and height of 1072 pixels. The zoomed in pixel has a red intensity of 255, green intensity of 235, and blue intensity of 223. [2]

Image steganography is the art of concealing information within images. The hidden information is usually a string, which can be represented in a binary sequence. Many techniques are available. We shall focus on the LSB substitution technique in this paper.

In LSB substitution, we would alter the least significant bit (LSB) of each pixel value such that it matches the parity of the bits in the string. A single unit change in the pixel values is unnoticeable by human eyes [3], but these changes can be detected by computer programmes with access to both the original image (a.k.a. Cover Image) and the modified image (a.k.a. Stego Image). Machine

Learning techniques can also be utilized to detect the presence of hidden information in images [4].

To illustrate, take for example a pixel from a greyscale image, which holds the value 148, equivalently 0x94 in hexadecimal or 10010100_2 in binary. To hide a bit value of '1' into this pixel, we will alter the LSB, or the rightmost bit of value 10010100_2 such that it changes from '0' to '1'. If we wish to hide a bit value of '0' instead, no changes have to be made as the LSB of this pixel value is already '0'.

To achieve this, two techniques are commonly used, LSB Replacement (LSBR) and LSB Matching (LSBM). In LSBR, the LSB is replaced with the corresponding bit value to hide. In the case of pixel value 10010100_2 with bit to hide '1', the result would be 10010101_2. In LSBM, if the parity value does not match, we will randomly add or subtract 1 (50% chance each) from the pixel value. In the case of pixel value 10010100_2 with bit to hide 1, the result would be either 10010101_2 or 10010011_2, depending on whether it was an addition or subtraction respectively. Contrary to popular belief, LSBM is harder to be detected than LSBR, as LSBR introduces statistical anomaly that can be exploited by detection tools [5].

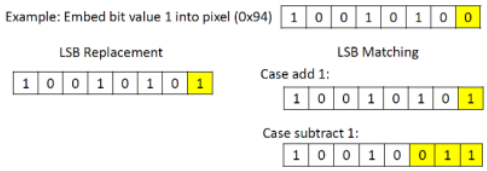

Figure 2. Embedding of bit value 1 to pixel 0x94, using LSBR and LSBM.

1.2 GALE-BERLEKAMP SWITCH GAME

The Gale-Berlekamp Switch Game, proposed independently by Elwyn Berlekamp and David Gale, is a zero-sum mathematical game usually played by 2 players. Lightbulbs are arranged in a matrix grid of size (m × n). A switch is placed on the end of every row and column (i.e. m + n switches in total), which controls the on-off state of the lightbulbs for the corresponding row or column. Initially, all switches are turned off. Toggling a switch will change the state of the light bulbs of the corresponding row or column (off → on or on → off). In the game, the first player will choose some of the lightbulbs to be turned on (independent of the switches; the switches are still toggled off). Afterwards, the second player will use the switches to keep as many lightbulbs as possible off. Finding the optimal choice for the second player is an NP-hard problem [6].

An example of Gale-Berlekamp Switch Game is shown in Figure 3. At the start, all the switches and light bulbs are turned off (1), denoted by '0'. Then, player 1 will choose some light bulbs to be turned on (2), denoted by '1'. Lastly, player 2's goal is to use the switches to turn off as many light bulbs as possible (3). By turning on the first column-switch (on the top from left to right) and the second row-switch (on the left from top to bottom), player 2 is able to reduce the number of lit bulbs from 5 to 1.

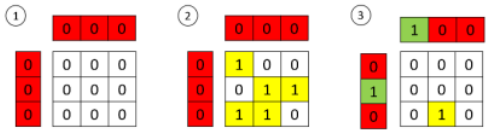

Figure 3. An example of Gale-Berlekamp Switch Game played by two players.

2 AIMS AND OBJECTIVES

In LSB substitution, regardless of whichever technique is used (e.g. LSBR, LSBM), the higher the number of LSB changed to the cover image, the easier it is for the hidden information to be detected by detection tools [7].

With regards to this information, this research proposes the integration of Gale-Berlekamp Switch Game in image steganography. This approach strategically minimizes the number of alterations required to the LSBs during LSB substitution, thereby enhancing the security of the hidden information, without compromising its integrity.

3 METHODOLOGIES

3.1 PREPARATIONS

We prepare a Cover Image, which is the image we wish to embed information in, as well as the information we wish to embed (i.e. hide). We will convert the Cover Image to a LSB Matrix, where each element in the LSB Matrix corresponds to the LSB of the pixel value in the Cover Image. For simplicity, we will assume that the Cover Image is in greyscale, that is, each pixel only holds a single 8-bit value, and thus only one LSB for each pixel. A Cover Image of size (w × h) will produce a LSB Matrix of equal size (w × h). In the case of RGB image, where each pixel holds three 8-bit values, each pixel will have three LSB. A Cover Image of size (w × h) will produce a LSB Matrix of size (w × h × 3). To mitigate this, various possible ways are available, for example, simply choosing one of the three primary colours to embed, thus reducing the LSB Matrix to a 2-dimensional size of (w × h), among other ways which will not be discussed in this paper.

We will also convert the information to a binary sequence of length m and arrange it in a square matrix shape of size (n × n), where $(n - 1)^2 \leq m \leq n^2$. A square grid is used as it simplifies the computational complexity and traceability of Gale-Berlekamp Switch Game, making the solutions easier to generalize [8]. Let us define this matrix as IM, for "Information Matrix".

Before embedding the information into the cover image, the 2 parties, sender and receiver of the Stego Image (i.e. modified image with embedded information), must first agree on a location in the cover image to embed information in. This is so that the receiver will know where in the Stego Image to extract the information from.

Suppose the top left-hand corner of the Cover Image is chosen. To embed IM of size n^2, we require a size of $(n + 1)^2$ from the LSB Matrix. Let us define this section of LSB Matrix, from row 0 to row n, and column 0 to column n, as LM.

The extra row and column are needed because we will assign row 0 and column 0 as the column and row switches respectively. Each element in row 0 will act as switches that control the on-off state of lightbulbs in the column below. In other word, element (0 , j) in LM will act as the column switch that controls the parity of the elements from (1 , j) to (n , j). Similarly, each element in column 0 will act as the switches that control the on-off state of the lightbulbs in the row on its right. Thus, the square space in LM from row 1 to row n, and column 1 to row n (which corresponds to a size of n^2), will be the location to embed IM. Let us define this section of LM as LM*.

To illustrate with an example, we will use a greyscale image as cover, and we wish to embed character 'Y' (ASCII 89 or 01011001_2) into the top left-hand corner of the cover image.

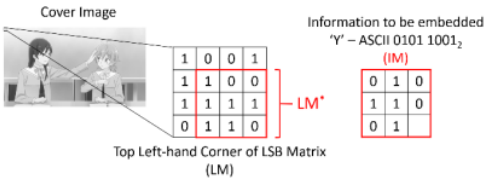

Figure 4. Starting Cover Image and information to be embedded. LM, LM*, and IM are defined here.

3.2 IDENTIFY MISMATCH

We will perform Bitwise Exclusive OR (XOR) operation between LM* and IM. With two inputs, output of XOR returns true if and only if the two inputs are different.

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Table 1. Truth Table of Exclusive OR

Let us define the resulting matrix of LM* ⊕ IM as M*, and the resulting LM as M. That is, M is an ((n+1) × (n+1)) matrix, and M* is the section of M from row 1 to row n, and column 1 to column n.

In M*, a value of '0' would suggest that the parity of the LSB and the bit to embed are the same, and thus, no changes have to be made to the corresponding pixel value in the Cover Image during the embedding of this information bit. On the other hand, a value of '1' would suggest that the parity of the LSB and the bit to embed are different, and the corresponding pixel value in the Cover Image will be changed during the embedding of this information bit. Our goal here is to reduce the number of '1's in M* after XOR, thus reducing the number of changes made to the Cover Image.

Back to the example, we have previously assigned the first row in LM as the column switches, and the first column as the row switches. The element (0,0) in LM is unused. Note that all switches are initially turned off, regardless of whether its initial value is '0' or '1'. We will bitwise XOR LM* (Section of LM from row 1 to 3, and column 1 to 3) and IM, both boxed in red. The mismatches are identified as '1' within M*.

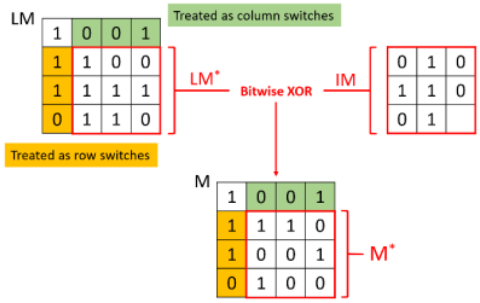

Figure 5. Bitwise XOR operation performed between LM* of LM and IM, resulting in M* in M.

3.3 APPLYING GALE-BERLEKAMP SWITCH GAME

Recall that we wish to minimize the number of changes made to the pixels in the Cover Image during embedding. This translates to two

constraints. First, we will have to minimize the number of '1's in M^*, which can be think of the lit lightbulbs in a Gale-Berlekamp Switch Game. Second, we will also have to minimize the number of changes made to the switches (i.e. row 0 and column 0 in M). In other words, we will be penalized for the number of switches toggled when we are minimizing the number of '1's in M^*. This is one extra constraint not found in the original Gale-Berlekamp Switch Game. In the original Gale-Berlekamp Switch Game, the only aim is to minimize the number of lit lightbulbs, with no penalties on the number of switches toggled.

We can craft out the following objective function regarding the optimization problem.

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\left(M_{ij}\cdot\left(1-H_{ij}\right)+\left(1-M_{ij}\right)\cdot H_{ij}\right)+\sum_{i=1}^{n}ALT\_R_i+\sum_{j=1}^{n}ALT\_C_j$$

Equation 1

In Equation 1,

- M_{ij} denotes the element at row i and column j of M.
- H_{ij} denotes whether the parity of the element at row i and column j of M has changed, where

$$H_{ij}=\left|X_i-Y_j\right|$$

Equation 1.1

  - H_{ij} = 1 if the parity of M_{ij} has changed, 0 otherwise.
- X_i denotes the new parity of row i switch (i.e. new parity of M_{i0}).
  - X_i = 1 if switch is turned on, 0 otherwise.
- Y_j denotes the new parity of column j switch (i.e. new parity of M_{0j}).
  - Y_j = 1 if switch is turned on, 0 otherwise.
- ALT_R_i denotes whether the parity of the row i switch (i.e. parity of M_{i0}) has changed, where

$$ALT\_R_i=(R_i+X_i)\ \%\ 2$$

Equation 1.2

  - ALT_R_i = 1 if the parity of M_{i0} has changed, 0 otherwise.
- ALT_C_j denotes whether the parity of the column j switch (i.e. parity of M_{0j}) has changed, where

$$ALT\_C_j=\left(C_j+Y_j\right)\ \%\ 2$$

Equation 1.3

  - ALT_C_j = 1 if the parity of M_{0j} has changed, 0 otherwise.
- R_i denotes the original parity of row i switch (i.e. original parity of M_{i0}).
  - Switch is originally turned off, regardless of whether R_i = 0 or R_i = 1.
- C_j denotes the original parity of column j switch (i.e. original parity of M_{0j}).
  - Switch is originally turned off, regardless of whether C_j = 0 or C_j = 1.

From Equation 1, we can observe that:

- With respect to Equation 1.1, state of lightbulb at row i and column j will change (i.e. H_{ij} = 1) if and only if either row i switch is turned on (X_i = 1, Y_j = 0) or column j switch is turned on (Y_j = 1, X_i = 0).

  State of lightbulb at row i and column j will not change (i.e. H_{ij} = 0) if and only if both row i and column j switches are turned on (X_i = 1, Y_j = 1), or both row i and column j switches are turned off (X_i = 0, Y_j = 0).
- For lightbulb at row i and column j (M_{ij}), if its state has changed (H_{ij} = 1), we will record the new state of lightbulb (1 – M_{ij}). Else, if its state did not change (H_{ij} = 0), we will record the original state of lightbulb (M_{ij}). This is reflected in

$$\left(M_{ij}\cdot\left(1-H_{ij}\right)+\left(1-M_{ij}\right)\cdot H_{ij}\right)$$

Equation 1 (Snippet)

- As a result, when we minimize

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\left(M_{ij}\cdot\left(1-H_{ij}\right)+\left(1-M_{ij}\right)\cdot H_{ij}\right)$$

Equation 1 (Snippet)

  We are minimizing the number of lit lightbulbs (i.e. the number of '1's in M^*, which is the section of M from row i = 1 to n, and column j = 1 to n).
- When we minimize

$$\sum_{i=1}^{n}ALT\_R_i+\sum_{j=1}^{n}ALT\_C_j$$

Equation 1 (Snippet)

  We are minimizing the number of changes made to the switches when we are playing Gale-Berlekamp Game to minimise the number of lit lightbulbs.

Since Equation 1.1 is not linear, we will convert it to the following linear equations and add them as constraints.

$$H_{ij} \leq X_i - Y_j + 2Z_{ij}$$
$$H_{ij} \geq X_i - Y_j - 2Z_{ij}$$
$$H_{ij} \geq Y_j - X_i$$
$$X_i - Y_j + 1 \leq 2(1 - Z_{ij})$$

Equation 2

Where Z_{ij} is a dummy Boolean variable.

- When $Z_{ij} = 0$, $H_{ij} \leq X_i - Y_j$ and $H_{ij} \geq X_i - Y_j$

  $\therefore H_{ij} = X_i - Y_j = |X_i - Y_j|$

- When $Z_{ij} = 1$, $X_i - Y_j \leq -1$

  $\therefore X_i = 0$ and $Y_j = 1$

  $H_{ij} = Y_j - X_i = |X_i - Y_j|$

Since Equation 1.2 is not linear, we will convert it to the following linear equations and add them as constraints.

$$R_i + X_i = 2T_i + ALT\_R_i$$

Equation 3

Where T_i is a dummy Boolean variable.

Since Equation 1.3 is not linear, we will convert it to the following linear equations and add them as constraints.

$$C_j + Y_j = 2S_j + ALT\_C_j$$

Equation 4

Where S_j is a dummy Boolean variable.

As a result, we have 8 binary decision variables.

- H being an n × n matrix.
- X being an n vector.
- Y being an n vector.
- Z being an n × n matrix.
- ALT_R being an n vector.
- ALT_C being an n vector.
- T being an n vector.
- S being an n vector.

With the objective function Equation 1, and constraints Equation 2, 3, 4, we can create a program to solve the optimization problem. Let us define the optimized M as OM, and the optimized M^* within M as OM^*.

Illustrated by our example, we will minimize the number of '1's within M^*, boxed in red, as well as the number of changes made to the switches highlighted in green and orange. Note that in M, the switches are all turned off, regardless of whether its value is '0' or '1'. In OM, however, the value of '0' means that the switch is turned off, while the value of '1' means that the switch is turned on. This is important during embedding and extraction of information. For both M^* and OM^*, the value of '0' means the lightbulb is turned off, and '1' means the lightbulb is turned on.

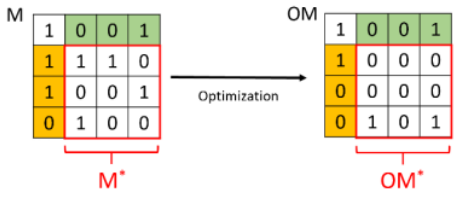

Figure 6. We are able to reduce the number of '1's within the red square box from 4 to 2, by only toggling one switch (second row switch from top).

3.4 EMBEDDING

We will use OM to decide how the LSB matrix will be altered during embedding. The remaining '1's in OM^*, as well as the changes we made to the switches (row 0 and column 0), are the bit changes to the cover image.

This is perhaps better illustrated with the example.

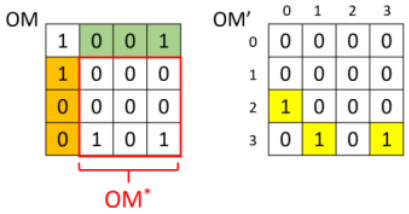

Figure 7. Computation of OM' from OM.

In Figure 7, OM' shows the positions of the pixels that need to have their LSBs altered during embedding, represented by '1's. The '1's at (3,1) and (3,3) in OM' are because they are '1's in OM^*. The '1' at (2,0) in OM' is because there is a change to parity of the switch at element (2,0) in M during optimization (from '1' in M to '0' in OM).

With OM', we can finally embed our information, IM, into the upper left corner LSB Matrix, LM. We will alter the parity of the element (i , j) in LM if and only if the corresponding element (i , j) in OM' is '1'. A '1' in element (i , j) in LM will be altered to '0' in LM', and vice versa.

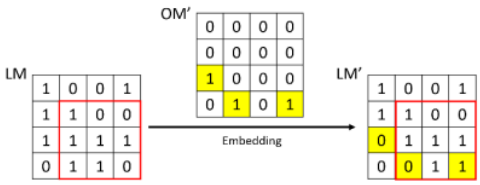

Figure 8. Embedding of information into LM to form LM'.

Finally, we can use LSBR, LSBM, or any other steganography techniques to alter the pixel value based on LM', which indicates the final LSB of each pixel value.

For comparison, we will repeat the same steganography process, without incorporating Gale-Berlekamp Switch Game.

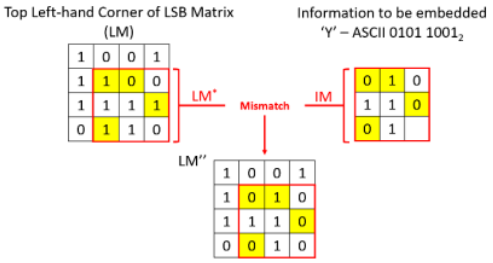

Figure 9. Embedding of information into LM to form LM''.

In Figure 9, for consistency, we will also embed IM into LM* to form LM''. Embedding is done by altering LM* such that the parity matches IM (i.e. parity of LSB of each pixel value matches that of the information to be embedded).

Notice that without optimization, four LSBs are changed in LM'', whereas in LM', with optimization, only three LSBs are changed. Thus, this example illustrates that the Gale-Berlekamp Switch Game approach is able to reduce the number of LSB changes by one.

3.5 EXTRACTION OF INFORMATION

On the receiver end, one may extract out the hidden information from the Stego Image by first converting it to LSB Matrix and extracting out the top left-hand corner. This section of LSB Matrix corresponds to LM', which is the matrix we used for embedding.

To extract out the information from LM', we shall turn off all switches in LM' (row 0 and column 0). Recall that '1' means on and '0' means off. Turning off the switches will also toggle the parity of the lightbulbs in the corresponding row or column.

In the example, we will turn off the first row-switch (i.e. element (1,0)) and the third column-switch (i.e. element (0,3)), and toggles the parity of the lightbulbs in row 1 and column 3. The result is 01011001_2 in the section we have embedded IM, which is the hidden information 'Y' (Recall that the last element (3,3) is unused).

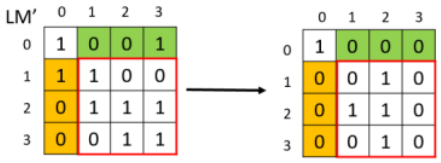

Figure 10. By turning off first row switch and third column switch, we can extract the information within the red square box.

4 FINDINGS

To implement the methodology presented in section 3 of this paper, a Python routine is written. Specifications are given below.

- Python version: 3.10.5
- Linear programming module used: Gurobi Optimizer version 11.0.1 build v11.0.1rc0
- OS: Windows 11
- CPU model: 12th Gen Intel(R) Core(TM) i7-12700H
- Thread count: 14 physical cores, 20 logical processors, using up to 20 threads.

A greyscale cover image, similar to the one in Figure 4, of size 64×64, is used. This translates to 4096 pixels. Random bit sequences of payload 0 to 0.3, in increments of 0.01, are generated. The term "payload" refers to the ratio of the length of hidden information (in bits) to the number of pixels in the cover image [9]. For example, a random bit sequence of payload 0.2 would mean that it has a length of $0.2 \times 4096 \approx 819$ bits.

The cover image is embedded with up to 20 different bit sequences for each payload, both with and without utilizing Gale-Berlekamp Switch Game. The number of LSB changed is recorded for every bit sequence, and the average was taken for each payload. The time taken to run the routine was also recorded. The results are depicted in Figure 11 and Figure 12.

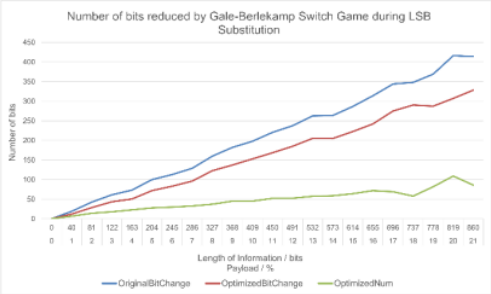

Figure 11. Graph depicting the number of bits reduced by Gale-Berlekamp Switch Game during LSB substitution.

In Figure 11, the blue line, OriginalBitChange, shows the average number of LSB changed when embedding information of varying payload without utilizing Gale-Berlekamp Switch Game. The red line, OptimizedBitChange, shows the average number of LSB changed when embedding information of varying payload with the utilization of Gale-Berlekamp Switch Game. The green line, OptimizedNum, is the difference between the OriginalBitChange (blue line) and OptimizedBitChange (red line). In other words, the number of LSB changes we can reduce when we incorporate Gale-Berlekamp Switch Game in LSB substitution.

From Figure 11, we can observe that even as payload increases, the percentage of number of LSB changes reduced with respect to the length of information stays generally constant. From the data collected, it is calculated that Gale-Berlekamp Switch Game can reduce the number of LSB changes by around 25% across all payloads.

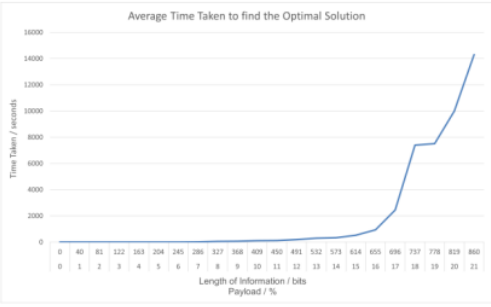

Figure 12. Graph depicting the average time taken to find the optimal solution to the Gale-Berlekamp Switch Game

However, as payload increases, the time taken to find the optimal solution to the Gale-Berlekamp Switch Game increases at an increasing rate, resembling an exponential growth. The similar time taken observed for payload of 18 and 19 can be attributed to the size of matrix used for Gale-Berlekamp Switch Game. Both payload of 0.18 and 0.19 translates to a matrix of size $\sqrt{737} \approx \sqrt{778} \approx 27 \times 27$.

Under the specification given above, the routine was unable to find the optimal solution to bit sequence of payload 0.22 after running continuously for 24 hours, after which the terminal was killed and the routine was deemed unsuccessful in finding the optimal solution within a reasonable amount of time.

This can be visualized by the size of matrix used for Gale-Berlekamp Switch Game. A payload of 0.21 translates to a matrix of size $\sqrt{860} \approx 29 \times 29$, while a payload of 0.22 translates to a matrix of size $\sqrt{901} \approx 30 \times 30$. Thus, we can conclude that the maximum length of information we can embed using the presented methodology (under the given specification) is $29^2 = 841$ bits = $\lfloor 105.125 \rfloor = 105$ ASCII characters.

5 CONCLUSION AND FUTURE WORK

Incorporating Gale-Berlekamp in Image Steganography leads to lesser bit changes when embedding the same information within the same image, without compromising the integrity of the information. The resulting Stego Image is less detectable by detection tools and thus more secure. Thus, this approach shows promise in advancing the field of steganography.

However, the high time complexity of solving integer optimization problem, as demonstrated by exponential increase in time taken as length of information increase, suggests that this methodology may not be very well suited for very large datasets. Further research can be done by expanding the Gale-Berlekamp Switch Game to a three-dimensional cube, thus decreasing the proportion of decision variables (i.e. higher lightbulbs to switches ratio). Finding a heuristic approach to this NP-hard problem may also be better to produce a faster and near-optimal solution.

6 ACKNOWLEDGEMENTS

This paper would not have been possible without the help and support of my supervisor, Asst Prof Kiah Han Mao. I would like to acknowledge the funding support from Nanyang Technological University – URECA Undergraduate Research Programme for this research project.

7 REFERENCES

[1] Zelazko, A. (2023, November), RGB colour model. Encyclopedia Britannica.

https://www.britannica.com/science/RGB-colour-model.

[2] Nio, N., Kadokawa Corporation (2018, October). Bloom into You.

[3] Neeta, D., Snehal, K., & Jacobs, D. (2006, December). Implementation of LSB steganography and its evaluation for various bits. In *2006 1st international conference on digital information management* (pp. 173-178). IEEE.

[4] Ge, S., Gao, Y., & Wang, R. (2007, August). Least significant bit steganography detection with machine learning techniques. In *Proceedings of the 2007 international workshop on Domain driven data mining* (pp. 24-32).

[5] Hashim, M. M., Rahim, M. S. M., Johi, F. A., Taha, M. S., Al-Wan, A. A., & Sjarif, N. N. A. (2018). An extensive analysis and conduct comparative based on statistical attach of LSB substitution and LSB matching. *International Journal of Engineering & Technology*, *7*(4), 4008-4023.

[6] Roth, R. M., & Viswanathan, K. (2008). On the hardness of decoding the Gale–Berlekamp code. *IEEE Transactions on Information Theory*, *54*(3), 1050-1060.

[7] Zakaria, A. A., Hussain, M., Wahab, A. W. A., Idris, M. Y. I., Abdullah, N. A., & Jung, K. H. (2018). High-capacity image steganography with minimum modified bits based on data mapping and LSB substitution. *Applied Sciences*, *8*(11), 2199.

[8] Hung, L. V., & Yu, X. (2021). Variants of the Gale-Berlekamp Switching Game and their Solutions: Balancing the Rectangle and the Cube. *arXiv preprint arXiv:2108.09290*.

[9] Lerch, D. (n.d.), Comparison of Image Steganography Tools. Daniellerch. https://daniellerch.me/stego/aletheia/tool-comparison-en/

URECA Research Paper.docx

ORIGINALITY REPORT

3%	1%	1%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Nanyang Technological University Student Paper	1%
2	Submitted to London Business School Student Paper	<1%
3	Bowlin, Garry. "Maximum frustration of bipartite signed graphs", Proquest, 20111004 Publication	<1%
4	Submitted to Purdue University Student Paper	<1%
5	mdpi.com Internet Source	<1%
6	JAE-GON KIM, YEONG-DAE KIM. "A space partitioning method for facility layout problems with shape constraints", IIE Transactions, 1998 Publication	<1%
7	Mohamed Abdel Hameed, Saleh Aly, M. Hassaballah. "An efficient data hiding method based on adaptive directional pixel value	<1%

differencing (ADPVD)", Multimedia Tools and Applications, 2017
Publication

Exclude quotes	On	Exclude matches	< 3 words
Exclude bibliography	On		