Chapter

Graph Models in Information Hiding

Hanzhou Wu

Abstract

Information hiding allows us to hide secret information into digital objects such as images without significantly distorting the objects. The object containing hidden information will be transmitted to a data receiver via a probably insecure channel. To securely transmit the object carrying hidden information, the distortion caused by data embedding should be as low as possible, which is referred to as the rate-distortion optimization problem. Many conventional methods optimize the data embedding procedure by a heuristic fashion, which may be not optimal in terms of the rate-distortion performance. In this chapter, we introduce novel approaches that use graph theory for information hiding. These graph models are general and can be used for improving the rate-distortion performance of information hiding systems. In addition to rate-distortion optimization, recent graph models used for system design of information hiding will be also reviewed. This chapter is intended as a tutorial introducing advanced graph models applied to information hiding.

Keywords: Information hiding, steganography, steganalysis, watermarking, graph theory, rate-distortion optimization, security, covert communication

1. Introduction

Information hiding [1] is referred to as the art of hiding secret data into a digital object such as image and video without significantly distorting the object content. The workflow of information hiding can be described as follows. A data hider uses a secret key to embed secret data into a digital object (typically also called *cover*) by modifying the cover content. The resulting object carrying secret data (typically also called *stego*) does not introduce noticeable artifacts and will be transmitted to a data receiver. During transmission, the stego may be attacked or analyzed by an adversary. When the data receiver receives the probably altered stego, he will be able to perfectly or near-perfectly reconstruct secret data from the stego. Different from cryptography that leaves clear traces on encrypted data, information hiding can be used to even conceal the existence of the current communication.

Information hiding is actually an emerging and interdisciplinary research area covering various applications, among which watermarking [2] and steganography [3] are two popular branches nowadays. In particular, the ease of use, reproduction, edit and distribution of digital commercial products has led increasing concern to copyright protection for digital media files, resulting in that, watermarking is still a major activity in digital media processing. Different from watermarking that aims to protect the copyright of digital product, steganography, as another important

branch of information hiding, conceals the existence of the hidden information, and therefore can be used for secret communication. It works by hiding a secret message into an innocent cover such as text, image, video and audio, by modifying the noise-like component of the cover in such a way that the resulting stego has a low distortion to the cover and therefore will not arouse suspicion.

As mentioned above, information hiding requires us to select a digital object as the cover to be embedded. A straightforward idea to categorize information hiding algorithms is therefore based on the type of selected cover. For example, image is still the most popular cover type due to its wide distribution over social networks. A number of information hiding algorithms are originally designed for images, e.g., [4–6]. Other covers such as video sequences [7–10], speech [11], natural language [12] are of increasing interest to researchers. Recently, studies have demonstrated that even social behaviors [13, 14] can be exploited for information hiding. In terms of data embedding strategies, information hiding can be roughly divided into four categories, i.e., *spatial domain*, *transformed domain*, *compressed domain* and *structural domain*. Taking digital image for explanation, spatial domain is corresponding to the most intuitive and efficient implementation, which allows us to directly modify the spatial pixels of an image. Data embedding in transformed domain often requires a higher computational complexity due to the transform operation. For example, by applying the discrete cosine transform (DCT) operation to a gray-scale image, one can use the DCT coefficients for hiding data. Comparing with data embedding in spatial domain, embedding in the transformed domain is more robust to malicious attacks. Data embedding in compressed domain is more complex and challenging than that in the spatial/transformed domain because the entropy in the compressed domain is high and therefore leads to small capacity for carrying secret data. Structural domain means to embed secret data in the file structure of a cover, implying that, no visual/auditory distortion will be introduced. However, from the viewpoint of an adversary, clear marks in the reserved field of a file may reveal the existence of information hiding, thus reducing the security.

From the information-theoretic view, information hiding can be modeled as a communication task, which aims to securely transmit a secret message to a receiver by embedding the secret message into a cover without significantly impairing the cover. On the one hand, we expect to embed as many secret bits as possible so that a sufficient payload can be carried by the cover. On the other hand, for a fixed size of secret message, we expect to keep the distortion caused by data embedding as low as possible so that the stego will be seemingly normal and thus will not arouse suspicion. Accordingly, a most commonly used method for evaluating information hiding algorithms is rate-distortion performance, where "rate" means the payload size and "distortion" measures the difference between the stego and the cover.

Many information hiding algorithms optimize the rate-distortion performance in a heuristic fashion. For example, reversible image watermarking [15, 16] allows both secret data and the original cover to be reconstructed at the data receiver side. A common approach to realize reversibility is shifting a so-called prediction-error histogram (PEH) in a reversible way, i.e., the original PEH can be fully recovered from the marked PEH. One has to select suitable shifting parameters such that the distortion caused by shifting can be kept low, thus resulting in good rate-distortion performance. Many works [17–19] heuristically select the shifting parameters for embedding, which may be not optimal in terms of rate-distortion optimization due to cover diversity. It motivates us to study deterministic optimization methods that are applicable to various covers and lead to optimal/near-optimal performance.

Based on the aforementioned analysis, in this chapter, we will introduce graph models that can be used for rate-distortion optimization of information hiding. All these models follow the identical framework. That is, an optimization problem to be

addressed in information hiding is first modeled as a graph problem. Then, by using a graph algorithm, the optimal solution to the graph problem can be found, indicating that, one can use the optimal parameters or strategies derived from the optimal solution of the graph problem (under constraints) for information hiding.

The rest of this chapter is organized as follows. First, we provide basic concepts and algorithms in graph theory in Section 2. Then, in Section 3, we introduce graph models in information hiding. Finally, we conclude this chapter in Section 4.

2. Basic concepts and algorithms in graph theory

A graph $G(V, E)$ is a non-linear structure that consists of two sets, i.e., node set (or say vertex set) $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$, where n and m represent the size of V and the size of E, respectively. One may write $|V| = n$ and $|E| = m$. Each edge e_i in E connects two nodes v_j and v_k in V, namely, $e_i = (v_j, v_k)$. The two nodes are typically different from each other, i.e., $j \neq k$. $G(V, E)$ is either *directed* or *non-directed*. The term "directed" means each edge in E is associated with a direction, e.g., $e_i = (v_j, v_k)$ represent a directed edge from v_j to v_k. Therefore, (v_j, v_k) is different from (v_k, v_j) for a directed graph. In contrast, in "non-directed" setting, there has no difference between (v_j, v_k) and (v_k, v_j), i.e., they are equivalent to each other. Unless mentioned, we will consider $G(V, E)$ as a non-directed graph. Many graph models and algorithms can be found in the literature. In this chapter, for self-contained, we briefly review graph techniques that will be used latter.

2.1 Graph traversal and coloring

Graph traversal is referred to as the process of visiting each node in a graph. It can be done by using depth-first search (DFS) or breadth-first search (BFS) [20]. Taking DFS for explanation, a node u is first randomly selected as the starting node and marked to show that it has been previously processed (or visited). If there is a node v that is adjacent to u and v has not been processed, v will be selected as the new starting node and should be marked as processed. Here, a node is adjacent to another node means that there is an edge connecting them. The above process will be recursively performed until all the nodes are visited. In summary, when the DFS procedure arrives at a certain node u, it tries to visit an adjacent node v of u, then an adjacent node w of v, and so on. Notice that, if $G(V, E)$ has multiple connected components, the DFS procedure may be executed multiple times so that each time a connected component can be processed. Unlike DFS, if the BFS procedure arrives at a node u, it first processes all the adjacent nodes of u. Then, for each processed adjacent node v, it will continue the similar procedure. Though DFS and BFS visit nodes in a different way, they have the same time complexity, i.e., $O(|V| + |E|)$.

During the DFS or BFS, each node can be assigned with a color, which refers to graph (node) coloring [20]. We can use two different colors (say "red" and "black") for graph coloring so that each node is colored with either red or black. Meanwhile, for each node (that is not isolated), there is always another adjacent node that has the different color to the present node. It can be done during the visiting process. For example, assuming that we have reached a node u from a node v, we can assign a color that is different from v to u, i.e., if v is black, u will be colored with red.

2.2 Weighted bipartite graph matching

$G(V, E)$ is called a *bipartite graph* if the node set V can be partitioned into two disjoint node sets V_1 and V_2 that all edges in E connect a node in V_1 and a node in V_2.

Namely, we can write $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \varnothing$ and $\forall\ (u, v) \in E, u \in V_1, v \in V_2$. A matching of G is such a subset of E that no two edges in the subset share a common node. A node is *matched* if it is connected by one edge in the matching. Otherwise, the node is said *non-matched*. A maximum matching of G is such a matching that it has the largest size, i.e., the total number of edges in the matching is maximum. A *weighted bipartite graph* means that each edge in E is assigned with a weight. The minimum weighted maximum matching of a weighted bipartite graph is such a maximum matching that the sum of weights of edges is minimum. One can use Hungarian algorithm or minimum cost maximum flow algorithm [20] to compute the minimum weighted maximum matching.

3. Graph models in information hiding

Figure 1 shows the general framework of information hiding, where an image is used as the cover. In this chapter, unless mentioned, we will use a grayscale image as the cover object. Referring to **Figure 1**, let $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ be a cover sequence, where N is the total number of cover elements. For example, if the cover is a 8-bit gray-scale image, all possible $x_i \in \mathbf{x}$ will be in range [0, 255], i.e., $\mathbf{x} \in \{0, 1, \ldots, 255\}^N$. Given secret data $\mathbf{m} = \{m_1, m_2, \ldots, m_L\} \in \{0,1\}^L$ and secret key $\mathbf{k} \in \{o, 1\}^T$, the goal of data embedding is to modify $\mathbf{x}$ as a new sequence $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$ (also called stego sequence) that $\mathbf{m}$ can be carried by $\mathbf{y}$. Namely, we can write.

$$\mathbf{y} = \text{Embed}(\mathbf{x}, \mathbf{m}, \mathbf{k}). \tag{1}$$

For data extraction, a data receiver should be able to extract $\mathbf{m}$ from $\mathbf{y}$, i.e.,

$$\mathbf{m} = \text{Extract}(\mathbf{y}, \mathbf{k}). \tag{2}$$

In some applications, e.g., reversible watermarking [16], it is further required that $\mathbf{x}$ should be perfectly recovered from $\mathbf{y}$. For a fixed size of $\mathbf{m}$, it is necessary to keep the distortion due to data embedding as low as possible. Namely, we expect to minimize the distortion between $\mathbf{x}$ and $\mathbf{y}$, denoted by $D(\mathbf{x}, \mathbf{y})$, for a fixed $\mathbf{m}$. In other words, for a fixed $D(\mathbf{x}, \mathbf{y})$, we hope to embed as much secret data as possible, i.e., the size of $\mathbf{m}$ is expected to be as large as possible. Many advanced information hiding algorithms are designed along this line. In the following subsections, we will introduce graph methods that can be used to deal with the above optimization task.

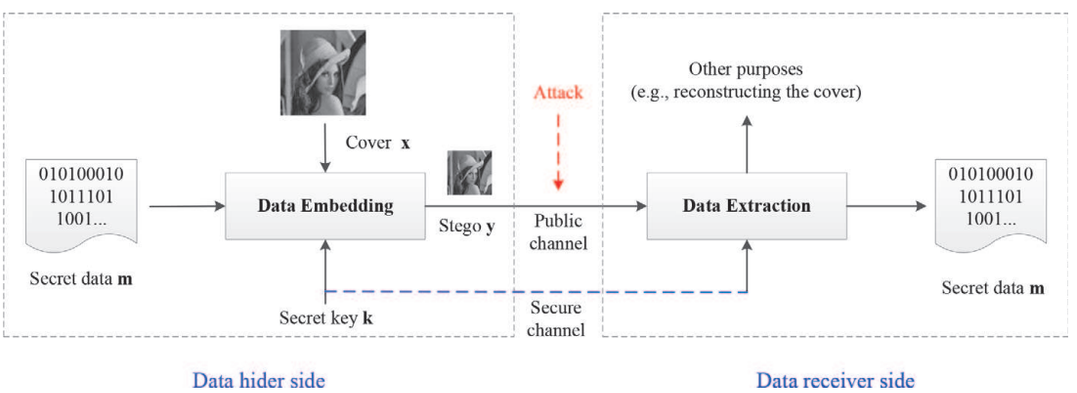

Figure 1.
General framework of information hiding.

3.1 Multi-bit mapping using graph coloring

A simplest information hiding method is least significant bit (LSB) replacement, which enables us to modify **x** by replacing the LSBs of all elements in **x** with secret bits. The data embedding operation for LSB replacement can be described as:

$$y_i = x_i - (x_i \bmod 2) + m_i, \tag{3}$$

where $1 \le i \le N = L$, "mod" means the modulo operation, and m_i is the i-th secret bit to be embedded. For example, if the value of the i-th cover pixel is 247 (i.e., $x_i = 247$), after embedding, the pixel value will be still 247 (i.e., $y_i = 247$) if the secret bit to be embedded is 1 (i.e., $m_i = 1$). Otherwise, if the secret bit to be embedded is 0 (i.e., $m_i = 0$), the pixel value will become 246, i.e., $y_i = 247 - (247 \bmod 2) + 0 = 246$. For data extraction, by extracting the LSBs of **y**, **m** can be recovered, i.e.,

$$m_i = y_i \bmod 2, 1 \le i \le N. \tag{4}$$

For example, a secret bit 1 (i.e., $m_i = 1$) can be extracted from a pixel whose value is 247 (i.e., $y_i = 247$) because $m_i = y_i \bmod 2 = 247 \bmod 2 = 1$. In LSB replacement, there are two ways to modify a cover element, i.e., keeping the cover element unchanged or flipping the LSB of the cover element. It indicates that, from the information-theoretic view, each cover element can carry exactly one secret bit, meaning that, the maximum payload size for **x** is N (bits). Obviously, when the number of ways of modifying a cover element is larger than two, the cover element will be able to carry more bits. This can be analyzed in a graph. Clearly, each cover element is corresponding to a node in a graph. The edges between the nodes denote the modification relationship between the nodes. For example, **Figure 2(a)** shows the graph for LSB replacement. Each even cover element is corresponding to v_1 and the odd cover elements correspond to v_2. v_1 and v_2 are respectively mapped to "0" and "1". During data embedding, if a cover element corresponding to v_1 matches the secret bit, the cover element will be unchanged. Otherwise, it will be replaced with a new value so that the new value is corresponding to v_2, matching the bit.

Let $A(v)$ represent the adjacent set of the node v, e.g., $A(v_1) = \{v_1, v_2\}$ in **Figure 2(a)**. Notice that, we consider a node itself as an element of the adjacent set. It can be inferred that, the total number of modifications to a node is equal to the size of the adjacent set. Moreover, a larger number of modifications means that the node can be used to carry more bits. For example, in **Figure 2(b)**, for each node, the size of adjacent set equals 4, indicating that, each node can carry 2 bits. Obviously, given a graph, we can assign a bit-stream to each node, representing that the node can carry the assigned bit-stream (whose size may equal 1). In order to ensure that the

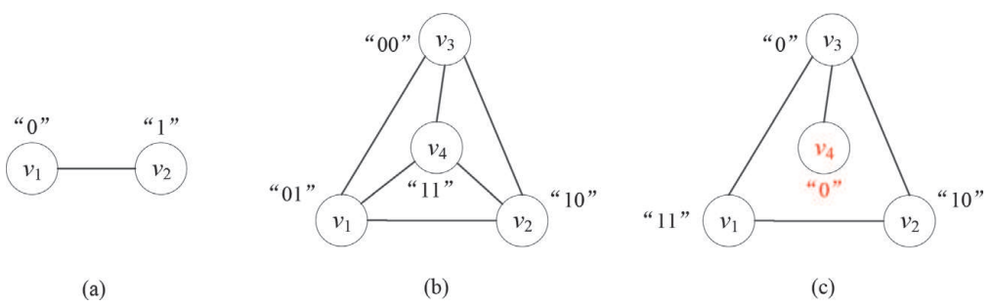

Figure 2.
Examples of analyzing information hiding in a graph: (a) two nodes $\{v_1, v_2\}$ are mapped to "0" and "1", (b) four nodes $\{v_1, v_2, v_3, v_4\}$ are respectively mapped to "01", "10", "00" and "11", (c) four nodes $\{v_1, v_2, v_3, v_4\}$ are respectively mapped to "11", "10", "0" and "0".

associated bit-streams can be used for information hiding, it is required that, for each non-isolated node, we should be able to select at least one node from its adjacent set such that the assigned bit-stream matches the secret data, i.e., the bit-stream should match a prefix of the secret data. For example, in **Figure 2(b)**, for all nodes, the adjacent set is $\{v_1, v_2, v_3, v_4\}$, which is mapped to {"01", "10", "00", "11"}. It can be inferred that, any secret data must start with a bit-stream belonging to {"01", "10", "00", "11"}. Therefore, we say the graph associated with bit-streams shown in **Figure 2(b)** can be used for information hiding. In contrast, the graph in **Figure 2(c)** cannot be used for information hiding. The reason is that, for v_4, the adjacent set is $\{v_3, v_4\}$, whose elements are all mapped to "0", which cannot match those secret streams starting with "1". Therefore, given a graph, how to assign bit-streams to nodes (so that all nodes can be utilized to carry data) is the key problem.

Mathematically, given a *connected* graph $G(V, E)$, the goal is to map each node to a bit-stream such that for each node, we can always select such a node from the adjacent set that the selected node matches the prefix of the secret data. Meanwhile, it is desirable that the assigned bit-streams can effectively exploit the modification relationship between nodes. It is defined as the multi-bit mapping problem, which can be addressed by multi-layer graph coloring [4], whose goal is to append a new bit (if any) to the end of previously assigned bits of each node during each layer so that all nodes can be finally mapped to a bit-stream. Below gives the details.

In the first layer, by using DFS or BFS, each node in the original graph G can be colored with either "red" or "black". Moreover, for each $u \in V$, there always exists such $v \in A(u)$ that v has a different color to u. We can assign a secret bit "0" to all red nodes and "1" to all black nodes. In this way, for each $u \in V$, there always exists one node $v \in A(u)$ that v has the opposite bit to u. In other words, for any cover element corresponding to $u \in V$, we can always modify the cover element such that the new value of the cover element corresponds to $v \in A(u)$ and the mapped bit of v is equal to the secret bit to be embedded.

Assuming that, the k-th layer coloring procedure has been completed, meaning that the maximum number of assigned bits for a node is k, we are to perform $(k + 1)$-th layer coloring. The detailed steps for $(k + 1)$-th layer coloring are as follows.

Step 1) Collect all nodes that have been previously mapped to a bit-stream with a length of k. For example, after executing the first layer coloring procedure, each node is mapped to one bit ("0" or "1"), i.e., the bit-stream has a length of 1. It means that, all nodes will be used in the second layer coloring since they are all mapped to a bit-stream with a length of 1.

Step 2) All collected nodes in **Step 1)** will be divided to 2^k sets according to the values of the bit-streams. For example, all the collected nodes will be divided to 2 subsets in the second layer. Namely, one set contains nodes that are mapped to "0", and the other set contains nodes that are mapped to "1". Let C_s represent the set including all nodes that are mapped to a bit-stream with a value of s, where $0 \leq s < 2^k$. For example, a node belongs to C_{15} if it is mapped to a bit-stream "001111", where $k = 6$. It is possible that C_s is an empty set. For each non-empty set C_s, all nodes belonging to C_s and all edges connecting two nodes belonging to C_s can be used to construct a graph $G_s(V_s, E_s)$, which should be a subgraph of $G(V, E)$.

Step 3) G_s may contain multiple connected components. We apply DFS or BFS to G_s so that each node in V_s can be assigned with one more bit. The use of DFS or BFS is similar to the first layer. Obviously, for each node in V_s, if we append the bit to the end of the previous bit-stream, the new bit-stream will have a length of $k + 1$. To ensure that the new bit-streams are suited to information hiding, some assigned bits in this step should be canceled, i.e., the bits assigned to some nodes (if any) in V_s will be canceled. In other words, the lengths of the new bit-streams of some

nodes in V_s will not become $k + 1$ because their assigned bits in this step are canceled. The details are given in the next step.

Step 4) Let $B_s = \{v \mid (u, v) \in E, u \in V_s, v \in V\text{-}V_s\}$ represent the check-set. It can be inferred that $B_s \cap V_s = \emptyset$, $B_s \subset V$ and $V_s \subset V$. For each node $v \in B_s$, if all adjacent nodes of v in V_s are assigned with the same bit in **Step 3)**, we randomly select one node among these adjacent nodes and cancel the newly assigned bit in **Step 3)** to the selected node. By processing all nodes in B_s, the newly assigned bits to some nodes in V_s can be canceled. The remaining assigned bits are regarded as the $(k + 1)$-th bit of the corresponding nodes. Repeat **Step 3–4)** until all C_s are processed.

The multi-layer graph coloring algorithm is finished when there has no new bit produced in a certain layer. For example, if there are 64 nodes in the original graph, the number of layers will be at most 6 since $64 = 2^6$. For better understanding the multi-layer coloring procedure, we provide an example in **Figure 3**, in which there are six nodes and nine edges in the graph. Our goal is to map each node in the graph to a bit-stream with an indefinite length. First of all, we can use DFS to assign either "0" or "1" to each node. In detail, assuming that the visiting order by DFS is $v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_3$, we can assign "0" to $\{v_1, v_4, v_6, v_3\}$ and "1" to $\{v_2, v_5\}$. Notice that, here, v_3 is visited from v_5. When v_5 has been previously mapped to "1", v_3 will be mapped to the opposite bit "0". $\{v_1, v_4, v_6, v_3\}$ and $\{v_2, v_5\}$ are two subsets that will be processed in the second layer. For the subset $\{v_1, v_4, v_6, v_3\}$, the visiting order by DFS is $v_1 \rightarrow v_3 \rightarrow v_4 \rightarrow v_6$, allowing us to assign "0" to $\{v_1, v_4\}$ and "1" to $\{v_3, v_6\}$. The check-set is $\{v_2, v_5\}$. For v_5, cancelation of any new bit is not needed, while for v_2, the bit "0" assigned to v_1 is pseudo-randomly canceled since all adjacent nodes of v_2 in $\{v_1, v_4, v_6, v_3\}$ are mapped to the same bit. For the subset $\{v_2, v_5\}$, the DFS order can be $v_2 \rightarrow v_5$, allowing us to assign "0" to v_2 and "1" to v_5. In this case, the check-set is $\{v_1, v_4, v_6, v_3\}$. For v_4, cancelation of any new bit is not needed, while for v_1, v_3 and v_6, "0" and "1" respectively assigned to v_2 and v_5 should be all canceled. In the third layer, since all nodes are isolated according to the previously assigned bits, the assignment is terminated. **Figure 3** has shown the final results, e.g., v_4 is finally mapped to "00".

It is inferred from **Figure 3** that, for each node, we can always select such a node from the adjacent set that the selected node must match a prefix of the secret bit-stream to be embedded, namely, the graph in **Figure 3** can be used for information hiding. For example, assuming that we have a cover sequence **x**, which corresponds to a node sequence $(v_1, v_3, v_5, v_2, v_4, v_4)$ (it is possible that two cover elements in **x** are corresponding to the same node), we can modify the node sequence as a new one $(v_1, v_3, v_6, v_2, v_3, v_4)$ if the secret data is **m** = [0010110100]. Based on the new node sequence and **x**, we can construct the stego sequence **y**. For data extraction, we first reconstruct the node sequence $(v_1, v_3, v_6, v_2, v_3, v_4)$ from **y**. Then, by orderly concatenating the bit-streams of the nodes in the sequence, we can reconstruct **m** without error. Notice that, the data hider and the data receiver should perform the same multi-layer graph coloring. Otherwise, the data receiver cannot retrieve **m**.

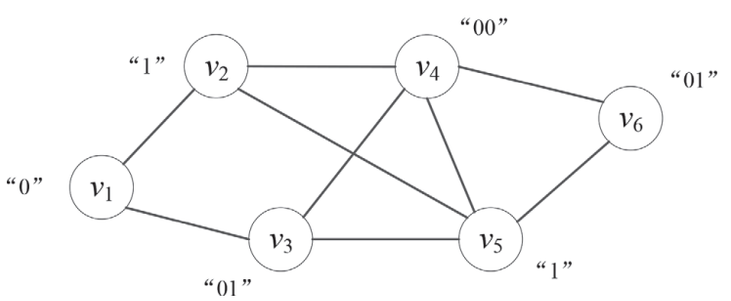

Figure 3.
Example for multi-layer graph coloring.

Remark: The multi-bit mapping by graph coloring enables us to map each cover element to a bit-stream with an indefinite length, which has effectively utilized the replacement relationship between different values of cover elements and therefore can achieve a high embedding rate (i.e., more secret bits can be hidden).

3.2 Reversible embedding using graph matching

Reversible (data) embedding, also called reversible watermarking or reversible data hiding, not only allows a data receiver to extract secret data from the stego, but also enables the cover to be perfectly reconstructed from the stego. It is quite useful for sensitive applications that require no permanent distortion of the cover such as military imaging. A straightforward idea to realize reversible embedding is lossless compression, which losslessly compresses the noise-like component of the cover to reserve extra room for data embedding, i.e., the extra room will be used to carry the secret data and the compressed code. As shown in **Figure 4**, by extracting the losslessly compressed code from the stego, one can reconstruct the original cover without any error. However, since it is not easy to significantly compress the noise-like component in a lossless way, the size of the secret data will be low.

Unlike lossless compression, histogram shifting (HS) [21] embeds secret data into a histogram determined from the cover by reversibly modifying the histogram. **Figure 5** shows an example of HS based reversible embedding in a gray-scale image. In **Figure 5**, a histogram determined from the original image is used to embed data. We first select two bins (peak bin/zero bin) from the histogram. The peak bin has the maximum occurrence, and the zero bin has no occurrence. Obviously, in **Figure 5**, the peak bin is "1" and the zero bin is "3". Then, we shift all bins between the two bins towards the zero bin by one step, i.e., the bin "2" will be shifted to the right. It is equivalent to modifying all pixels having a value of 2 as that have a new value of 3. Thereafter, we use the peak bin "1" to carry secret data. Assuming that, the secret data is "0100", for each pixel having a value of 1, it will be unchanged if the present secret bit to be embedded is "0"; otherwise, it will be replaced with a new value of 2 if the secret bit is "1". In this way, the secret data can be embedded, resulting in a new image containing secret data (also called marked image). It is easy for a data receiver to extract secret data and reconstruct the original image. First of all, secret data can be retrieved by processing all pixels having a value of 1 or 2 in the

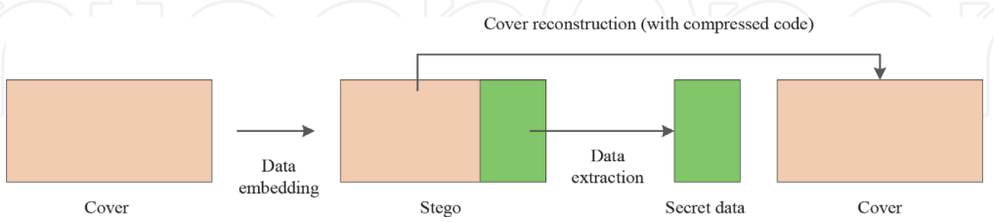

Figure 4.
Sketch for lossless compression based reversible embedding.

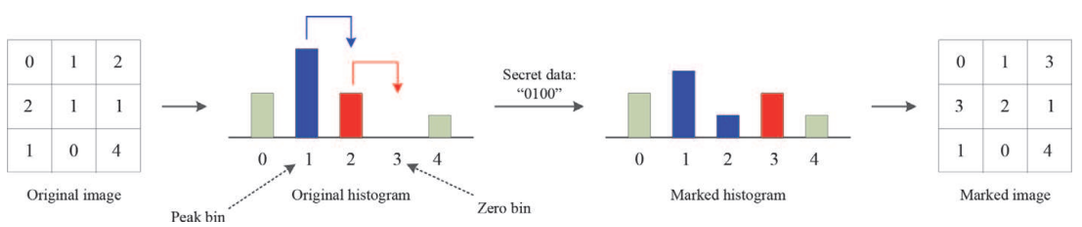

Figure 5.
An example of using HS for reversible embedding.

marked image. Then, all pixels that have a value of 2 in the marked image will be modified with a value of 1. Finally, for those pixels having a value of 3, they will be modified with a value of 2. In this way, the resulting image will be equal to the original image. We refer a reader to [21] for more details about HS based reversible embedding.

Figure 5 corresponds to the classical HS operation, which can be improved by using two techniques. The first one is using a prediction-error histogram (PEH) for reversible data embedding, rather than the histogram directly determined from the original cover. The second one is using more histogram bins for data embedding, rather than only one peak bin in **Figure 5**. For better understanding, **Figure 6** shows an example for constructing a PEH. In **Figure 6**, one can determine a histogram directly from the cover image, i.e., "Histogram I". It can be seen that, the maximum occurrence is less than 3000, meaning that, we can embed no more than 3,000 bits if we use only one peak bin. However, it can be significantly improved if we can produce a PEH, e.g., by dividing the pixels into two sets (i.e., cross set and dot set), the pixels in the dot set (which are unchanged during data embedding) can be used to predict the pixels in the cross set (e.g., using the average value of the four adjacent pixels to predict the present pixel), allowing us to determine the prediction errors (PEs) of all pixels in the cross set. The PEs can be then used to build a PE histogram (PEH), i.e., "Histogram II", from which we can find that the maximum occurrence is more than 150,000, indicating that, we can embed more than 150,000 secret bits by using a single PE bin. Obviously, we can use more PE bins for data embedding, which will result in the better rate-distortion behavior.

Mathematically, given a PEH **H**, let $h(v)$ be the frequency of the PEH bin with value v, which is assumed in the range $(v_{\min}, v_{\max})$. One may write $v_{\max} = -v_{\min} = L$, e.g., $L = 256$ for a 8-bit grayscale image. Let A and B be a set including all PEH bins and a set containing all non-zero occurrence bins. It means that, $A = \{v|\text{-}L < v < L\}$ and $B = \{v|h(v) > 0\} \subset A$. The above HS operation can be mathematically described as [22]: For a peak-bin set $P = \{p_1, p_2, \ldots, p_s\} \subset B$, we build two injective functions g_0 and g_1 such that $G_0 = \{g_0(p_1), g_0(p_2) \ldots, g_0(p_s)\} \subset A$, $G_1 = \{g_1(p_1), g_1(p_2) \ldots, g_1(p_s)\} \subset A$ and $G_0 \cap G_1 = \varnothing$. We also construct a third injective function f: $B\backslash P \rightarrow A\backslash(G_0 \cup G_1)$. Assuming that, the size of the secret data is exactly equal to $h(p_1) + h(p_2) + \ldots + h(p_s)$, for reversible embedding, we first use f to shift all PEH bins in $B\backslash P$ into $A\backslash(G_0 \cup G_1)$. Then, secret bits can be embedded into the PEH by further shifting the PEH bins in P into $G_0 \cup G_1$. Since g_0, g_1, f are injective, the original PEH can be reconstructed from the marked PEH (i.e., the PEH containing hidden information). **Figure 7** shows the relationship between three sets A, B, P and three

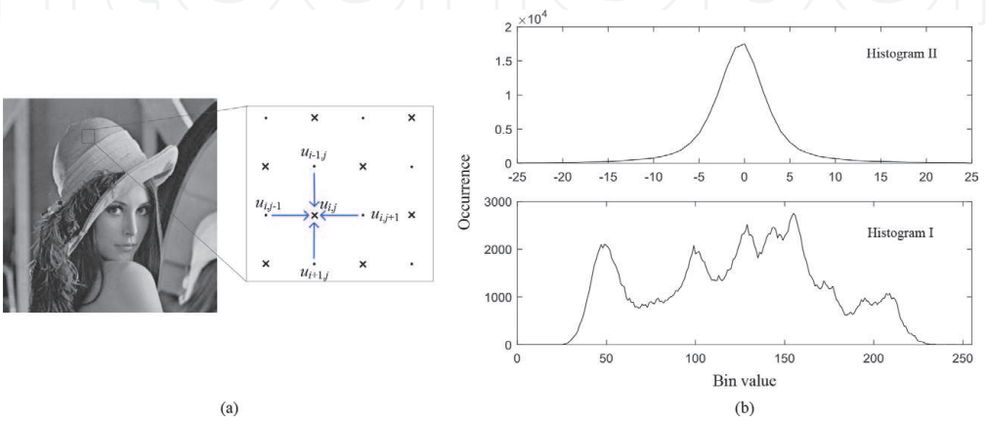

Figure 6.
An example for constructing a PEH: (a) predicting pixels marked as "cross" with pixels marked as "dot", (b) two PE histograms, where "Histogram I" is determined directly from the original image shown in (a), and "Histogram II" is determined by collecting the prediction errors for all possible $u_{\{i,j\}}$ shown in (a).

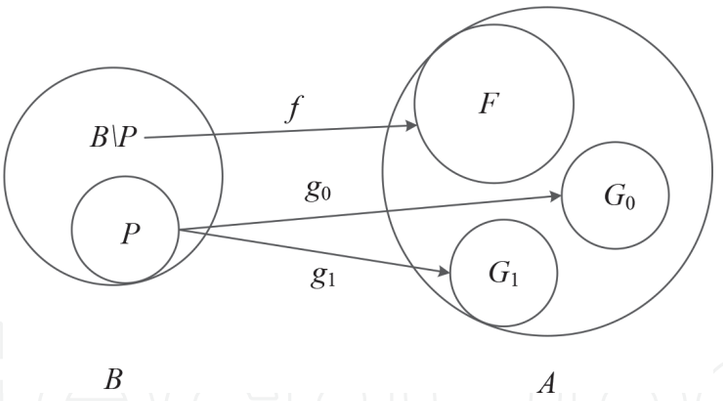

Figure 7.
Relationship between three sets A, B, P *and three functions* g_0, g_1, f.

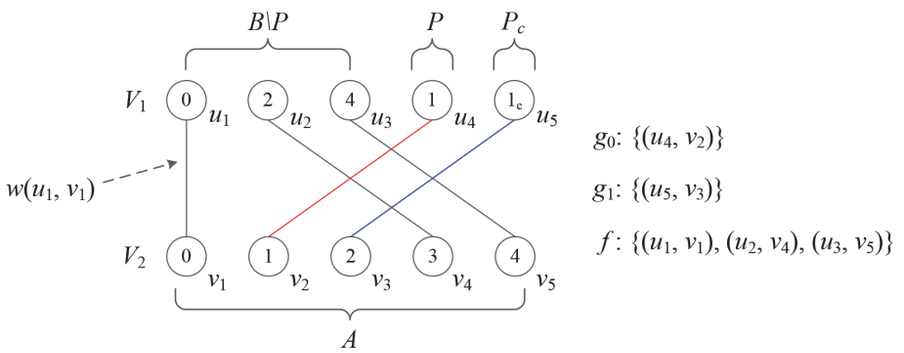

Figure 8.
Building a (weighted) bipartite graph for the example in ***Figure 5****.*

functions g_0, g_1, f. For better understanding, we take **Figure 5** for example. In **Figure 5**, we have $A = \{0, 1, 2, 3, 4\}$, $B = \{0, 1, 2, 4\}$, $P = \{1\} \subset B$, $G_0 = \{g_0(1) = 1\}$ and $G_1 = \{g_1(1) = 2\}$. Moreover, we have $f(0) = 0, f(2) = 3$ and $f(4) = 4$. Notice that, though **Figure 5** does not use any prediction procedure, one may assume that the prediction value of each pixel is exactly equal to zero. Thus, "Histogram I" is actually a special case of PEH.

Shifting PEH bins requires us to accordingly modify the image pixels so that the marked image is corresponding to the shifted PEH. Regardless of the detailed steps of modifying the image pixels, it can be easily inferred that, in **Figure 7**, for a fixed P, it is necessary to find the optimal (g_0, g_1, f) so that the distortion introduced by data embedding can be minimized. If this problem can be well addressed, one can enumerate P to find optimal (P, g_0, g_1, f) so that the distortion is globally minimum. Fortunately, we can use bipartite graph matching to deal with the former problem. It requires us to model all involved PEH bins as graph nodes and the three injective functions as sets of edges. For example, **Figure 8** shows the bipartite graph for the example shown in **Figure 5**. It can be inferred that, for the bipartite graph $G(V, E)$, the node set V consists of two disjoint subsets V_1 and V_2, where $V = V_1 \cup V_2$, $V_1 = B \cup P_c = (B \backslash P) \cup P \cup P_c$ and $V_2 = A$. Moreover, the edge set E can be divided to three subsets E_1, E_2 and E_3, where $E_1 = \{(u, v) \mid u \in B \backslash P, v \in V_2\}$, $E_2 = \{(u, v) \mid u \in P, v \in V_2\}$, and $E_3 = \{(u, v) \mid u \in P_c, v \in V_2\}$. The three injective functions (g_0, g_1, f) will be then corresponding to the subsets (E_2, E_3, E_1), respectively. Notice that, though A and B have the common elements, they correspond to different nodes. Moreover, P and P_c include the same elements though they correspond to different nodes.

Each edge in E tells us how to process the corresponding PEH bin, e.g., in **Figure 8**, the edge (u_2, v_4) means that the PEH bin with a value of 2 will be shifted to the PEH bin with a value of 3. Obviously, shifting a PEH bin will introduce

distortion. If the distortion is additive, i.e., the overall distortion caused by data embedding is defined as the sum of distortion caused by each cover element, each edge could be associated with a weight representing the distortion of shifting the corresponding PEH bin, resulting in that the overall distortion should be the sum of weights of all edges. For example, in **Figure 8**, we can determine a weight for each edge, e.g., $w(u_1, v_1)$ denotes the distortion caused by shifting the PEH bin "0" to "0" (i.e., unchanged) and can be therefore assigned to the edge (u_1, v_1). Thus, the overall distortion due to HS in **Figure 5** is equal to the sum of weights of edges shown in **Figure 8**. Here, we ignore the steps to determine the weights since it is not the main interest of this chapter. We refer a reader to [22] for more details about the weight determination.

Therefore, we can conclude that, once g_0, g_1 and f are determined, we can easily construct the corresponding bipartite graph. However, (g_0, g_1, f) can be optimized from the viewpoint of rate-distortion optimization. Different (g_0, g_1, f) correspond to different bipartite graphs. All these bipartite graphs have the same node set, and the only difference is that the edge set is different from each other. Obviously, by merging all the nodes and edges, we can construct a new bipartite graph $G^*(V^*, E^*)$. $G(V, E)$ is therefore a subgraph of $G^*(V^*, E^*)$, meaning that, arbitrary (g_0, g_1, f) can be modeled as a subset of E^*. In terms of rate-distortion performance, we expect to find such (g_0, g_1, f) that they introduce the minimum distortion. How to find such optimal (g_0, g_1, f) is very important. Rethinking **Figure 8**, We can further infer that, g_0, g_1 and f ensure that, each node in V_1 can be matched by exactly one node in V_2, i.e., (g_0, g_1, f) is corresponding to a *maximum matching* of G. Since $V = V^*$ and $E \subset E^*$, we can find that (g_0, g_1, f) should be also a *maximum matching* of G^*. Since we expect that the distortion caused by (g_0, g_1, f) is minimum, it is further inferred that the optimal (g_0, g_1, f) is corresponding to the *minimum weight maximum matching* of G^*. Thus, once we have built G^*, by applying any graph algorithm that solves the *minimum weight maximum matching (MWMM)* of a bipartite graph, we can find the optimal (g_0, g_1, f) with respect to rate-distortion optimization. In order to build G^*, we first initialize $V^* = V = V_1 \cup V_2$, $V_1 = B \cup P_c = (B \backslash P) \cup P \cup P_c$ and $V_2 = A$. Then, for each PEH bin, we collect all the possible shifting operations, which allows us to add the corresponding edges to G^* so as to construct E^* [22].

For better understanding, **Figure 9** shows an example for searching the optimal (g_0, g_1, f). In **Figure 9(a)**, each node in V_1 involves multiple edges, implying that, the corresponding PEH bin can be shifted to any one among multiple candidates. For example, the node u_1 is connected by four different edges, i.e., (u_1, v_1), (u_1, v_2), (u_1, v_3), (u_1, v_4), indicating that, there are four possible ways to shift "-5", i.e., shifting "-5" to any one in {"-6", "-5", "-4", "-3"}. By solving the MWMM, we can construct a subgraph as shown in **Figure 9(b)**, from which we can identify the optimal (g_0, g_1, f), e.g., in **Figure 9(b)**, g_0 is corresponding to $\{(u_8, v_6), (u_9, v_8)\}$, i.e., $g_0(-1) = -1$ and $g_0(1) = 1$, which means that, for those pixels with a PE value of "-1" or "1", a part of them will be unchanged so as to carry secret bits "0"s.

Remark: Reversible embedding by weighted graph matching enables us to find the optimal HS parameters from the viewpoint of rate-distortion optimization, i.e., the optimal HS parameters can be obtained by determining the MWMM, providing a different perspective to the conventional optimization approaches. However, it is admitted that, in the graph matching model, it is required that P has been fixed in advance. If we relax this requirement, the time complexity will become very high since enumerating all possible P is time consuming [22]. For example, there are a total of $\binom{100}{4}$ ways to construct P if we have $|P| = 4$, $|B| = 100$ (notice that, $P \subset B$). Therefore, for a large $|P|$, when P is also required to be optimized, how to reduce

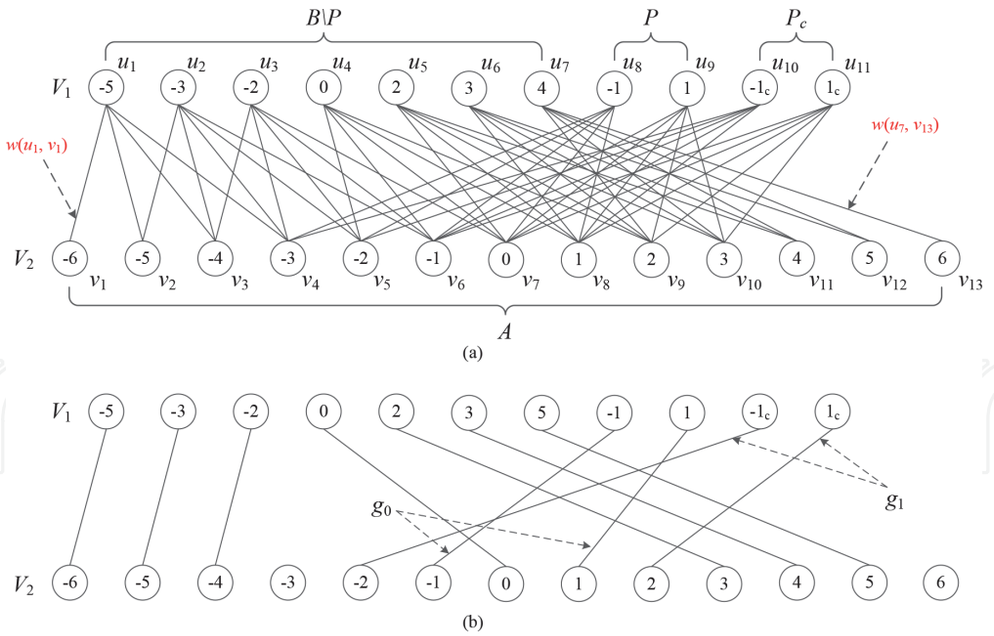

Figure 9.
Example for finding optimal (g_o, g_1, f)*: (a)* $G^*(V^*, E^*)$*, (b) the MWMM. In (b), all nodes in* V_1 *should be matched by exactly one node in* V_2*, and it is possible that some nodes in* V_2 *may be not matched since* $|V_1| \leq |V_2|$*, e.g.,* v_4 *and* v_{13} *are not matched in this example.*

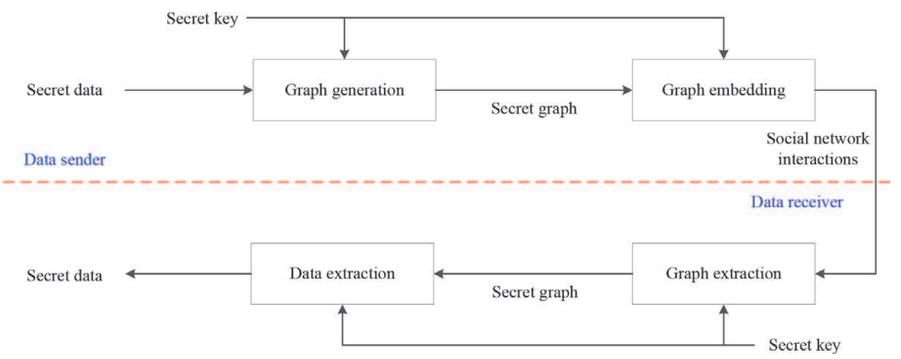

Figure 10.
General framework for graph steganography.

the overall computational complexity needs further investigation. Heuristic search strategies would be desirable for dealing with this problem.

3.3 Graph steganography in social networks

Many conventional algorithms use media objects as the cover. Recently, social behaviors are exploited for steganography. Comparing with media objects, social behaviors can be more easily concealed by the very large number of normal social activities. Moreover, both the data sender and the data receiver can easily integrate into the social networks. It is possible for the data receiver that he observes social behaviors for data extraction, without taking any other actions, which can well protect the real identify of the data receiver. From the point of algorithm design, one can extend existing methods originally designed for media objects to social behaviors, which may not well exploit the characteristics of social networks. On the other hand, by modeling social networks as graphs, we can use graph theory for conveying secret data, which is referred to as *graph steganography* [13, 14].

Figure 10 shows the general framework for graph steganography. In **Figure 10**, the data sender (or say data hider) first converts secret data to a (directed) graph.

Then, the graph will be embedded into social network by producing a sequence of seemingly-normal social interactions. For data receiver, he can retrieve the secret graph by observing the interactions without taking any other actions. In this way, he can finally recover secret data from the graph by inverting the graph generation procedure. In the following, we show the core steps for the general framework.

Given secret data **m**, the graph generation procedure first converts **m** to a non-directed graph $G_1(V_1, E_1)$. Then, $G_1(V_1, E_1)$ is extended to a directed graph $G_2(V_2, E_2)$ that can be released in a social network platform by orderly producing a sequence of interactions. In detail, given n nodes indexed from 1 to n, we can construct $2^{\binom{n}{2}}$ different graphs, each of which is corresponding to a binary string that has a length of $\binom{n}{2}$. Suppose that $\mathbf{m} \subset \{0, 1\}^L$, where L here represents the size of **m**, i.e., $|\mathbf{m}| = L$. One can append "0" (if necessary) to the end of **m** so that $L = \binom{n}{2}$. In other words, we can always assume that $n = \left(\sqrt{8L+1}+1\right)/2$. In order to construct $G_1(V_1, E_1)$, we first initialize $V_1 = \{v_1, v_2, \ldots, v_n\}$ and $E_1 = \varnothing$. Then, for each $m_k \in \mathbf{m}$, we add an edge (v_x, v_y) to E_1 if and only if $m_k = 1$, where x and y are determined as:

$$x = \min\left\{j \mid 0 < j < n, \sum_{i=1}^{j}(n-i) \geq k\right\}, \tag{5}$$

and

$$y = x + k - \sum_{i=1}^{x-1}(n-i). \tag{6}$$

In this way, $G_1(V_1, E_1)$ can be finally obtained. **Figure 11** shows an example for constructing $G_1(V_1, E_1)$, from which we can find that each bit in **m** is corresponding to the existence of the corresponding edge, e.g., $m_1 = 1$ implies that there is an edge between v_1 and v_2, and $m_8 = 0$ indicates that there has no edge between v_3 and v_4. It can be easily inferred that, G_1 may contain multiple connected components. One can directly release G_1 in a social network platform by producing social operations. For example, a set of social bots can be used for producing social behaviors so that G_1 can be carried by these social behaviors. The social bots will be corresponding to V_1, and the social behaviors are corresponding to E_1. Since G_1 is a non-directed graph and may contain multiple connected components, it typically requires the data sender and the data receiver to share the mapping relationship between social

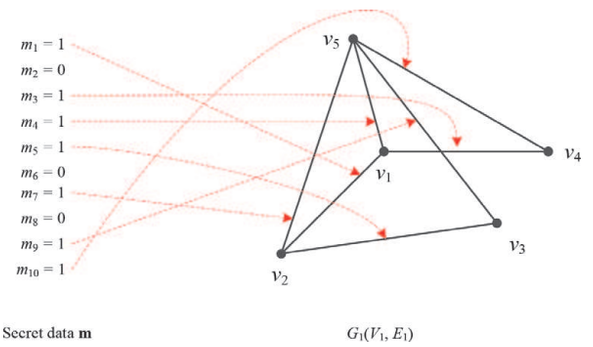

Figure 11.
An example of converting secret data to a non-directed graph.

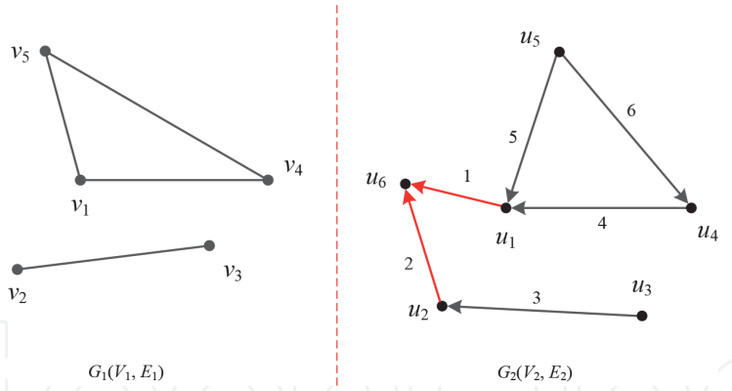

Figure 12.
An example for constructing G_2 *according to* G_1.

bots and nodes in V_1 so that the data receiver can reliably extract **m** by observation. Actually, G_1 can be extended to a directed graph so that there has no need for the data sender and the data receiver to share the mapping relationship between social bots and graph nodes. In other words, by extending $G_1(V_1, E_1)$ to a directed graph $G_2(V_2, E_2)$, the social bots can orderly produce social behaviors based on G_2 so that the data receiver can perfectly reconstruct G_2 by observation without knowing the mapping relationship between nodes in G_2 and social bots in advance (because the mapping relationship can be determined during observation).

In order to construct $G_2(V_2, E_2)$, we first initialize $V_2 = \{u_1, u_2, \ldots, u_{n+1}\}$ and $E_2 = \emptyset$. Then, we update E_2 by orderly processing $V_1 = \{v_1, v_2, \ldots, v_n\}$. In detail, for each $v_i \in V_1, 1 \leq i \leq n$, we update $E_2 = E_2 \cup \{(u_i, u_{n+1})\}$ if there does not exist such $j < i$ that $(v_j, v_i) \in E_1$. Otherwise, for all possible $j < i$, we add an edge (u_i, u_j) to E_2 if $(v_j, v_i) \in E_1$. Notice that, G_2 is a directed graph, meaning that, each edge in E_2 will be associated with a direction, e.g., (u_j, u_i) is an edge with a direction from u_j to u_i. For better understanding, **Figure 12** shows an example for constructing G_2 based on G_1. It can be observed that, if we ignore the directions of edges, G_1 will be a subgraph of the connected graph G_2. It can be further inferred that, during the generation, each edge in E_2 can be associated with an index, representing the time of inserting the edge, e.g., an index value of 1 is assigned to (u_1, u_6) since this edge is the first one added to E_2. G_2 will be released in the social network by producing a sequence of social interactions. Each interaction is corresponding to an edge in E_2. The social interactions are orderly produced according to the index value of each edge, which means that an interaction with a smaller index will be preferentially produced. The social interactions are dependent of the social network platform, e.g., "forwarding", "commenting", and "liking" another user's tweet. A requirement is that, for each edge (u_x, u_y) in E_2, the interaction should be produced by the starting node u_x, e.g., in **Figure 12**, by using "forwarding" as the interaction, u_5 is required to "forward" the tweet of u_4 to represent the directed edge (u_5, u_4). The advantage is that, by taking into account the producer of the interaction, one can identify the direction of the corresponding edge, which will be beneficial to the construction of G_2.

The data receiver has the ability to perfectly reconstruct G_2 from observations. Let $(a_i, b_i, c_i), 1 \leq i \leq M_e = |E_2|$, denote all orderly collected interactions. (a_i, b_i, c_i) means the user a_i produces an interaction c_i to another user b_i. The following steps [13] will be then performed to reconstruct G_2.

Step 1) Set $i = j = 1$ and initialize $V_2 = \{u_0\}$ and $E_2 = \emptyset$.

Step 2) Insert a new node u_j to V_2 and map a_i to u_j if a_i has not been previously mapped to a node in V_2. Map b_i to u_0 if b_i has not been previously mapped to a node in V_2. Set $j = j + 1$ if u_j is in V_2.

Step 3) Insert a new edge (u_x, u_y) to E_2, where u_x and u_y are the mapped nodes of a_i and b_i. Mark a_i and b_i as "*processed*". Set $i = i + 1$, and go to Step 2) if $i < M_e$.

Step 4) Replace u_0 with u_{n+1} to build $G_2(V_2, E_2)$, and terminate the procedure.

After constructing G_2, by removing v_{n+1} and the involved edges from G_2, G_1 can be obtained by further eliminating the directions of all edges. With G_1, secret data **m** can be perfectly extracted. In this way, graph steganography is finally realized.

Remark: Graph steganography exploits social behaviors for conveying secrets. The graph steganographic scheme introduced in this subsection enables us to send $\binom{n}{2}$ secret bits to the receiver, which can be further improved if more interactions are performed among users. In terms of security, directly releasing G_2 in the social network may allow an attacker to reveal the existence of steganography since G_2 is dependent of other social behaviors. In other words, G_2 is an isolated graph, i.e., if an attacker finds a node in G_2, by applying DFS, he can further reconstruct G_2. To this end, by adding edges connecting nodes in G_2 and outside nodes (i.e., innocent users that does not join steganography), the above problem can be solved [13].

3.4 Other graph strategies in information hiding

In addition to the aforementioned graph models, graph theory can be also used for other purposes in information hiding. The reason is that graph theory enables us to model various types of relations and processes.

Content Prediction and Selection: In information hiding, the data hider has to select the suitable cover elements out from a cover for data embedding. We can use DFS or BFS (for a built graph) to deal with this problem. For example, in reversible image watermarking, it is expected to use as many smooth pixels as possible since smooth pixels can provide better rate-distortion performance. For a smooth pixel, it is very likely that its adjacent pixel is also smooth. By modeling pixels as graph nodes, we can use an edge to connect adjacent smooth pixels. Thus, selecting smooth pixels is equivalent to determining connected components. For instance, in **Figure 6(a)**, the pixels in the cross set are used for data embedding. The pixels in the dot set are unchanged. Each pixel in the cross set can be predicted by the adjacent pixels in the dot set, e.g., using the average value of adjacent pixels for prediction. We can build such a graph that the node set consists of all pixels in the cross set. For any two pixels in the cross set, they are adjacent to each other if their Manhattan distance is lower than a threshold, e.g., 2. Thus, for any two adjacent pixels, if the difference between their prediction values is lower than a given threshold, the corresponding nodes can be connected by an edge. In this way, we can use DFS or BFS to compute the maximum connected component of G and only use the corresponding pixels for carrying secret data. It relies on the assumption that two adjacent pixels having the close prediction values are equally smooth [5].

Information Hiding in Graphs: Unlike media objects that are organized into a formatted repository whose elements can be effectively processed, increasing data are captured as graphs such as social networks, deep learning models. These data have very high commercial value, e.g., social network data can be used for mining valuable information of users so as to improving the quality of service, deep neural network models have achieved great success in visual computing. How to protect the ownership of these graph data has been an important task. This requires us to hiding information into graph data without impairing the value of the graph data. One can extend conventional media based algorithms to graph data, e.g., [23]. On the other hand, similar to graph steganography in social networks, secret data can be first translated as graphs and then embedded into the host graph data, e.g., [24].

4. Conclusion and discussion

This chapter has introduced advanced graph models applicable to information hiding. In detail, we first model the data embedding operation in a graph, in which all cover elements are mapped to nodes and the modification relationship between cover elements are mapped to edges between the corresponding nodes. In this way, by applying the graph coloring technique to nodes, each node can be mapped to a bit-stream with an indefinite length, which allows a data hider to modify the cover elements according to the mapped bit-streams. Since the mapped bit-streams have fully exploited the modification relationship, a high data-embedding capacity can be achieved. We also model the parameter optimization task of HS based reversible watermarking as a weighted graph matching problem. In the built bipartite graph, the nodes are corresponding to the PEH bins to be shifted, and the edges show all the possible shifting operations. The weight of an edge equals the distortion caused by shifting one node in the edge to the other. By finding the MWMM in the built weighted bipartite graph, we can identify the optimal shifting solution for a fixed peak-set. In addition, we have also introduced graph steganography, which uses a graph to represent secret data and sends the graph via a social network platform by producing a sequence of ordered interactions such as "liking" and "forwarding". In summary, all the above graph models first use nodes to denote cover elements and use edges to represent somehow relationship between nodes. Then, the target problem to be addressed is modeled as a graph optimization problem. By solving the graph optimization problem, the optimal or near-optimal information hiding strategies can be determined, accordingly providing superior information hiding performance. We have also discussed other potential graph strategies that can be used in information hiding. Since graph theory enables us to model various types of relations and processes in information hiding, it is believed that, graph models and methods will play an increasingly important role in information hiding.

Acknowledgements

It was supported by the National Natural Science Foundation of China under Grant No. 61902235, the Shanghai "Chen Guang" Project under Grant No. 19CG46.

Conflict of interest

No conflict of interest exists in this chapter, and it is approved by all authors for publication.

DOI: http://dx.doi.org/10.5772/intechopen.98592

IntechOpen

Author details

Hanzhou Wu
School of Communication and Information Engineering, Shanghai University, Shanghai, China

*Address all correspondence to: h.wu.phd@ieee.org

References

[1] Petitcolas F, Anderson R, Kuhn M. Information hiding – A survey. Proceedings of the IEEE, 1999; 87(7): 1062-1078. DOI: 10.1109/5.771065

[2] Cox I, Miller M, Bloom J, Fridrich J, Kalker T. Digital watermarking and steganography. 2nd Ed. Morgan Kaufmann; 2007. 624 p. DOI: 10.1016/B978-0-12-372585-1.X5001-3

[3] Fridrich J. Steganography in digital media: principles, algorithms, and applications. 1st Ed. Cambridge University Press; 2009. 466 p. DOI: 10.1017/CBO9781139192903

[4] Wu H, Wang H, Zhao H, Yu X. Multi-layer assignment steganography using graph-theoretic approach. Multimedia Tools and Applications, 2015; 74(18):8171-8196. DOI: 10.1007/s11042-014-2050-y

[5] Wu H. Patch-level selection and breadth-first prediction strategy for reversible data hiding. In: Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'2020); 4–8 May 2020; Barcelona, Spain. New York: IEEE; 2020, p. 2837-2841

[6] Wu H, Shi Y, Wang H, Zhou L. Separable reversible data hiding for encrypted palette images with color partitioning and flipping verification. IEEE Transactions on Circuits and Systems for Video Technology, 2017; 27 (8): 1620-1631. DOI: 10.1109/TCSVT.2016.2556585

[7] Liu Q, Wu H, Zhang X. Adaptive video data hiding with low bit-rate growth based on texture selection and ternary syndrome-trellis coding. Multimedia Tools and Applications, 2020; 79(43): 32935-32955. DOI: 10.1007/s11042-020-09613-y

[8] Chen Y, Wang H, Wu H, Wu Z, Li T, Malik A. Adaptive video data hiding through cost assignment and STCs. IEEE Transactions on Dependable and Secure Computing, 2019; Early Access, DOI: 10.1109/TDSC.2019.2932983

[9] Chen Y, Wang H, Tang X, Liu Y, Wu H, Chen Y. A novel two-dimensional reversible data hiding method with high embedding capacity in H.264/advanced video coding. International Journal of Distributed Sensor Networks, 2020; 16(3): 1550147720911001, DOI: 10.1177/1550147720911001

[10] Chen Y, Wang H, Choo K, He P, Salcic Z, Kaafar M, Zhang X. DDCA: A distortion drift-based cost assignment method for adaptive video steganography in the transform domain. IEEE Transactions on Dependable and Secure Computing, 2021; Early Access, DOI: 10.1109/TDSC.2021.3058134

[11] Xue Y, Wu K, Wang Y, Chen Y, Zhong P, Wen J. Robust speech steganography using differential SVD. IEEE Access, 2019; 7: 153724-153733. DOI: 10.1109/ACCESS.2019.2948946

[12] Yang Z, Guo X, Chen Z, Huang Y, Zhang Y. RNN-Stega: Linguistic steganography based on recurrent neural networks. IEEE Transactions on Information Forensics and Security, 2019; 14(5): 1280-1295. DOI: 10.1109/TIFS.2018.2871746

[13] Wu H, Wang W, Dong J, Wang H. New graph-theoretic approach to social steganography. IS&T Electronic Imaging, Media Watermarking, Security, and Forensics, 14-17 January 2019; San Francisco, CA, USA. 2019. p. 539-1-539-7(7)

[14] Wu H, Zhou L, Li J, Zhang X. Securing graph steganography over

social networks via interaction remapping. International Conference on Artificial Intelligence and Security, 17–20 July 2020; Hohhot, China; 2020. p. 303-312

[15] Wu H, Wang H, Shi Y. PPE-based reversible data hiding. Proceedings of ACM Workshop on Information Hiding and Multimedia Security (IH& MMSec'16); 20–22 June 2016; Vigo, Spain. 2016. p. 187-188

[16] Wu H, Wang H, Shi Y. Dynamic content selection-and-prediction framework applied to reversible data hiding. IEEE International Workshop on Information Forensics and Security (WIFS'16), 4-7 December 2016; Abu Dhabi, UAE; 2016. p. 1-6

[17] Sachnev V, Kim H, Nam J, Suresh S, Shi Y. Reversible watermarking algorithm using sorting and prediction. IEEE Transactions on Circuits and Systems for Video Technology, 2009; 19 (7): 989-999. DOI: 10.1109/ TCSVT.2009.2020257

[18] Hong W, Chen T, Shiu C. Reversible data hiding for high quality images using modification of prediction errors. Journal of Systems and Software, 2009; 82(11): 1833-1842. DOI: 10.1016/j. jss.2009.05.051

[19] Hsu F, Wu M, Wang S. Reversible data hiding using side-match predictions on steganographic images. Multimedia Tools and Applications, 2013; 67(3): 571-591. DOI: 10.1007/ s11042-012-1047-7

[20] Cormen T, Stein C, Rivest R, Leiserson C. Introduction to algorithms. 3rd Ed. The MIT Press; 2009. 1292 p. ISBN: 9780262033848

[21] Ni Z, Shi Y, Ansari N, Su W. Reversible data hiding. IEEE Transactions on Circuits and Systems for Video Technology, 2006; 16(3): 354-362. DOI: 10.1109/ TCSVT.2006.869964

[22] Wu H, Zhang X. Reducing invertible embedding distortion using graph matching model. IS&T Electronic Imaging, Media Watermarking, Security and Forensics, 26-30 January 2020; San Francisco, CA, USA; 2020. p. 21-1-21-10(10)

[23] Al-Khafaji H, Abhayaratne C. Graph spectral domain blind watermarking. In: Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'2019); 12–17 May 2019; Brighton, United Kingdom. New York: IEEE; 2019, p. 2492-2496

[24] Zhao X, Liu Q, Zheng H, Zhao B. Towards graph watermarks. In: Proceedings of ACM on Conference on Online Social Networks (COSN'15); 2–3 November 2015; Palo Alto, California, USA. New York: ACM; 2015, p. 101-112