New Graph-Theoretic Approach to Social Steganography

Hanzhou Wu[†*]***, Wei Wang***[†]***, Jing Dong***[†]***, Hongxia Wang***[‡]
[†]***Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China***
[‡]***College of Cybersecurity, Sichuan University, Chengdu 610065, China***
[*]***h.wu.phd@ieee.org***

Abstract

In this paper, we introduce a new and novel graph-theoretic steganographic approach applicable to online social networking services (SNSs). The proposed approach translates a secret message to be embedded as an undirected graph called message-graph, the structural information of which is concealed within a new directed graph. The new directed graph is released in a SNS platform by producing a sequence of ordered multiple-user interaction events. To secure communication, we propose to split a specific vertex to multiple copies and insert new vertices and edges to the directed graph. A receiver is able to reconstruct the directed graph from observations. Both the message-graph and the secret message can be orderly retrieved without error. It is probably the first work deeply focusing on the practical design of interaction based steganography using graph-theoretic approach.

Introduction

People are experiencing the many advantages and convenience of the Internet technologies and services, among which the social networking service (SNS) has become more and more popular nowadays due to its openness, interaction and timeliness [1, 2]. A SNS platform such as Facebook[1], Twitter[2], and Sina Weibo[3], is an Internet-based social application that allows people to build social networks, relationships or links with other people who are willing to share similar interests and daily-life activities by releasing posts involving multimedia data. Moreover, a SNS platform enables people to easily comment, forward, and vote others' sharing, which has greatly enriched our spiritual life.

The variety of stand-alone and built-in SNSs also provide us new access to realizing *steganography* [3]. Typically, steganography aims to hide a secret message in a digital *cover* such as image by slightly modifying the noise-like component of the cover. The resulting object called *stego* will not introduce any noticeable artifacts and will be sent to a legal receiver who can perfectly reconstruct the embedded data according to the shared key. As a means to secure communication, different from cryptography, steganography even conceals the presence of communication.

A number of media based steganographic algorithms have been introduced in past years such as F5 [4], MB [5], StegHide [6], LSBM [7], MME [8], YASS [9], HUGO [10], UNIWARD [11], UED [12], WOW [13], HILL [14], and so on. By analyzing these state-of-the-arts, two commonly-used principles for designing steganographic systems can be obtained [15]. One is model-preserving steganography in which the encoder aims to preserve the selected model of the cover source during data embedding, e.g., [5], [6]. The other one treats steganography as a *rate-distortion* optimization task [16]. For a required payload, it is expected to reduce the introduced distortion as low as possible, believing that, a minimized distortion always corresponds to a high security level where a well-designed distortion measure is required to resist against steganalysis approaches [17, 18, 19, 20].

Massive media data with different characteristics are widely distributed over SNSs, some of which also provide personalized media-editing APIs for users. It allows the steganographic communication to be easily covered by different normal operations or behaviors, resulting in many false positives from the viewpoint of steganalysis. We believe that a part of conventional advanced media-based steganographic algorithms can be applied to SNSs, which, by making adjustments accordingly, should overcome the possible lossy operations of SNS platforms such as compression. Namely, *robustness* should be considered when to use steganography over SNSs, which is not the main interest of this paper.

Steganography within a media file permits hiding a limited amount of data per one file, and, from an attacker's view, the modified file may be accessible for steganalysis experts [21]. Instead of using a media file, steganography can be also realized by social interactions. Interaction-level steganography hides a message by creating an abstract *graph pattern* [22] based on user interactions in SNS platforms. The existence of communication can be easily concealed and removed by the steganographer, e.g., one can delete his/her own comments, cancel "Likes" or other erasable actions. If all abnormal user interactions are not captured, there is little information for steganalysis experts to analyze. As a result, steganography by interactions is more difficult to detect and even eliminate for steganalysis experts. This motivates us to study interaction based steganography (defined as *social steganography*).

There has no doubt that the essence of steganography is communication, which modulates a state X to another state Y by a specific rule and a shared key. For steganography, we usually have $Y = \mathbf{Emb}(X, M, K)$ and $M = \mathbf{Ext}(Y, K)$ where K is the secret key and M for the secret message. In media steganography, both X and Y are media files such as digital images. Different from media steganography, social steganography uses user interactions (or say behaviors). In this case, one can consider both X and Y as a social network, which can be represented by a graph that consists of vertices, edges and other necessary information. Generally, the vertices represent users, and the edges reveal the social interactions between users. Even though X and Y can represent the social network corresponding to a whole SNS platform, there has no such need since we usually only focus on such users involving the present social steganography. Unlike media steganography, social steganography corresponds to a *time-varying* system since

[1]https://www.facebook.com/
[2]https://twitter.com/
[3]https://weibo.com/

SNS platforms are always updating no matter whether steganography is taking place or not.

Assuming that, we have an initialized social network S_0 (ignoring its detailed form), we can record a sequence of social networks $\{S_0, S_1, ..., S_N\}$ as time goes by. One may simply think that S_t is corresponding to time $t \in [0, N]$. A social steganographic scheme $\mathscr{A}$ requires us to pick two indexes $0 \leq a < b \leq N$ out such that $X = S_a$ and $Y = S_b$. The data embedding procedure for $\mathscr{A}$ corresponds to a sequence of *key-controlled* user interactions within time $[a, b-1]$, which enables S_a to become S_b. For a receiver, according to the shared key, he should be able to identify interaction events and reconstruct an *abstract graph* from these observations. The structural information of the abstract graph reveals the true secret message. Thus, for social steganography, we have $M = \mathbf{Ext}(S_a, S_{a+1}, ..., S_b, K)$ since interactions are identified from difference between adjacent networks. Notice that, due to the openness, a desired receiver should be always able to observe the necessary information of events involving steganography.

Based on the above perspective, we will present a graph theory based social steganographic approach below. The structure is organized as follows. We first present the proposed basic steganographic framework. Then, we analyze the basic framework from aspects of capacity, complexity and security. Thereafter, we introduce graph-modification techniques for securing steganographic communication. We also show an example for practical use. Finally, we conclude this paper and provide some new perspectives and further research directions.

Basic Social Steganographic Framework

We will use graph theory. A graph $G(V, E)$ consists of V and E, where $V = \{v_1, v_2, ..., v_n\}$ is the vertex set (or node set), and $E = \{e_1, e_2, ..., e_m\}$ is the edge set. $\forall e \in E$, we can write $e = (u, v)$ where $u \in V$ and $v \in V$. $G(V, E)$ is either *directed* or *undirected*. A directed graph is a graph that the edges have a direction associated with them. An undirected graph is a graph that all edges have no orientation, i.e., $(u, v) \in E$ is equivalent to $(v, u) \in E$. A subgraph of a graph $G(V, E)$ is another graph $G'(V', E')$ formed from a subset of the vertices and edges of $G(V, E)$, namely, $V' \subset V$ and $E' \subset E$. $\forall e' = (u', v') \in E'$, we have $u' \in V', v' \in V'$.

The basic framework mainly includes message-graph generation, message-graph embedding and message-graph reconstruction. The first step translates a message as an undirected graph. The second step is to construct a new directed graph that conceals the topological information of the message-graph. The new directed graph can be then released as a form of multiple-user interaction-events in a SNS platform. The third step enables a receiver to reconstruct the message-graph from observations, which allows the secret message to be fully recovered.

Message-Graph Generation

Let $\mathbf{m} = [m_1 m_2 m_3 ... m_l] \subset \{0, 1\}^l$ be a secret message. We convert $\mathbf{m}$ to an undirected message-graph $G_0(V_0, E_0)$, ensuring that, given $G_0(V_0, E_0)$, one can reconstruct $\mathbf{m}$ without error. The length of $\mathbf{m}$, i.e., l, can be "self-contained". Namely, given any bit-string started from $\mathbf{m}$, one can split $\mathbf{m}$ out. This can be done by inserting auxiliary data to $\mathbf{m}$.

Given n vertices indexed from 1 to n, $2^{\binom{n}{2}}$ different graphs can be constructed. Each graph can represent a bit-string with a length of $\binom{n}{2}$. When the length of $\mathbf{m}$ is smaller than $\binom{n}{2}$, one can

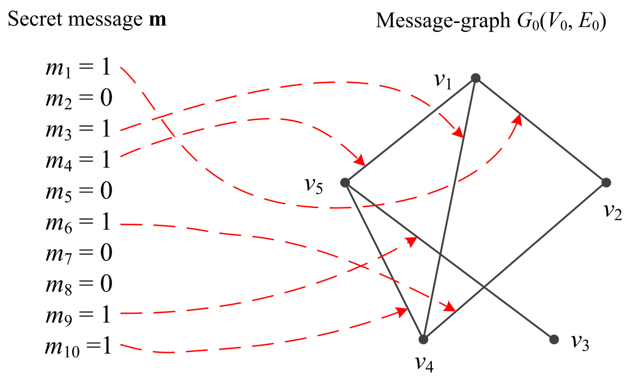


Figure 1. *An example of translating a bit-string as an undirected graph.*

always append "0"s to $\mathbf{m}$ to constitute a bit-string with a length exactly equal to $\binom{n}{2}$. For simplicity, we assume that $l = \binom{n}{2}$, i.e., $n = (\sqrt{8l+1} + 1)/2$. It is straightforward to construct $G_0(V_0, E_0)$ for $\mathbf{m}$. In detail, we first initialize $V_0 = \{v_1, v_2, ..., v_n\}$ and $E_0 = \emptyset$. Then, for every $m_z \in \mathbf{m}$, if $m_z = 0$, no operation is performed. Otherwise, we update E_0 as $E_0 = E_0 \cup \{(v_x, v_y)\}$, where x and y are determined as:

$$x = \min \{ j \mid 1 \leq j \leq n-1, \sum_{i=1}^{j} (n-i) \geq z \}, \tag{1}$$

and

$$y = x + z - \sum_{i=1}^{x-1} (n-i). \tag{2}$$

Fig. 1 shows an example, where $\mathbf{m}$ = [1011010011], $l = 10$, $n = 5$. Since $m_1 = 1$, we find $x = 1$ and $y = 2$, indicating that there is an edge between v_1 and v_2. As $m_9 = 1$, we find $x = 3$ and $y = 5$, meaning that, (v_3, v_5) should be inserted. For those bits with value "0", though an index-pair can be determined with Eqs. (1, 2), no edge will be inserted. Given the message-graph, one can reconstruct $\mathbf{m}$ as well. In detail, for each $1 \leq z \leq l$, we can determine two indexes x and y with Eqs. (1, 2). If there exists (v_x, v_y), then we have $m_z = 1$, otherwise, we specify $m_z = 0$. We still take Fig. 1 for explanation. Suppose that $z = 8$, we can obtain $x = 3$ and $y = 4$. As (v_3, v_4) does not exist, we therefore have $m_8 = 0$. Notice that, the number of edges always equals the Hamming weight of $\mathbf{m}$, i.e., $m = |E_0| = \sum_{i=1}^{l} m_i$.

Message-Graph Embedding

We will construct a new directed graph $G_1(V_1, E_1)$ concealing $G_0(V_0, E_0)$. If we ignore the directions of edges in G_1, G_0 will be a subgraph of G_1. We denote V_0 and E_0 by $V_0 = \{v_1, v_2, ..., v_n\}$ and $E_0 = \{e_1, e_2, ..., e_m\}$. And, we write $V_1 = \{v'_1, v'_2, ..., v'_{n'}\}$ and $E_1 = \{e'_1, e'_2, ..., e'_{m'}\}$, where $n' = n + 1$ and $m' > m$. It is also assumed that $v_1 = v'_1$, $v_2 = v'_2$, ..., $v_n = v'_n$, i.e., $V_0 \subset V_1$ and $V_1 \setminus V_0 = \{v'_{n'}\} = \{v'_{n+1}\}$.

Assuming that, both $G_0(V_0, E_0)$ and V_1 are known, we are to determine E_1. The edges in E_1 are associated with a direction. An edge $(v'_i, v'_j) \in E_1$ indicates a direction from v'_i to v'_j. And, (v'_j, v'_i) corresponds to a direction from v'_j to v'_i. We will generate E_1 by *orderly* processing $\{v_1, v_2, ..., v_n\}$. In detail, we first initialize

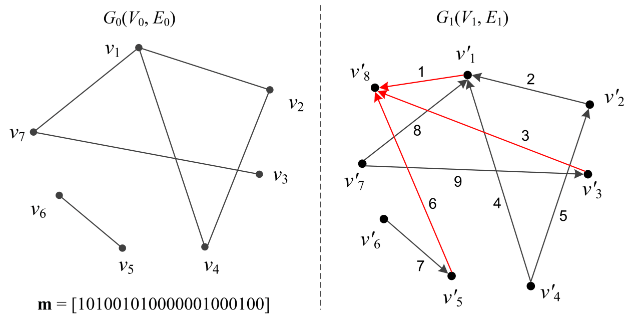


Figure 2. *An example of constructing $G_1(V_1,E_1)$ based on $G_0(V_0,E_0)$.*

$E_1 = \emptyset$, and for each $v_i \in V_0$ $(1 \leq i \leq n)$, we perform one of the following two operations to update E_1.

OP 1. If $\exists j < i$, $(v_j, v_i) \in E_0$, then, $\forall j < i$, if $(v_j, v_i) \in E_0$, we insert (v'_i, v'_j) to E_1.

OP 2. If $\nexists j < i$, $(v_j, v_i) \in E_0$, we insert (v'_i, v'_{n+1}) to E_1.

In this way, we can generate $G_1(V_1,E_1)$. Fig. 2 shows an example, where $n = 7, m = 6, n' = 8$. With G_0, we first process v_1 with **OP 2**, i.e., (v'_1, v'_8) is inserted to E_1. Then, we process v_2 with **OP 1**, resulting in the insertion of (v'_2, v'_1). **OP 2** will be performed for v_3, which adds (v'_3, v'_8) to E_1. The subsequent procedure is straightforward. Fig. 2 has shown the final G_1. If we ignore the directions of E_1, G_0 will be a subgraph of G_1. In addition, it is observed from Fig. 2 that, each directed edge in E_1 can be associated with an index value that represents the time of inserting it into E_1. For example, (v'_1, v'_8) is associated with "1" since it is the first edge adding to E_1. (v'_7, v'_3) has a value of "9", meaning that it is the 9-th edge adding to E_1.

G_1 will be released in a selected SNS platform by creating a sequence of user interactions, which are orderly produced according to the associated index-values of the edges in E_1. Namely, the interaction corresponding to an edge with a smaller index will happen prior to that with a larger index. The interaction operations are not unique and actually depend on the used SNS platform. For example, a user may produce the interaction event by forwarding, commenting or "liking" another user's micro-blog. This indicates that, it is relatively free for us to choose the types of interaction events.

A necessary requirement is that, for each directed edge in E_1, the corresponding interaction event is produced by the starting vertex of the edge. For any directed edge (v'_i, v'_j), the starting vertex is v'_i. For example, for (v'_1, v'_8) shown in Fig. 2, the interaction event will be produced by v'_1. The goal of such requirement is to ensure that, a legal receiver can identify the corresponding vertex-index of a user account. Notice that, when to release G_1 in a SNS platform, each vertex in G_1 corresponds to a user account.

Message-Graph Reconstruction

A legal receiver reconstructs G_1 before reconstructing G_0. A steganographer should have previously created a set U, where $|U| \geq n'$. U is only shared between the steganographer and the receiver. In other words, the steganographer actually selects n' users in U to constitute V_1. The interactions between n' users reveal E_1. Since the interactions are orderly produced, the receiver can construct a directed graph $G_2(V_2,E_2)$ from observations. Let (a_i, b_i, c_i) $(1 \leq i \leq m')$ denote the orderly observed interactions. Here, a_i and b_i represent two users. c_i is the interaction operation. a_i is the producer of c_i. In order to construct $G_2(V_2,E_2)$, we perform the following steps:

Step 1. Set $V_2 = \emptyset$, $E_2 = \emptyset$, $i = j = 1$, perform **Step 2**.

Step 2. If $i > m'$, perform **Step 5**; otherwise, go to **Step 3**.

Step 3. If a_i has not been processed previously, we then map a_i to a vertex denoted by v'_j and set $j = j+1$; otherwise, a_i must have been previously mapped to a vertex. If b_i has not been processed previously, we then map b_i to a vertex denoted by $v'_{n'}$; otherwise, b_i must have been previously mapped to a vertex. Notice that, n' can be easily obtained by determining the total number of different users from observations. We then perform **Step 4**.

Step 4. Let v'_x and v'_y be the mapped vertices of a_i and b_i, respectively. We update $V_2 = V_2 \cup \{v'_x, v'_y\}$, and $E_2 = E_2 \cup \{(v'_x, v'_y)\}$. Mark the user accounts a_i and b_i as *processed*. Set $i = i+1$ and perform **Step 2**. It does not matter if an account has been previously marked as *processed*.

Step 5. Collect $G_2(V_2,E_2)$ and terminate the procedure.

We write $V_2 = V_1$ and $E_2 = E_1$, i.e., G_2 is equivalent to G_1. We remove $v'_{n'}$ and all edges involving $v'_{n'}$ from G_2. By ignoring the directions of edges, we can obtain G_0, from which we can extract the secret message. Notice that, to collect (a_i, b_i, c_i) $(1 \leq i \leq m')$, users in $U \setminus V_1$ should not take actions at the same time.

Capacity, Complexity and Security

The embedding capacity for the basic framework depends on the size of $G_0(V_0,E_0)$. Given n, a total of $\binom{n}{2}$ bits can be conveyed in a SNS platform, resulting in an embedding rate of $0.5 \cdot (n-1)$ bits per vertex (bpv), which is significantly larger than most traditional media based steganography from the viewpoint of cover-element utilization.

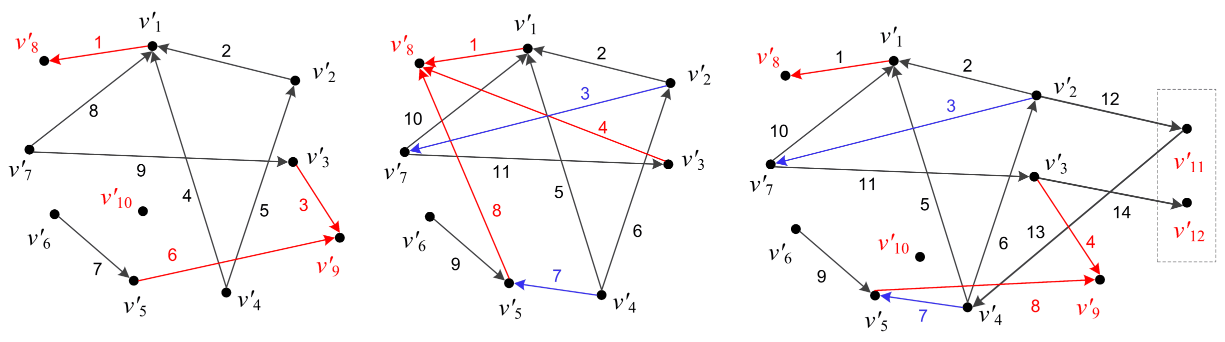
The computational complexity to construct G_0 is $O(l)$. We have $m \approx l/2$ if $\mathbf{m}$ has been encrypted. When to construct G_1, v'_{n+1} will be connected by at most n edges. Thus, the computational complexity of constructing G_1 is $O(m+n)$. The complexity to obtain G_0 from G_1 will be $O(n)$ since we only need remove $v'_{n'}$ and involved edges. Obviously, to extract $\mathbf{m}$, we have to check the existence of each edge, resulting in a complexity of $O(\frac{1}{2}n^2 - \frac{1}{2}n)$.

$\mathbf{m}$ should be encrypted prior to translation. A steganalysis expert may detect the existence of G_1 and even reconstruct G_1. Simply releasing G_1 probably allows an attacker to successfully reconstruct G_0, G_1 and $\mathbf{m}$. The reason is that, G_1 corresponds to a *connected* graph if we ignore the orientations of edges. If the steganalysis expert has found a vertex in G_1, he can collect all vertices having a path to the found vertex. All collected vertices including the found vertex constitute V_1. The interactions reveal E_1. Accordingly, the expert could reconstruct G_1, G_0 and $\mathbf{m}$. Here, it is assumed that, the attacker has collected $(a_i, b_i, c_i), i \in [1, m']$, which may be a strong assumption.

As mentioned above, $m = \sum_{i=1}^{l} m_i \approx \frac{l}{2}$, which may cause suspicion of an analyzer. Therefore, to secure the steganographic communication, we may not directly release G_1 in a SNS platform, but rather modify G_1 in a way that an attacker will not find G_1 easily. Meanwhile, the release of modified G_1 should guarantee that a receiver can extract G_0 and $\mathbf{m}$. We deal with it below.

Secure and Erasable Graph Modifications

We propose to combine three methods to modify G_1 to deal with the security problem mentioned above, i.e.,



(a) Modified $G_1(V_1, E_1)$ with **M 1** (b) Modified $G_1(V_1, E_1)$ with **M 2** (c) Modified $G_1(V_1, E_1)$ with **M 1**, **M 2** and **M 3**

Figure 3. *Examples of applying **M 1**, **M 2**, and **M 3** to G_1: (a) v'_8 is split to three copies (v'_8, v'_9, v'_{10}), (b) two edges (v'_2, v'_7) and (v'_4, v'_5) are inserted, (c) two vertices (v'_{11}, v'_{12}) and involved edges are inserted.*

M 1. Split $v'_{n'}$ to multiple vertices.

M 2. Insert *key-controlled* directed edges to G_1.

M 3. Add *invalid* vertices and edges.

M 1 makes G_1 relatively sparse, i.e., the ratio between the number of existing edges in G_1 and the maximum possible number of edges is reduced. Such sparse operation reduces the degree of the original $v'_{n'}$. It also increases the difficulty to determine the raw G_1. When to split $v'_{n'}$, the head vertices of all the involved directed edges are randomly re-selected from the newly inserted vertices, which can be done by modifying **OP 2** as follows.

Modified OP 2. Let V_s denote the set of split vertices of v'_{n+1}. If $\nexists j < i$, $(v_j, v_i) \in E_0$, we insert (v'_i, v'_r) to E_1, where $v'_r \in V_s$ is randomly selected according to a key (notice that, the receiver does not know this key).

We take Fig. 3 (a) (based on Fig. 2) for better explanation. We split v'_8 to three new vertices $\{v'_8, v'_9, v'_{10}\}$, and then modify the edges (v'_1, v'_8), (v'_3, v'_8), (v'_5, v'_8) shown in Fig. 2 as (v'_1, v'_8), (v'_3, v'_9), (v'_5, v'_9) shown in Fig. 3 (a), respectively. It is observed that, though we have added a new vertex v'_{10}, no edge is assigned to v'_{10}, which does not affect the reconstruction of G_0 as $v'_{10} \notin G_0$.

It is not explicit currently that whether the sparsity due to the split operation exposes significant steganalysis features or not. It inspires us to insert *key-controlled* directed edges to G_1, i.e., **M 2**, to control the sparsity. Previously, we construct G_1 by orderly processing vertices in V_0. During the processing, either **OP 1** or **OP 2** is performed. It is observed that, **OP 1** always links a vertex to another one that has a smaller index value, e.g., as shown in Fig. 2, the head vertices of all black edges have a smaller index value than the corresponding tail vertices. This property enables a receiver to identify the corresponding vertex-indexes of user accounts from observations and finally recover G_0 without any error. It can be therefore inferred that, a necessary requirement of using **M 2** is that a receiver should be able to remove the inserted edges so as to recover G_0 without error. This is can be done by only simply modifying **OP 1** as follows:

Modified OP 1. If $\exists j < i$, $(v_j, v_i) \in E_0$, then, $\forall j < i$, if $(v_j, v_i) \in E_0$, we insert (v'_i, v'_j) to E_1. Thereafter, for each $j \in (i, n]$, we use a seed to generate a bit value $w_k \in \{0, 1\}$ (it may be not evenly distributed), we insert (v'_i, v'_j) to E_1 if $w_k = 1$. (The receiver does not know the seed and the bit-generation algorithm.)

Therefore, by replacing **OP 1** with **Modified OP 1**, we can achieve **M 2**. The correctness relies on the property that the indexes of head vertices of all edges (except for those involve $v'_{n'}$) in the original G_1 are less than the tail vertices. In this way, at the decoding side, one can easily identify the newly inserted edges due to **Modified OP 1** as the newly inserted edges meet that, the indexes of head vertices of all edges in the modified G_1 are larger than the tail vertices. Fig. 3 (b) shows an example using **M 2**. It is straightforward to reconstruct the original G_1 from the modified G_1 due to **M 2** based on above property. Notice that, when to use **M 2**, the message-graph reconstruction procedure is not the same as the basic framework. We show the corresponding reconstruction procedure in the end of this section.

One may further insert invalid vertices and edges to enhance security. This is quite different from **M 1** and **M 2**. It requires a steganographer to build a new vertex set R that has no intersection with U, i.e., $R \cap U = \emptyset$. However, unlike U, for the steganographer, there has no need to share R with a receiver since R does not carry useful information for data decoding. The goal of R is to enhance the difficulty of reconstructing G_0 for an attacker. It is easily inferred that, when G_1 is modified by **M 3**, the way to reconstruct the original G_1 from the modified G_1 is straightforward. In detail, with the modified G_1 due to **M 3**, the receiver can obtain the original G_1 by just removing those vertices not in U as well as the edges involving those removed vertices.

Accordingly, by combining the above three methods, we can construct a new graph which is more complex than the original G_1, e.g., Fig. 3 (c). A necessary requirement is to ensure that G_0 can be perfectly recovered, meaning that, the above graph modifications should be erasable. Let G_3 denote the resulting directed graph by applying **M 1**, **M 2** and **M 3**. With G_0, R and V_s, we perform below steps on G_0 to generate $G_3(V_3, E_3)$.

Step 1. For each vertex $v_i \in V_0$ $(1 \leq i \leq n)$, perform either **Modified OP 1** or **Modified OP 2** according to G_0. We can therefore generate a modified G_1 due to **M 1** and **M 2**.

Step 2. Randomly insert a certain number of edges (u, v) to the modified G_1, where $u \in V_1, v \in R$ or $u \in R, v \in V_1$. The resulting new graph will constitute G_3.

Once G_3 is released in a SNS platform, a receiver needs to reconstruct G_0. The reconstruction steps are:

Step 1. Set $V_0 = \emptyset$, $E_0 = \emptyset$, $i = j = 1$, perform **Step 2**.

Step 2. If $i > |E_3|$, perform **Step 5**; otherwise, we are to perform **Step 3**. Notice that, $|E_3|$ can be determined by the receiver.

Step 3. If $a_i \notin U$ or $b_i \notin U$ (note that, $V_s \subset U$), set $i = i + 1$

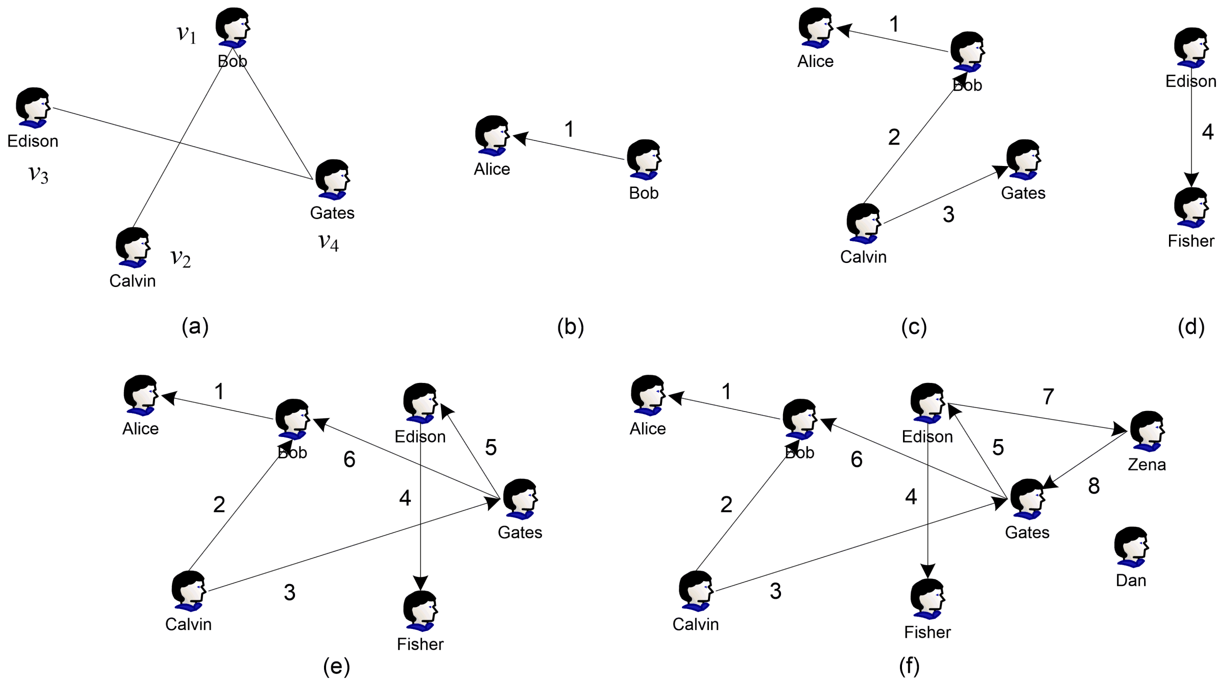


Figure 4. *A simple example for the proposed steganographic approach using* ***M 1****,* ***M 2*** *and* ***M 3****.*

and perform **Step 2**. Otherwise, if a_i has not been processed, we map a_i to v_j and set $j = j+1$. We then perform **Step 4**.

Step 4. Let v_x be the mapped vertex of a_i. We update $V_0 = V_0 \cup \{v_x\}$. We also update $E_0 = E_0 \cup \{(v_x, v_y)\}$ if b_i has been previously mapped to v_y and $y < x$. Mark a_i as *processed*. Set $i = i+1$ and perform **Step 2**. Notice that, when b_i has not been previously mapped to a vertex, we will not update E_0.

Step 5. Collect $G_0(V_0, E_0)$ and terminate the procedure.

Accordingly, we have successfully applied **M 1**, **M 2** and **M 3** to G_1 (or say G_0), and also recover G_0 without error.

Simple and Intuitive Example

We here show a simple example for practical use. We will not take the real SNS platforms such as Facebook and Twitter for explanation, but use notations to demonstrate the steganographic mechanism due to its simplicity and generality. Suppose that, we have $U = \{Alice, Bob, Calvin, Dan, Edison, Fisher, Gates\}$ as well as $\mathbf{m} = [101001]$. The steganographer first randomly chooses $\{Bob, Calvin, Edison, Gates\}$ as the vertices of G_0 and orderly maps them to $\{v_1, v_2, v_3, v_4\}$. Notice that, the mapping between the users and the vertices is not shared between the steganographer and a receiver, but rather is determined by the receiver from observations. Fig. 4 (a) shows the message-graph, which can be easily obtained by the message-graph generation algorithm.

We are to embed such graph pattern by user interactions. At first, *Bob* produces an interaction to *Alice*, as shown in Fig. 4 (b). It is pointed that, both *Alice* and *Fisher* are chosen to constitute V_s. Then, as shown in Fig. 4 (c), for *Calvin*, he first produces an interaction to *Bob* to carry a message bit, and, then produces an interaction to *Gates*, which is corresponding to **M 2**. Thereafter, *Edison* produces an event to *Fisher* since there has no edge between v_3 and $\{v_1, v_2\}$. Notice that, *Fisher* is randomly chosen from the split set $\{Alice, Fisher\}$. Going on, *Gates* generates two different events indexed by 5 and 6, respectively, as shown in Fig. 4 (e). Finally, two events $(7, Edison \mapsto Zena)$ and $(8, Zena \mapsto Gates)$ are produced, which correspond to **M 3**. Thus, we have embedded **m** into the social network, i.e., Fig. 4 (f).

At the receiver side, we should recover G_0 from Fig. 4 (f). We orderly process the interactions according to their indexes. Initially, we have $V_0 = \emptyset$ and $E_0 = \emptyset$. At first, for $Bob \mapsto Alice$, we map *Bob* to v_1 and update $V_0 = \{v_1\}$. Then, for $Calvin \mapsto Bob$, we map *Calvin* to v_2 and update $V_0 = \{v_1, v_2\}$, $E_0 = \{(v_1, v_2)\}$. We skip "$Calvin \mapsto Gates$" since *Gates* has not been mapped. For $Edison \mapsto Fisher$, we will map *Edison* to v_3 and update $V_0 = \{v_1, v_2, v_3\}$. Then, for $Gates \mapsto Edison$, we map *Gates* to v_4 and update $V_0 = \{v_1, v_2, v_3, v_4\}$, $E_0 = \{(v_1, v_2), (v_3, v_4)\}$. Thereafter, for $Gates \mapsto Bob$, we update $E_0 = \{(v_1, v_2), (v_3, v_4), (v_1, v_4)\}$. Since *Zena* does not belong to U, no operation will be performed for interactions 7 and 8. Accordingly, we have reconstructed G_0, where $V_0 = \{v_1, v_2, v_3, v_4\}$ and $E_0 = \{(v_1, v_2), (v_3, v_4), (v_1, v_4)\}$, from which we can determine **m** = [101001].

Remark 1. We perform **M 3** after G_0 has been concealed. Actually, **M 3** can be performed at any time since either the corresponding head vertices or the tail vertices are not in U, which can be easily identified and then skipped by the receiver.

Remark 2. Though the receiver knows U, he/she does not know V_0 initially, which can be determined from observations.

Remark 3. For the **Modified OP 1**, **Modified OP 2**, and **M 3**, the steganographer should hold some keys for producing user interactions. However, all these keys (having been mentioned above) are unknown to the receiver.

Remark 4. In Fig. 4, the edge $Gates \mapsto Edison$ has a smaller index than $Gates \mapsto Bob$. Actually, it is also allowed that $Gates \mapsto Bob$ has a smaller index than $Gates \mapsto Edison$.

Remark 5. In application scenarios, the usable types of user interactions could be predetermined by the sender and receiver.

Conclusion, Discussion and Future Works

The main idea of proposed work is to embed a graph pattern (carrying a secret payload) into a SNS platform by producing user interactions. Different from many traditional algorithms, a specific media file is not necessary (though the interactions may involve media data). To secure communication, we propose three kinds of graph modification techniques to make the graph to be released complex, which would significantly increase the reconstruction difficulty for a steganalysis expert. The data embedding capacity is proportional to the number of vertices of the message-graph. The proposed work is a general framework, which can be extended to steganography using other social behaviors.

Traditional media based steganographic algorithms focus on minimizing the impact due to data embedding, which requires us to well model the cover/stego. In other words, the state-of-the-arts actually pay heavily attention to the security of media content (to resist against content-aware steganalysis). For social steganography, more attentions should be paid to security of social behaviors (interactions) [23]. Indeed, the social behaviors may also involve media data. In this case, we probably use media-based steganographic algorithms as an assistant means. On the other hand, social steganography corresponds to a time-varying (network) system. From the viewpoint of social network, we should guarantee that the social steganographic activities do not cause suspicion by the network monitor. It means that, the steganographic activities should follow statistical characteristics of normal social activities.

From the point of graph theory, social steganography inserts a secret graph pattern (called *message-graph*) into another open social network (called *cover network*), resulting in a seemingly-normal social network (called *stego network*). For a steganalysis expert, two directions could be investigated. One is to reveal the existence of message-graph from a social network, for which *Markov analysis* and *complex network analysis* may be functional. Data mining in graphs [22, 24] can be desirable as well. Another one is to extract the hidden message-graph, which would be quite challenging. In addition, unlike media steganography, social steganography accepts multiple steganographers, i.e., a stego network may contain multiple different message-graphs. It probably leads us to new directions of steganography in the future.

Acknowledgment

It was partly supported by National Natural Science Foundation of China under grant No. U1536110, 61502496, U1536120, U1636201, U1736119 and 61772529, and the National Key Research and Development Program of China under the grant No. 2016YFB1001003. It was also partly supported by the Key Lab of Information Network Security and the Ministry of Public Security of China.

References

[1] D. M. Boyd, and N. B. Ellison. Social network sites: definition, history, and scholarship. *Journal of Computer-mediated Communication*, 13(1): 210-230, 2007.

[2] B. Chang, T. Xu, Q. Liu, and E. Chen. Study on information diffusion analysis in social networks and its applications. *International Journal of Automation and Computing*, 15(4): 377-401, 2018.

[3] J. Fridrich. Steganography in digital media: principles, algorithms, and applications. *Cambridge University Press*, 2009.

[4] A. Westfeld. F5: A steganographic algorithm. In: *Proc. International Workshop on Information Hiding*, pp. 289-302, 2001.

[5] P. Sallee. Model-based steganography. In: *Proc. International Workshop on Information Hiding*, pp. 154-167, 2003.

[6] S. Hetzl, and P. Mutzel. A graph-theoretic approach to steganography. In: *IFIP International Conference on Communications and Multimedia Security*, pp. 119-128, 2005.

[7] J. Mielikainen. LSB matching revisited. *IEEE Signal Processing Letters*, 13(5): 285-287, 2006.

[8] Y. Kim, Z. Duric, and D. Richards. Modified matrix encoding technique for minimal distortion steganography. In: *Proc. International Workshop on Information Hiding*, pp. 314-327, 2007.

[9] K. Solanki, A. Sarkar, and B. S. Manjunath. YASS: Yet another steganographic scheme that resists blind steganalysis. In: *Proc. International Workshop on Information Hiding*, pp. 16-31, 2007.

[10] T. Pevny, T. Filler, and P. Bas. Using high-dimensional image models to perform highly undetectable steganography. In: *Proc. International Workshop on Information Hiding*, pp. 161-177, 2010.

[11] V. Holub, J. Fridrich, and T. Denemark. Universal distortion function for steganography in an arbitrary domain. In: *EURASIP Journal on Information Security*, 2014:1-13, 2014.

[12] L. Guo, J. Ni, and Y. Shi. An efficient JPEG steganographic scheme using uniform embedding. In: *Proc. International Workshop on Information Forensics and Security*, pp. 169-174, 2012.

[13] V. Holub, and J. Fridrich. Designing steganographic distortion using directional filters. In: *Proc. IEEE International Workshop on Information Forensics and Security*, pp. 234-239, 2012.

[14] B. Li, M. Wang, J. Huang, and X. Li. A new cost function for spatial image steganography. In: *Proc. IEEE International Conference on Image Processing*, pp. 4206-4210, 2014.

[15] T. Filler, and J. Fridrich. Gibbs construction in steganography. *IEEE Transactions on Information Forensics and Security*, 5(4): 705-720, 2010.

[16] T. Filler, J. Judas, and J. Fridrich. Minimizing additive distortion in steganography using syndrome-trellis codes. *IEEE Transactions on Information Forensics and Security*, 6(3): 920-935, 2011.

[17] Y. Shi, C. Chen, and W. Chen. A markov process based approach to effective attacking JPEG steganography. In: *Proc. International Workshop on Information Hiding*, pp. 249-264, 2006.

[18] J. Fridrich, and J. Kodovsky. Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, 7(3): 868-882, 2011.

[19] G. Xu, H. Wu, and Y. Shi. Structural design of convolutional neural networks for steganalysis. *IEEE Signal Processing Letters*, 23(5): 708-712, 2016.

[20] G. Xu, H. Wu, and Y. Shi. Ensemble of CNNs for steganalysis: an empirical study. In: *Proc. ACM Workshop on Information Hiding and Multimedia Security*, pp. 103-107, 2016.

[21] W. Mazurczyk, and K. Szczypiorski. Trends in steganography. *Communications of the ACM*, 57(3): 86-95, 2014.

[22] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In: *Proc. ACM International Conference on World Wide Web*, pp. 181-190, 2007.

[23] H. Wu, W. Wang, J. Dong, Y. Xiong, and H. Wang. A graph-theoretic model to steganography on social networks. *Preprint arXiv:1712.03621*, Online Available, 2018.

[24] T. Washio, and H. Motoda. State of the art of graph-based data mining. *Proc. ACM SIGKDD Explorations Newsletter*, 5(1): 59-68, 2003.