Targeted Poisoning Attacks on Social Recommender Systems

Rui Hu*, Yuanxiong Guo†, Miao Pan‡, Yanmin Gong*
*Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249
†Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249
‡Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204
Email: rui.hu@my.utsa.edu, yuanxiong.guo@utsa.edu, mpan2@uh.edu, yanmin.gong@utsa.edu

***Abstract*—With the popularity of online social networks, social recommendations that rely on ones social connections to make personalized recommendations have become possible. This introduces vulnerabilities for an adversarial party to compromise the recommendations for users by utilizing their social connections. In this paper, we propose the targeted poisoning attack on the factorization-based social recommender system in which the attacker aims to promote an item to a group of target users by injecting fake ratings and social connections. We formulate the optimal poisoning attack as a bi-level program and develop an efficient algorithm to find the optimal attacking strategy. We then evaluate the proposed attacking strategy on real-world dataset and demonstrate that the social recommender system is sensitive to the targeted poisoning attack. We find that users in the social recommender system can be attacked even if they do not have direct social connections with the attacker.**

I. Introduction

The recommender system has become an imperative component of modern information and e-commerce applications, which is widely deployed by websites (e.g. Netflix, Spotify, Amazon and Youtube) to recommend relevant items (e.g., movies, musics, products and video) to users. The recommender system aims to locate the data that is most relevant to a user by utilizing the user's historical behavior to predict the user's possible future likes and interests. With the popularity of social networks, recommender systems can take advantage of rich social relationships to further improve the effectiveness of recommendations [1]–[4]. The intuition is that in our real life people tend to adopt behaviors exhibited by those they are interacting with (i.e. the people they have relationships with). For example, we always ask our friends for recommendations of movies, music or restaurants. The recommender systems integrated with social networks (known as social recommender systems) seek to improve the performance of recommendations for a user by learning from the preference of the user's friends. In a social recommender system, each user' preference has direct or indirect influences on the preferences of other users in the social network.

However, recommender systems are susceptible to a risk of being maliciously attacked as they become increasingly popular in the industry. One common attack is poisoning attack in which attackers inject fake data into a recommender system such that the system makes recommendations as attackers' desire. For example, the unscrupulous producers may try to influence recommender systems in such a way that their items are recommended to users more often. This can be done by creating Sybil or fake accounts [5] and then manipulating these fake accounts to give the promoted item high rating scores. Meanwhile, these attackers will give well-crafted rating scores to a set of other items to improve the promotion. In a social recommender system, the attacker can take a variety of actions to mislead the system. It is well-known that online social networks are vulnerable to Sybil attacks, in which the attackers generate a number of fake accounts to perform malicious activities such as cyberbullying, distrupting democratic election and influencing financial markets. These attackers leverage the social influence among users to propagate information they create to mislead the behavior of victims. In social recommender systems, a user's preference is directly influenced by the preferences of his/her friends. Therefore, from the perspective of the attacker in a social recommender system, it can leverage the social influence of the social network to strengthen its attack. For example, the attacker can become friends with normal users by sending friend requests through the fake accounts, and then these normal users' preferences will be directly influenced by the fake ratings injected by the attacker.

Poisoning attacks have been studied for several specific recommender systems [6]–[10]. For example, [6] and [7] studied the poisoning attacks on a nearest-neighbor-based recommender system, [8] proposed poisoning attacks on association-rule-based recommender systems, and the poisoning attacks on graph-based recommender systems and factorization-based recommender systems are studied in [9] and [10], respectively. These attacks aim to maximally degrade the performance of the whole systems (also known as availability attack) or promote/demote a specific item to all users (also known as integrity attack) via injecting fake ratings. However, the poisoning attacks on social recommender systems have not been studied yet. Since the attacker in a social recommender system can generate the fake social relationships in addition to ratings, existing poisoning attacks that only inject fake ratings are not optimized in a social recommender, and more studies are needed for understanding poisoning attacks against social recommender systems.

In the paper, we study the poisoning attack on social recommender systems. Instead of performing the availability

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

and integrity attacks, we propose the targeted poisoning attack which aims to promote an item to a group of target users. Our design is mainly focused on the commonly-used factorization-based social recommendation. Our main contributions are as follows:

- We provide the first systematic study on targeted poisoning attacks to social recommender systems. Two types of attack actions are considered for the attacker: injecting fake ratings and generating fake relationships with normal users.
- We formulate the optimal poisoning attack as a bi-level program and then develop an efficient solution algorithm to find the optimal attack strategy, in which fake ratings and fake relationships are alternatively computed based on projected gradient descent.
- We evaluate our targeted poisoning attacks on real-world dataset and demonstrate that the factorization-based social recommender system is sensitive to our targeted poisoning attacks. The attacker can successfully influence the recommendations for the target users by generating fake relationships.

The rest of the paper is organized as follows: We first describe the preliminaries of the factorization-based social recommender system in Section II. Then, we describe our targeted poisoning attack model in Section III and formulate the poisoning attack as a bi-level optimization problem. Next, we propose an efficient solution algorithm to obtain the optimal attack strategy in Section IV. Finally, we evaluate our attack on real-world dataset in Section V and make conclusion in Section VI.

II. Preliminaries

Let $\mathcal{U} = \{u_1, u_2, \cdots, u_m\}$ be the set of m users and $\mathcal{E} = \{(u_i, u_f)|i, f \in [m]\}$ be the set of edges that represent the social relationships between pairs of users in a social recommender system. We use $\mathbf{E} \in \mathbb{R}^{m \times m}$ to denote the weight matrix of edges, where each row $\mathbf{e}_i = \{(e_{if})|f \in [m]\}$ represents the edge weight vector of user u_i. Specifically, an edge weight $e_{if} \in \{0, 1\}$ represents the weight of edge (u_i, u_f), where $e_{if} = 1$ indicates that u_i and u_f are friends and $e_{if} = 0$ indicates that u_i and u_f are not friends. Here, we consider bi-directional relationships so that $e_{if} = e_{fi}$. Let $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ be a set of n items and $\mathbf{R}$ be the user-item rating score matrix. Each entry r_{ij} in $\mathbf{R}$ for $i \in [m]$ and $j \in [n]$ denotes the rating score that the user u_i gave to the item v_j and is, without loss of generality, assumed to be in the set of integers $\{0, 1, \ldots, r_{\max}\}$, where $r_{\max}$ is the maximum rating score and $r_{ij} = 0$ indicates that user u_i did not rate item v_j. Since in real world applications each user only rates a very small portion of the items, the matrix $\mathbf{R}$ usually contains many zero entries and is extremely sparse.

Given a user-item rating matrix $\mathbf{R}$, the goal of recommender systems is to recommend each user N items that the user did not rate before via analyzing $\mathbf{R}$. As one of the most popular methods in recommender systems, factorization-based recommendation algorithms [11] aim to reduce the data dimension while preserving the major information content via approximating the rating matrix $\mathbf{R}$ by a multiplication of two low-rank matrices, i.e. $\mathbf{R} \approx \mathbf{X}^T\mathbf{Y}$, where $\mathbf{X} \in \mathbb{R}^{l \times m}$ and $\mathbf{Y} \in \mathbb{R}^{l \times n}$ are the latent user and item matrices with $l \ll \min(m, n)$, respectively. Here, the column vectors $\mathbf{x}_i$ of $\mathbf{X}$ and $\mathbf{y}_j$ of $\mathbf{Y}$ represent the latent feature vectors of user u_i and item v_j, respectively.

Considering that users' preferences could be influenced by their friends, the following factorization-based social recommendation model has been proposed in [12]:

$$\min_{\mathbf{X},\mathbf{Y}} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}(r_{ij} - \mathbf{x}_i^T\mathbf{y}_j)^2 + \beta\sum_{i=1}^{m}\sum_{f\in\mathcal{F}_i} g_{if}\|\mathbf{x}_i - \mathbf{x}_f\|_F^2 + \lambda_1\|\mathbf{X}\|_F^2 + \lambda_2\|\mathbf{Y}\|_F^2, \quad (1)$$

where I_{ij} is the indicator function that is equal to 1 if the user u_i rated the item v_j and 0 otherwise, $\mathcal{F}_i$ is the set of indexes of user u_i's friends, i.e., $\mathcal{F}_i = \{f|e_{if} = 1\}$, g_{if} indicates the similarity score between users u_i and u_f to be specified later, $\|\cdot\|_F^2$ denotes the Frobenius norm, and $\beta, \lambda_1, \lambda_2 > 0$ are regularization parameters. Intuitively, if two users have common behaviors, their preferences are similar. The cosine similarity function is used to define the similarity score between two users' rating behaviors, i.e.,

$$g_{if} = \frac{\mathbf{r}_i\mathbf{r}_f^T}{\|\mathbf{r}_i\|_2\|\mathbf{r}_f\|_2}, \quad (2)$$

where the row vector $\mathbf{r}_i := [r_{ij}|j \in [n]]$ denotes the rating vector of user u_i. From the above definition, the similarity score g_{if} is within the range $[0, 1]$, and a larger g_{if} means the two users u_i and u_f are more similar.

For Problem (1), the optimal solution can be found by using gradient descent in latent user and item matrices $\mathbf{X}$ and $\mathbf{Y}$. We use $\mathbf{U} = [\mathbf{u}_i|i \in [m]]$ to denote the optimal user feature matrix and $\mathbf{V} = [\mathbf{v}_j|j \in [n]]$ to denote the optimal item feature matrix. The recommender system uses $\mathbf{U}$ and $\mathbf{V}$ to make predictions for zero entries in the original user-item matrix $\mathbf{R}$ by computing $\hat{\mathbf{R}} = \mathbf{U}^T\mathbf{V}$ and then recommends N unrated items with the highest predicted rating scores for each user.

III. Targeted Poisoning Attack

A. Attack Model

Attacker's goal: In the targeted poisoning attack, the attacker aims to promote an item v_t to a set of target users $\mathcal{K}$ by injecting fake ratings and edges. Assume the recommender system recommends a list of items $\mathcal{A}_k$ to the target user $u_k \in \mathcal{K}$, consisting of N unrated items that have the largest predicted rating scores. If the item $v_t \in \mathcal{A}_k$, the attacker has successfully attacked the target user u_k. Therefore, to attack the target user u_k, the attacker's goal is to maximize the hit probability $\Pr(v_t \in \mathcal{A}_k)$. Since it is hard to directly model

the hit probability, similar to [9], we use the Wilcoxon-Mann-Whitney loss [13] to approximate it, which is defined as

$$\mathcal{L}_k(\hat{\mathbf{r}}_k) = \sum_{\{j|v_j \in \mathcal{A}_k\}} \frac{1}{1+\exp(-(\hat{r}_{kj} - \hat{r}_{kt}))}, \tag{3}$$

where $\hat{\mathbf{r}}_k := \mathbf{u}_k^T \mathbf{V}$ is the predicted rating score vector for the target user u_k. A smaller attack loss indicates a larger hit probability. Accordingly, to attack all users in $\mathcal{K}$, the overall loss function of the attacker is $\mathcal{L} = \sum_{\{k|u_k \in \mathcal{K}\}} \mathcal{L}_k(\mathbf{u}_k^T \mathbf{V})$.

Attacker's background knowledge: We consider the white-box setting where the attacker has a high level of knowledge about the recommender system. Specifically, we assume that (i) the attacker knows the algorithm used by the recommender system; and (ii) the user rating scores and social relationships are public and can be crawled and collected by the attacker.

Attacker's actions: The attacker is assumed to control a set of malicious accounts/nodes in the system. The attacker can create fake ratings on selected items through these malicious nodes. The attacker can also become friends with the target users in order to affect their recommendation results. Given the limited budget of creating friendships with normal users, we assume that each malicious node can have friendships with at most n_e users. Furthermore, to avoid simple malicious node detection methods based on the number of ratings, we assume that each malicious node can rate at most n_r items.

B. Attack as an Optimization Problem

Assume the attacker controls a set of m_s malicious nodes $\mathcal{S} = \{u_s | s \in [m_s]\}$. Let $\mathbf{r}_s = \{(r_{sj}) | j \in \mathcal{V}\}$ be the rating score vector of a malicious node u_s, where r_{sj} represents the rating score that the malicious node u_s gives to the item v_j. Let $\mathbf{e}_s = \{(e_{si}) | i \in \mathcal{U}\}$ be the edge weight vector of a malicious node u_s, where e_{si} represents the edge weight between the malicious node u_s and the user u_i. Then, the attacker's goal is to find the optimal rating score vector $\{\mathbf{r}_s | s \in [m_s]\}$ and edge weight vector $\{\mathbf{e}_s | s \in [m_s]\}$ for all malicious nodes to minimize the total attack loss. The targeted poisoning attack can be formulated as the following bi-level program [14]:

$$\min_{\{\mathbf{r}_s \in \mathcal{P}_r, \mathbf{e}_s \in \mathcal{P}_e, \forall s \in [m_s]\}} \quad \mathcal{L} = \sum_{\{k|u_k \in \mathcal{K}\}} \mathcal{L}_k(\mathbf{u}_k^T \mathbf{V}), \tag{4}$$

where $\mathbf{u}_k$ and $\mathbf{V}$ are obtained by solving the following optimization problem:

$$\begin{aligned}
\min_{\mathbf{X},\mathbf{Y}} & \sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}(r_{ij} - \mathbf{x}_i^T \mathbf{y}_j)^2 + \sum_{s=1}^{m_s}\sum_{j=1}^{n} I_{sj}(r_{sj} - \mathbf{x}_s^T \mathbf{y}_j)^2 + \\
& \beta \sum_{i=1}^{m} \Big(\sum_{f \in \mathcal{F}_i^u} g_{if} \|\mathbf{x}_i - \mathbf{x}_f\|_F^2 + \sum_{s \in \mathcal{F}_i^s} g_{is} \|\mathbf{x}_i - \mathbf{x}_s\|_F^2 \Big) + \\
& \beta \sum_{s=1}^{m_s} \Big(\sum_{f \in \mathcal{F}_s^u} g_{sf} \|\mathbf{x}_s - \mathbf{x}_f\|_F^2 + \sum_{s' \in \mathcal{F}_s^s} g_{ss'} \|\mathbf{x}_s - \mathbf{x}_{s'}\|_F^2 \Big) \\
& + \lambda_1 \|\mathbf{X}\|_F^2 + \lambda_2 \|\mathbf{Y}\|_F^2.
\end{aligned} \tag{5}$$

Here, $\mathcal{F}_i^u := \{f | e_{if} = 1, u_f \in \mathcal{U}\}$ is the set of indexes of normal users that have friendship with user u_i, $\mathcal{F}_i^s := \{s | e_{is} = 1, u_s \in \mathcal{S}\}$ is the set of indexes of malicious users that have friendship with user u_i, $\mathcal{P}_r = \{r_{sj} \in \{0, 1, \cdots, r_{max}\}, \|\mathbf{r}_s\|_0 \le n_r\}$ is the feasible region of rating score vector $\mathbf{r}_s$, and $\mathcal{P}_e = \{e_{si} = e_{is} \in \{0, 1\}, \|\mathbf{e}_s\|_0 \le n_e\}$ is the feasible region of edge weight vector $\mathbf{e}_s$.

IV. Computing Optimal Attack Strategies

A. Solution Framework

Solving the bi-level optimization problem above is highly challenging because of the discrete variables $\mathbf{r}_s$ and $\mathbf{e}_s$. For bi-level optimization problems with continuous variables, approximate solutions can be found by gradient descent methods based on the KKT conditions of the lower level problem [10], [15], [16]. However, such methods cannot be directly applied to our case. Inspired by [9], [17], we propose a framework to approximately solve our problem. First of all, instead of optimizing the fake data (i.e. the rating scores and edge weights) of all malicious nodes simultaneously, we optimize their fake data sequentially. In particular, given the original rating scores and edge weights, as well as the fake rating scores and edge weights added so far, we find the rating scores and edge weights for the next malicious node to minimize the attacker's loss. Furthermore, for each malicious node, we propose an alternative optimization approach to find the optimal rating scores and edge weights. Specifically, we alternatively generate fake rating scores $\mathbf{r}_s$ with fixed fake edge weights and then generate fake edge weights $\mathbf{e}_s$ with fixed fake rating scores until convergence. The details of our framework are given in Algorithm 1. In what follows, we present how to generate fake rating scores and edge weights.

Algorithm 1 Our Targeted Poisoning Attacks

Input: Rating matrix $\mathbf{R}$, social edge weights $\mathbf{E}$, item to be promoted v_t, target users $\mathcal{K}$, malicious nodes $\mathcal{S}$, maximum iteration number H, parameters n_e, n_r, r_{max}.

```
for s = 1 to m_s do
    r_s ← 0, e_s ← 0;
    //Alternatively update edge weights and rating scores.
    for h = 1 to H do
        r_s ← FRGenerator(R, E, r_s, e_s, n_r);
        e_s ← FEGenerator(R, E, r_s, e_s, n_e);
    end for
    //Inject the rating scores and edge weights of malicious node u_s to the system.
    R ← R ∪ r_s, E ← E ∪ e_s;
end for
return {r_s, e_s | s ∈ [m_s]}
```

B. Fake Rating Generator

In this section, we describe the details of generating fake rating scores. With fixed fake edge weights $\{\mathbf{e}_s : s \in [m_s]\}$,

the attacker's objective becomes

$$\min_{\{\mathbf{r}_s \in \mathcal{P}_r : s \in [m_s]\}} \quad \mathcal{L} = \sum_{\{k|u_k \in \mathcal{K}\}} \mathcal{L}_k(\mathbf{u}_k^T \mathbf{V}). \tag{6}$$

Since the fake rating scores are discrete variables, gradient descent based algorithms cannot be used to find the optimal solution. Therefore, we propose an approximation technique to optimize the fake rating scores. Specifically, we relax the fake rating scores as continuous variables, find the optimal solution that minimizes the total attack loss in (6), and then transform the solution to discrete rating scores. The core part of this approximation technique is the projected gradient descent, which updates $\mathbf{r}_s$ in its feasible region based on the gradient. Specifically, at iteration t, we update $\mathbf{r}_s$ as follows,

$$\mathbf{r}_s{}^{(t)} = \text{Proj}_{\mathcal{P}_r}\big(\mathbf{r}_s{}^{(t-1)} - \alpha_\tau \nabla_{\mathbf{r}_s} \mathcal{L}\big), \tag{7}$$

where $\text{Proj}_{\mathcal{P}_r}$ is the projection operator onto the feasible region of $\mathcal{P}_r$ and α_τ is the step size. To project $\mathbf{r}_s$ into its feasible region, we first update $\mathbf{r}_s$ using the gradient descent and choose n_r items that have the largest rating scores as the candidate items to be rated. Then we truncate the rating scores of candidate items to the level of $[0, r_{\max}]$ and set the rating scores of non-candidate items to zero. By discretizing these rating scores, we finally obtain the rating scores. The details are given in Algorithm 2.

In the following, we describe the computing process of the gradient $\nabla_{\mathbf{r}_s} \mathcal{L}$ in (7). Indeed, assuming that $\mathcal{L}$ does not depend directly on $\mathbf{r}_s$, but only through $\hat{\mathbf{r}}_k$, we can compute the gradient of $\mathcal{L}$ w.r.t the element of $\mathbf{r}_s$ using the chain rule as:

$$\nabla_{r_{sj}} \mathcal{L} = (\nabla_{r_{sj}} \hat{\mathbf{r}}_k)(\nabla_{\hat{\mathbf{r}}_k} \mathcal{L}_k)(\nabla_{\mathcal{L}_k} \mathcal{L}). \tag{8}$$

The last two terms in the above equation are straightforward to compute as

$$(\nabla_{\hat{r}_{kj}} \mathcal{L}_k)(\nabla_{\mathcal{L}_k} \mathcal{L}) = \begin{cases} \dfrac{e^{\hat{r}_{kt} - \hat{r}_{kj}}}{(1 + e^{\hat{r}_{kt} - \hat{r}_{kj}})^2}, \text{ if } j \in \{j | v_j \in \mathcal{A}_k, j \neq t\} \\ \displaystyle\sum_{\{j' | v_{j'} \in \mathcal{A}_k\}} \dfrac{-e^{\hat{r}_{kj} - \hat{r}_{kj'}}}{(1 + e^{\hat{r}_{kj} - \hat{r}_{kj'}})^2}, \text{ if } \{j | v_j \notin \mathcal{A}_k, j = t\} \\ 0, \text{ if } j \in \{j | v_j \in \mathcal{A}_k, j = t\} \cup \{j | v_j \notin \mathcal{A}_k, j \neq t\} \end{cases}. \tag{9}$$

The first gradient term is much harder to evaluate because $\hat{\mathbf{r}}_k$ is computed from the optimal solution to Problem (5). Inspired by [10], [15], we exploit the Karush-Kuhn-Tucker (KKT) conditions of (5) to approximately compute $\nabla_{r_{sj}} \hat{\mathbf{r}}_k$. The underlying trick is to replace Problem (5) with its KKT conditions and require such conditions remain valid while updating $\mathbf{r}_s$. Specifically, the optimal solution of Problem (5) satisfies following KKT conditions:

$$\lambda_1 \mathbf{u}_i = \sum_{j=1}^{n} I_{ij}(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)\mathbf{v}_j + 2\beta \sum_{f \in \mathcal{F}_i^u} g_{if}(\mathbf{u}_f - \mathbf{u}_i) + 2\beta \sum_{s \in \mathcal{F}_i^s} g_{is}(\mathbf{u}_s - \mathbf{u}_i), \forall i \in [m], \tag{10}$$

and

$$\lambda_2 \mathbf{v}_j = \sum_{i=1}^{m} I_{ij}(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)\mathbf{u}_i + \sum_{s=1}^{m_s} I_{sj}(r_{sj} - \mathbf{u}_s^T \mathbf{v}_j)\mathbf{u}_s, \forall j \in [n]. \tag{11}$$

Subsequently, $\mathbf{u}_k$ and $\mathbf{v}_j$ can be expressed as functions of the fake ratings r_{sj}. Given that $\hat{\mathbf{r}}_k = \mathbf{u}_k^T \mathbf{V}$, differentiating again using the chain rule, one yields:

$$\nabla_{r_{sj}} \hat{\mathbf{r}}_k = (\nabla_{r_{sj}} \mathbf{u}_k)^T \mathbf{V} + \mathbf{u}_k^T (\nabla_{r_{sj}} \mathbf{V}). \tag{12}$$

Solving $\nabla_{r_{sj}} \mathbf{u}_k$ based on (10), we obtain

$$\nabla_{r_{sj}} \mathbf{u}_k = \Big(\sum_{j=1}^{n} I_{kj} \mathbf{v}_j \mathbf{v}_j^T + 2\beta \big(\sum_{f \in \mathcal{F}_k^u} g_{kf} + \sum_{s \in \mathcal{F}_k^s} g_{ks} \big) \mathbf{1}_l + \lambda_1 \mathbf{1}_l \Big)^{-1} e_{ks} (\nabla_{r_{sj}} g_{ks})(\mathbf{u}_s - \mathbf{u}_k). \tag{13}$$

Here, e_{ks} is the weight of edge (u_k, u_s), $\mathbf{1}_l$ is a l-dimensional identity matrix and $\nabla_{r_{sj}} g_{ks}$ is the gradient of similarity function g_{ks} w.r.t r_{sj}. According to (2), we have that

$$\nabla_{r_{sj}} g_{ks} = \frac{r_{kj}}{\|\mathbf{r}_k\|_2 \|\mathbf{r}_s\|_2} - \frac{\mathbf{r}_k \mathbf{r}_s^T r_{sj}}{\|\mathbf{r}_k\|_2 \|\mathbf{r}_s\|_2^3}. \tag{14}$$

Then, based on (11) we obtain the solution of $\nabla_{r_{sj}} \mathbf{V}$, where each column is

$$\nabla_{r_{sj}} \mathbf{v}_j = I_{sj} \Big(\sum_{i=1}^{m} I_{ij} \mathbf{u}_i \mathbf{u}_i^T + \mathbf{u}_s \mathbf{u}_s^T + \lambda_2 \mathbf{1}_l \Big)^{-1} \mathbf{u}_s. \tag{15}$$

Combining with (8)-(9) and (12)-(15), we can finally obtain the gradient of the total attack loss w.r.t the rating score vector $\nabla_{\mathbf{r}_s} \mathcal{L}$.

Algorithm 2 Fake Rating Generator (FRGenerator)

Input: Rating matrix $\mathbf{R}$, social edge weights $\mathbf{E}$, fake rating scores $\mathbf{r}_s$, fake edge weights $\mathbf{e}_s$, number of fake ratings n_r, step size α_τ, maximum iteration number T.

```
r_s^(0) ← r_s
for t = 1 to T do
    Solve Problem (5) to get the optimal U and V;
    Compute the gradient ∇_{r_s} L;
    r_s^(t) ← r_s^(t−1) − α_τ ∇_{r_s} L;
end for
Select n_r items with largest rating scores as candidate items;
Update r_s by truncating the rating scores of candidate items to the level of [0, r_max] and setting the rating scores of non-candidate items as zero;
Discretize r_s;
return r_s
```

C. Fake Edge Generator

In this section, we show how to generate the edge weight vector of a malicious node. With fixed fake rating scores, the attacker's objective is

$$\min_{\{\mathbf{e}_s \in \mathcal{P}_e : s \in [m_s]\}} \quad \mathcal{L} = \sum_{\{k | u_k \in \mathcal{K}\}} \mathcal{L}_k(\mathbf{u}_k^T \mathbf{V}). \tag{16}$$

To solve Problem (16), we approximately optimize the discrete variables $\mathbf{e}_s$ based on the projected gradient descent. Specifically, we relax the fake edge weights as continuous variables, find the optimal solution that minimizes the total attack loss in Problem (16), and then discretize them to obtain the final edge weights. In particular, at iteration t, we update the edge weight vector $\mathbf{e}_s$ by the projected gradient descent:

$$\mathbf{e}_s{}^{(t)} = \text{Proj}_{\mathcal{P}_e}\big(\mathbf{e}_s{}^{(t-1)} - \alpha_\tau \nabla_{\mathbf{e}_s} \mathcal{L}\big), \tag{17}$$

where $\text{Proj}_{\mathcal{P}_e}$ is the projection operator onto the feasible region of $\mathcal{P}_e$ and α_τ is the step size. To project the fake edge weights into its feasible region, we first update $\mathbf{e}_s$ using the gradient descent and choose n_e edges with largest weights as the candidate edges to be added. Then, we truncate the weight of candidate edges to the range of $[0, 1]$, set the weight of non-candidate edges as zero, and then approximate the edge weights to the nearest integers. The details of generating fake edge weights are given in Algorithm 3. In what follows, we describe the computation of the required gradient $\nabla_{\mathbf{e}_s} \mathcal{L}$.

We still exploit the KKT conditions of Problem (5) to compute the gradient. Specifically, assuming that $\mathcal{L}$ does not depend directly on $\mathbf{e}_s$ but only through $\hat{\mathbf{r}}_k$, we compute the gradient of $\mathcal{L}$ w.r.t the element of $\mathbf{e}_s$ using the chain rule as:

$$\nabla_{e_{si}} \mathcal{L} = (\nabla_{e_{si}} \hat{\mathbf{r}}_k)(\nabla_{\hat{\mathbf{r}}_k} \mathcal{L}_k)(\nabla_{\mathcal{L}_k} \mathcal{L}). \tag{18}$$

The solution of the last two gradient term is given in (9), so we mainly show the computation of the first gradient term. From (10) and (11), we can see that $\mathbf{u}_i$ depends on e_{si} but $\mathbf{v}_j$ does not depend on e_{si}. Therefore, differentiating again with the chain rule, one yields:

$$\nabla_{e_{si}} \hat{\mathbf{r}}_k = (\nabla_{e_{si}} \mathbf{u}_k)^T \mathbf{V}. \tag{19}$$

According to (10), we have

$$\nabla_{e_{si}} \mathbf{u}_k = \Big(\sum_{j=1}^{n} I_{kj} \mathbf{v}_j \mathbf{v}_j^T + 2\beta \big(\sum_{f \in \mathcal{F}_k^u} g_{kf} + \sum_{s \in \mathcal{F}_k^s} g_{ks} \big) \mathbf{1}_l + \lambda_1 \mathbf{1}_l \Big)^{-1} g_{ks} (\mathbf{u}_s - \mathbf{u}_k). \tag{20}$$

Finally, using (18)-(20), we can obtain the gradient of the total attack loss w.r.t the fake edge weights $\nabla_{\mathbf{e}_s} \mathcal{L}$.

Algorithm 3 Fake Edge Generator (FEGenerator)

Input: Rating matrix $\mathbf{R}$, social edge weights $\mathbf{E}$, fake rating scores $\mathbf{r}_s$, fake edge weights $\mathbf{e}_s$, number of fake edges n_e, step size α_τ, maximum iteration number T.

$\mathbf{e}_s^{(0)} \leftarrow \mathbf{e}_s$
for $t = 1$ to T **do**
  Solve Problem (5) to get the optimal $\mathbf{U}$ and $\mathbf{V}$;
  Compute the gradient $\nabla_{\mathbf{e}_s} \mathcal{L}$;
  $\mathbf{e}_s^{(t)} \leftarrow \mathbf{e}_s^{(t-1)} - \alpha_\tau \nabla_{\mathbf{e}_s} \mathcal{L}$;
end for
Select n_e edges with the largest weight as candidate edges;
Update $\mathbf{e}_s$ by truncating the weights of candidate edges to the level of $[0, 1]$ and setting the weights of non-candidate edges as zero;
Discretize $\mathbf{e}_s$;
return $\mathbf{e}_s$

V. EVALUATION

A. Experimental Setup

Dataset. We use the publicly available FilmTrust dataset [18] to evaluate the performance of our attack. FilmTrust is a movie sharing and rating website. The FilmTrust dataset contains 35,494 ratings applied to 2,071 items by 1,642 users and 1,309 social links. To avoid the "cold-start" problem, we consider users having at least 1 rating record. The original rating score in FilmTrust dataset is within the range $\{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$, which is scaled to the new range $\{1, 2, \ldots, 8\}$ in simulations.

Evaluation Metrics. To evaluate the performance of our attacks, we calculate the Attack Success Rate (ASR). Specifically, a target user is successfully attacked if the recommender system recommends the promoted item to him/her, and ASR is the ratio of the successful attacks. Larger ASR corresponds to better attacking effect.

For the recommender system in our experiments, we randomly split the rating data into training (60%), validation (20%), and test (20%) sets. We use the 5-fold cross validation to tune the hyperparameters β, λ_1 and λ_2, and then train a social recommender system and compute predicted ratings for all users. In addition, the maximum rating score $r_{\max} = 8$. We set the size of recommendation list $N = 10$ and the number of malicious nodes $|\mathcal{S}| = 1$ for all experiments.

B. Attack Performance

We first simulate our poisoning attack to promote an item to only one target user, and then simulate our attack to promote an item to a small group of target users.

Attack on a single target user. In this attack, attacker's goal is to promote the item v_t to the target user u_k. In order to show the performance of this attack, we perform 300 independent targeted poisoning attacks with different choices of target users and promoted items. In this experiment, we randomly choose three users from the normal users as the target users. For each target user, we promote 100 popular items (i.e. items that have the largest predicted rating scores for the target user) independently to him/her. Then, we calculate the ASR of these attacks. Here, we set the maximum number of fake edges n_e as 1. The result in Fig. 1 shows the ASR of our attack w.r.t. the number of fake ratings. The ASR increases as the number of fake ratings increases. As the original recommender system will recommend 10 items that have the largest predicted rating

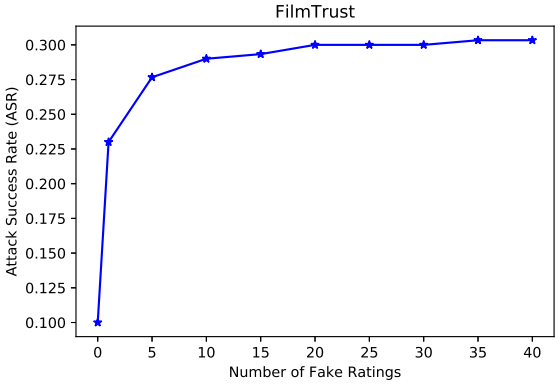

Fig. 1. Poisoning attack on a single target user.

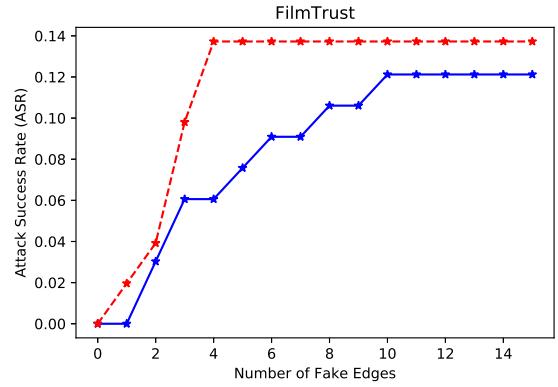

Fig. 2. Poisoning attack on a group of target users.

scores to each target user, the ASR is 0.1 when there has no fake rating. Note that for each attack we choose the optimal number of fake ratings using cross-validation since the optimal number of fake ratings for different target users and promoted items are different. Thus, the ASR in Fig. 1 stops increasing when the number of fake ratings is large enough.

Attack on a group of target users. In this experiment, the attacker aims to promote an item to a group of target users connecting with each other. In the experiment, we select a small group of users by extracting the ego-network of a user from the original social network. Specifically, we extract the ego-network of the user 508 (denoted by group 1) and the user 187 (denoted by group 2), who have the largest number of friends. Group 1 contains 66 users and its network density is 0.083, while group 2 contains 51 users and its network density is 0.130. We preform the targeted poisoning attack on these two groups independently. Specifically, for the attack on each group, we randomly select an item that has not been rated by target users to promote. Here, we set the maximum number of fake ratings n_r as 1. The results in Fig.2 show the ASR of our attacks w.r.t. the number of fake edges. As the number of fake edges increases, the ASR increases at first and stops increasing when the number of fake edges is large enough. Moreover, according to the ASR of group 2, we can see that the number of victims is greater than the number of fake edges when the number of fake edges is less than 10. It shows that a set of users that are not connected with the malicious node are indirectly attacked.

VI. Conclusion

In this work, we have proposed the targeted poisoning attack on the social recommender system which aims to promote an item to a set of target users. We have formulated the attack as a bi-level program and developed an approximation algorithm to solve it efficiently. Through evaluations on the real-world dataset, we have demonstrated that the social recommender system is sensitive to the targeted poisoning attacks. Moreover, we find that the attacker can successfully attack some target users even though the attacker has no direct relationships with them.

Acknowledgment

The work of R. Hu and Y. Gong is supported by the U.S. National Science Foundation under grant CNS-1850523. The work of M. Pan is supported in part by the U.S. National Science Foundation under grants US CNS-1350230 (CAREER), CNS-1646607, CNS-1702850, and CNS-1801925.

References

[1] H. Ma, H. Yang, M. R. Lyu, and I. King, "Sorec: social recommendation using probabilistic matrix factorization," in *ACM CIKM*, 2008.
[2] M. Jamali and M. Ester, "A matrix factorization technique with trust propagation for recommendation in social networks," in *ACM RecSys*, 2010, pp. 135–142.
[3] G. Guo, J. Zhang, and N. Yorke-Smith, "Trustsvd: collaborative filtering with both the explicit and implicit influence of user trust and of item ratings," in *AAAI*, 2015.
[4] B. Yang, Y. Lei, J. Liu, and W. Li, "Social collaborative filtering by trust," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 8, pp. 1633–1647, 2017.
[5] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*, 2002, pp. 251–260.
[6] S. K. Lam and J. Riedl, "Shilling recommender systems for fun and profit," in *WWW*, 2004, pp. 393–402.
[7] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams, "Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness," *ACM Transactions on Internet Technology*, vol. 7, no. 4, p. 23, 2007.
[8] G. Yang, N. Z. Gong, and Y. Cai, "Fake co-visitation injection attacks to recommender systems." in *NDSS*, 2017.
[9] M. Fang, G. Yang, N. Z. Gong, and J. Liu, "Poisoning attacks to graph-based recommender systems," in *ACM ACSAC*, 2018, pp. 381–392.
[10] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," in *NeurIPS*, 2016.
[11] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.
[12] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, "Recommender systems with social regularization," in *ACM WSDM*, 2011, pp. 287–296.
[13] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *ACM WSDM*, 2011.
[14] Z. Huang, M. Pan, and Y. Gong, "Robust truth discovery against data poisoning in mobile crowdsensing," in *IEEE GLOBECOM*, 2019.
[15] M. Zhao, B. An, Y. Yu, S. Liu, and S. J. Pan, "Data poisoning attacks on multi-task relationship learning," in *AAAI*, 2018.
[16] M. Sun, J. Tang, H. Li, B. Li, C. Xiao, Y. Chen, and D. Song, "Data poisoning attack against unsupervised node embedding methods," *arXiv preprint arXiv:1810.12881*, 2018.
[17] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *ACM SIGKDD*, 2018.
[18] G. Guo, J. Zhang, and N. Yorke-Smith, "A novel bayesian similarity measure for recommender systems," in *IJCAI*, 2013, pp. 2619–2625.