PoisonRec: An Adaptive Data Poisoning Framework for Attacking Black-box Recommender Systems

Junshuai Song†, Zhao Li*‡, Zehong Hu*, Yucheng Wu†, Zhenpeng Li*, Jian Li* and Jun Gao†‡
†The Key Laboratory of High Confidence Software Technologies, Ministry of Education,
Department of Computer Science, Peking University, China
Email: {songjunshuai,wuyucheng,gaojun}@pku.edu.cn
*Alibaba Group, China
Email: {lizhao.lz,zehong.hzh,zhen.lzp,zeshan.lj}@alibaba-inc.com
‡The corresponding authors

***Abstract*—Data-driven recommender systems that can help to predict users' preferences are deployed in many real online service platforms. Several studies show that they are vulnerable to data poisoning attacks, and attackers have the ability to mislead the system to perform as their desires. Considering the realistic scenario, where the recommender system is usually a black-box for attackers and complex algorithms may be deployed in them, how to learn effective attack strategies on such recommender systems is still an under-explored problem. In this paper, we propose an adaptive data poisoning framework, PoisonRec, which can automatically learn effective attack strategies on various recommender systems with very limited knowledge. PoisonRec leverages the reinforcement learning architecture, in which an attack agent actively injects fake data (user behaviors) into the recommender system, and then can improve its attack strategies through reward signals that are available under the strict black-box setting. Specifically, we model the attack behavior trajectory as the Markov Decision Process (MDP) in reinforcement learning. We also design a Biased Complete Binary Tree (BCBT) to reformulate the action space for better attack performance. We adopt 8 widely-used representative recommendation algorithms as our testbeds, and make extensive experiments on 4 different real-world datasets. The results show that PoisonRec has the ability to achieve good attack performance on various recommender systems with limited knowledge.**

***Index Terms*—Data Poisoning Attack, Recommender System, Reinforcement Learning**

I. Introduction

Recommender system, as a data-driven way to predict online users' potential preferences, is playing an increasingly important role in online service platforms. The basic idea behind recommender systems is collaborative filtering (CF) [1]–[3], which can help us make predictions about users' potential interests by collecting other users' preference information. In recent years, many recommendation algorithms have been proposed and achieved great success in practical applications [4]–[9].

There have been some evidences showing that existing recommender systems are vulnerable under data poisoning attacks (also known as shilling attacks) [1], [10], [11]. Some attackers may inject some fake data into a recommender system, so that the recommender system is poisoned, and performs as their desires [11], [12]. Promoting target items is a common attack objective in a real-world recommender system, after which the target items may be recommended to users more frequently than before. For example, Yang et al. [12] have successfully performed the item promotion attack on several real-world popular services like Youtube and eBay where the co-visitation recommender system [12] is deployed. It is profitable for attackers, and we mainly consider such an item promotion problem in this work.

Fig. 1. An attack example. The attacker tries to promote the target item *Hat* by frequently clicking *Hat* and another popular item *Chair*.

In the literature, there are mainly two categories of attack methods proposed. The first category is manually designed heuristic methods, which follow simple rules to conduct attacks [1], [10], [11], [13], [14]. For example, a common strategy is to frequently click the target item and another popular item so that the recommender system may be misled that the target item is closely related to the popular one as illustrated in Figure 1. Then, users who have bought the popular item may receive recommended results including the target item. An obvious drawback of methods in this category is that the simple rules cannot always achieve satisfying attack performance on various recommender systems.

Another category of attack methods is learning-based [12], [15]–[17]. Most of them are particularly designed for given recommender systems [12], [15], [16] where specific recommendation algorithms are deployed. In addition, they usually require strong knowledge, like explicit user-item interaction information [17], in their attack background.

Recommender systems with implicit feedback [5] are more

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00021

practical. They are always required protecting users' privacy, and users' profile information and interactions with the system are confidential. For security reasons, they will not disclose their system composition, algorithms, etc. We consider such a realistic black-box scenario, which means the recommender system is totally agnostic to attackers. It is obvious that attacking a black-box recommender system is very challenging. On the one hand, we only have very limited background knowledge about the recommender system. We know nothing about the log data, the system components, and the recommendation algorithms, etc. On the other hand, there are various algorithms developed in the system, which might have been upgraded and be far more complex than before [3], [6], [9] with the development of the deep learning technique. It is very difficult and also time-consuming to directly design effective attacks for each of them.

To our best knowledge, it is still an under-explored problem to learn effective attack strategies on different complex black-box recommender systems. Model-free reinforcement learning [18]–[20] could help with this goal, which does not make any assumption, and requires no (or limited) knowledge on the interactive environment (the recommender system to be attacked in this work). Inspired by its success in many areas [21]–[24], we adopt it as the core component of our solution.

We propose PoisonRec, an adaptive data poisoning attack framework. Since PoisonRec is decoupled from the interactive recommender system under the model-free reinforcement learning architecture, it can learn effective attack strategies on various recommender systems, and adopt any suitable advanced reinforcement learning method. It repeatedly injects generated fake user behaviors into a recommender system, and improves its own attack strategies through available reward signals until the model converges. Specifically, we model the sequential attack behavior trajectory as an MDP. Given the state s_t that contains a variable-length attack trajectory at step t, we use the long short-term memory (LSTM) network to embed them into a fixed-length hidden variable, based on which the next action a_t can be sampled. For the reward signal, we consider the observable number of Page View (PV) [22], [25] on a pre-defined target item set under the black-box setting. To overcome the low convergency problem on a large action space in reinforcement learning, we further reformulate the action space through a Biased Complete Binary Tree (BCBT), which can obviously improve the model performance and convergency efficiency of PoisonRec. We adopt real-world data from Steam, MovieLens and Amazon, and choose 8 different widely-used recommendation algorithms as our attack testbeds. Extensive experiments show that PoisonRec has the ability to adaptively learn effective attack strategies on different kinds of recommender systems.

The remainder of this paper is organized as follows. We first review preliminaries in Section II, and then propose our PoisonRec framework in Section III. Section IV reports the experimental results. Finally, we conclude the paper in Section V.

II. Preliminaries

We review the development of recommender systems in recent years, and discuss existing attack methods.

A. Development of Recommender Systems

Recommender systems can predict users' preferences, and have been greatly developed in recent years. ItemKNN [12], [26] is an item-based CF method, and it recommends items through calculating similarities between items. MF [2] is a basic model-based CF recommendation algorithm. BPR [5] optimizes MF through a pair-wise ranking loss. With the development of the deep neural network technique, many deep learning-based methods [3], [4] are also proposed. He et al. [3] give NeuMF to replace the inner product in MF with a neural architecture, so that it can learn an arbitrary user-item preference function from data. Suvash et al. [27] utilize the auto-encoder paradigm and propose AutoRec, which can directly predict users' potential interests towards all items. Besides, sequential and graph structure information proved to have obvious benefits in predicting users' preferences. Balázs et al. [6] and Zhou et al. [28] utilize users' historical behaviors to predict the next item that the user may click or purchase. Wang et al. [9] exploit the high-order structure information in the user-item graph, and achieve much better recommendation performance.

B. Attack Methods on Recommender Systems

The most common attack methods against recommender systems are manually designed heuristic methods [1], [10], [11] like popular attack and random attack. They can be applied to any type of recommender systems, but the results are usually not satisfying. Recent studies on attacking recommender systems achieve much better performance [12]–[17]. Seminario et al. [13], [14] propose to select "power users" and "power items" in advance when developing the attack strategy. Yang et al. [12] try to attack a simple non-personalized co-visitation recommender system. They model the attack strategy as a constrained linear optimization problem, by solving which they can spoof the recommender system to make recommendations as their desires. Christakopoulou et al. [17] propose a method that can be adopted to attack various recommender systems, in which they first generate fake rating scores for the multiple controlled user accounts to initialize a matrix, and then use an *approximate gradient* method to iteratively update the matrix for obtaining the effective attack strategy.

III. PoisonRec: An Adaptive Data Poisoning Attack Framework

In this section, we first summarize some frequently-used notations in Table I, and give an overview of real recommender systems and attackers' knowledge requirements under the black-box setting. Then, we formulate the attack problem, and finally propose the novel adaptive data poisoning attack framework, PoisonRec.

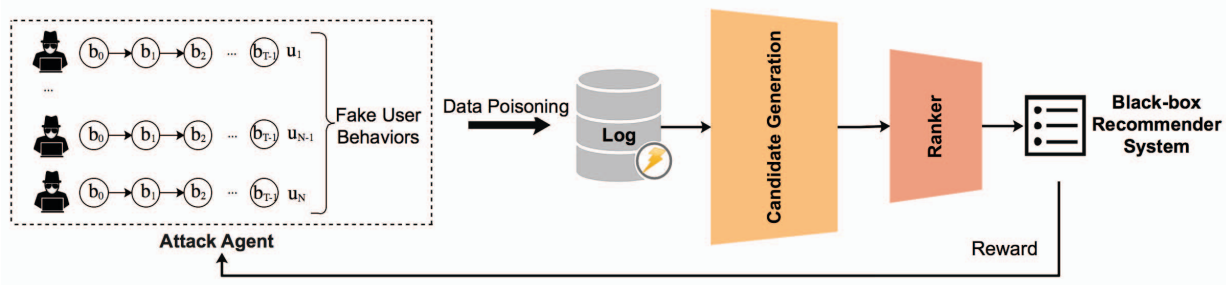


Fig. 2. The overall framework of PoisonRec.

Notation	Explanation
C_u	It is the candidate set that a user u may interested in.
L_u	It is the list recommended to a user u.
I	It represents the original (existing) item set in a recommender system.
I_t	It is the target item set that the attacker wants to promote, which consists of 8 new items in this paper.
N	It is the number of attackers.
τ	It means an attack trajectory that consists of T items.

TABLE I
FREQUENTLY-USED NOTATIONS.

A. Overview

1) Recommender System: A typical recommender system usually consists of at least two separate components: Candidate Generation and Ranker. Candidate Generation [4] is responsible for selecting hundreds of items from the entire corpus into the candidate set C_u for each user u. Ranker [3], [6], [9], [29] is used to rank the items in C_u, which can more accurately estimate the user preference scores on items, and then the top-k items with the highest scores could be selected as the final recommendation result L_u.

2) Attack Background: We discuss the attacker's goal, knowledge, and capability in this part.

Attacker's Goal. We introduce a widely-used concept, the number of Page View (PV) [22], [25], that measures items' exposure within a certain period of time in the real online recommender system. We formally defined the number of PV on a pre-defined target item set I_t as $RecNum$. Specifically, we give:

$$RecNum = \sum_{u} |L_u \cap I_t| \tag{1}$$

where u is a user accessing the system, L_u indicates the recommended results, and I_t is the target item set. Though attackers cannot obtain L_u for other users when attacking a real recommender system under the black-box setting, $RecNum$, as a statistical indicator, is usually available. Attackers know how many users have viewed their products, how many goods they have sold, etc. Given a target item set I_t, the attacker's goal is to increase $RecNum$ as much as possible. In addition, it is worth explaining how data poisoning attack affects $RecNum$. The recommendation model in a recommender system is always trained on all users' log data. If the attacker injects some fake user data into a recommender system, this part of data will affect the training of the recommendation model in this attacked recommender system, and get the model to learn something it should not. If the attacked recommender system is misled that a target item is a popular one, many other users may receive this target item in their recommendation results, leading to a high $RecNum$.

Attacker's Knowledge. We focus on learning highly effective strategies to attack various complex recommender systems under the black-box setting.

- Attackers do not know any detail about the target recommender system, *e.g.*, what components it contains, or what the Candidate Generation and Ranker algorithms are.
- We focus on recommender systems with implicit feedback [5] instead of explicit user-item rating information, in which the interactions between other users and the systems are private, and cannot be used as attackers' background knowledge. Attackers can only crawl basic item information like item title, item description and item popularity (the sales volume of an item).
- Attackers can only use available signals like the number of PV on the pre-defined target items to improve the attack strategy under the black-box setting. More specific information like the user-item preference scores given by the Ranker algorithm in a recommender system is agnostic to attackers.

Attacker's capability. When attacking real-world recommender systems, attackers can perform different operations, like click and purchase, on items by order, which means that some fake behavior data is injected into the history log of a system. Attackers can also do some operations like changing the attributes, but we do not discuss these cases. Other different types of attacks like removing logs from the recommender system, which require strict permission, are not allowed here.

As illustrated in Figure 2, there is an attack agent in PoisonRec. It interacts with a black-box recommender system through generated fake user behavior data, and then improves the attack strategies based on available reward signals. For convenience, we use the click behavior for the attack trajectory in the following parts.

B. Problem Definition

N is the number of attackers. I is the original item set in a recommender system, and I_t is the pre-defined target item set. $\tau = \{a_0, a_1, ..., a_{T-1}\}$ is a T-length attack trajectory. The possible discrete space that an attacker may search is in the form of:

$$\tau = (a_0, a_1, ..., a_{T-1}),\ a_t \in I \cup I_t \tag{2}$$

where a_t refers to the sampled item to be clicked at step t, and $t \in [0, 1, ..., T-1]$. After receiving all attackers' fake trajectories $\boldsymbol{\tau} = [\tau^0, \tau^1, ..., \tau^{N-1}]$, they can be injected into a recommender system. Then, the goal is to find the policy network $\pi_\theta(\cdot)$ of which $R(\boldsymbol{\tau})$ is maximized:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau^i \sim \pi_\theta(\cdot)}\{R(\boldsymbol{\tau})\} \tag{3}$$

where τ^i is sampled from the policy network $\pi_\theta(\cdot)$, $i \in [0, 1, ..., N-1]$, and $R(\boldsymbol{\tau})$ is the reward function, which will be further introduced in Equation 4.

C. Model Design

We formulate the attack trajectory as an MDP in reinforcement learning [20], [30], [31]. Since the N attackers are homogeneous, we require that they share the same policy network $\pi_\theta(\cdot)$ as in [32]. In the following, for convenience, we discuss the model design with a single attacker ($N = 1$), while the definition can be expanded to multiple attackers in a trivial way.

State. S is the state space, and we use $s_t = \{u, a_0, a_1, ..., a_{t-1}\} \in S$ to represent the state at step t, which consists of the current attacker u and all the selected sequential actions (items) so far.

Action. A is the action space, and we have $|A| = |I| + |I_t|$. The action at time step t is to predict the item a_t that needs to be clicked, and a_t belongs to $I \cup I_t$.

Reward. As discussed in the Attacker's Goal in Section III-A2, the number of PV ($RecNum$) is a realistic and available evaluation indicator that can be used as the reward signal to guide the optimization process of PoisonRec. The reward function is defined as:

$$R(\tau) = \sum_{t=0}^{T-1} \gamma^t R_t(s_t, a_t) = RecNum \tag{4}$$

where $\tau = \{a_0, a_1, ..., a_{T-1}\}$ is the sampled attack trajectory, $R_t(s_t, a_t)$ is the received reward at step t, and γ is the discount rate. $RecNum$ is calculated after the complete attack trajectory is injected into the recommender system, and we set $\gamma = 1.0$, $R_t(s_t, a_t) = 0,\ 0 \le t \le T-2$ and $R_t(s_t, a_t) = RecNum,\ t = T-1$. With multiple attackers, the set of sampled trajectories $\boldsymbol{\tau}$ is injected into a recommender system together, and we have $R(\boldsymbol{\tau}) = RecNum$, which is then used to direct the training process of PoisonRec.

Policy Network. There are two parameterized neural networks, an LSTM network [33] and a deep neural network (DNN), in the policy network $\pi_\theta(\cdot)$ used here. In practice, given the state s_t=$\{u, a_0, a_1, ..., a_{t-1}\}$, we first exploit the LSTM network to embed it into a fixed-length hidden variable. Formally, we define:

$$h_t = \text{LSTM}(e_u, e_{a_0}, e_{a_1} ..., e_{a_{t-1}}) \tag{5}$$

where a_t represents the sampled item at step t, e_u and e_{a_t} means the feature representation for user u and item a_t, respectively. We let $|h| = |e|$, and set the size of all hidden layers in LSTM to be equal to $|e|$.

Then we sample next action a_t from the policy network, whose output follows a multinomial distribution on the action space A. For a given h_t, we sample a_t according to:

$$a_t \sim \pi_\theta(\cdot|s_t) = \frac{exp(\mathcal{D}(h_t) \cdot e_j)}{\sum_{j \in I \cup I_t} exp(\mathcal{D}(h_t) \cdot e_j)} \tag{6}$$

where e_j is the feature representation of item j, $\mathcal{D}(\cdot)$ is a DNN whose output dimension is equal to $|e_j|$, and $\mathcal{D}(h_t) \cdot e_j$ is the point-wise product between them. More specifically, we adopt a 2-layer DNN with $Relu$ as the activation function for $\mathcal{D}(\cdot)$, and the size of each layer is also set to $|e|$.

Algorithm 1 PoisonRec Framework

Input: policy network $\pi_\theta(\cdot)$, learning rate α, the number of attackers N, the length of an attack trajectory T, the number of sampled training examples in each training step M, the batch size B, and the number of epochs K.
output: the learned policy network $\pi_\theta(\cdot)$.
function POISONREC($\pi_\theta(\cdot)$, α, N, T, M, B, K)
    Initialize the parameters in the policy network $\pi_\theta(\cdot)$.
    for $step \leftarrow 1, 2, ...$ **do**
        for each $m < M$ **do**
            Sample $\tau^u \leftarrow \{a_0, a_1, ..., a_{T-1}\}$ for each attacker u from the policy network $\pi_\theta(\cdot)$.
            Get the poisoning dataset $D^p \leftarrow \boldsymbol{\tau}$ that consists of the N attackers' trajectories.
            DataPoisoning(D^p).
            $R(\boldsymbol{\tau}) \leftarrow RecNum$ according to Equation 1.
            Store the current example.
        end for
        for each $k < K$ **do**
            Sample a batch of examples where $B \le M$.
            Normalize $\boldsymbol{R}(\boldsymbol{\tau})$ according to Equation 8.
            Update $\pi_\theta(\cdot)$ with normalized rewards following Equation 7.
        end for
    end for
end function

function DATAPOISONING(D^p)
    // Fake data works in a black-box recommender system, which is agnostic to the attacker.
    Reload the Ranker R.
    Update R with D^p.
end function

D. Model Solving

Considering the expensive interaction cost, we exploit the highly popular PPO method in PoisonRec, which can obviously improve the efficiency of sample utilization.

We adopt the *clipped surrogate objective* in PPO [20], and get the update rule of $\pi_\theta(\cdot)$:

$$\theta = \theta + \alpha \min \left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} R^i(\boldsymbol{\tau}), \right. \\ \left. cilp(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1-\epsilon, 1+\epsilon) R^i(\boldsymbol{\tau}) \right) \nabla \log \pi_\theta(a_t|s_t) \tag{7}$$

where α is the learning rate, $p_\theta(a_t|s_t)$ and $p_{\theta'}(a_t|s_t)$ are the sample probabilities of a_t given s_t under the current parameters θ and the old parameters $\theta^{'}$, respectively. $R^i(\boldsymbol{\tau})$ could be the $RecNum$ within a period of time after the recommender system is poisoned by the i-th attack trajectory set $\boldsymbol{\tau}$ in a training batch, where $\boldsymbol{\tau}$ consists of N attackers' trajectories. ϵ $(= 0.1)$ is the hyperparameter used to clip the probability ratio defined in PPO, and $clip(x, a, b)$ limits x between $[a, b]$.

Note that, $R(\boldsymbol{\tau})$ is usually a large discrete number, leading to the difficulty of convergency in a reinforcement learning algorithm. Here, we normalize $\boldsymbol{R}(\boldsymbol{\tau}) = [R^i(\boldsymbol{\tau})]_{0 \le i \le B-1}$ in a sampled batch with Normal (Gauss) Distribution:

$$\hat{R}^i(\boldsymbol{\tau}) = \frac{R^i(\boldsymbol{\tau}) - \mu(\boldsymbol{R}(\boldsymbol{\tau}))}{\sigma(\boldsymbol{R}(\boldsymbol{\tau}))} \tag{8}$$

where $\mu(\boldsymbol{R}(\boldsymbol{\tau}))$ is the mean of $\boldsymbol{R}(\boldsymbol{\tau})$, $\sigma^2(\boldsymbol{R}(\boldsymbol{\tau}))$ is the variance of $\boldsymbol{R}(\boldsymbol{\tau})$, and B is the batch size. $\hat{\boldsymbol{R}}(\boldsymbol{\tau}) = [\hat{R}^i(\boldsymbol{\tau})]_{0 \le i \le B-1}$ is the normalized rewards.

The complete training process is described in Algorithm 1. We also give a kind of possible attack scenario that might happen in a real online recommender system in function $DataPoisoning(D^p)$.

E. Optimization Through Biased Complete Binary Tree

In previous sections, we have fully introduced the framework of PoisonRec. Benefitting from the model-free reinforcement learning architecture, PoisonRec can be used to adaptively attack various recommender systems.

However, it suffers the low convergency problem when directly applied to learn attack strategies on a large action space ($|A| = |I \cup I_t|$). Intuitively, since we focus on the item promotion attack in this work, some actions on the target items I_t are always needed. If we directly sample items from A $(= I \cup I_t)$ as shown in Equation 6, the probability of sampling the target items at each step is only $\frac{|I_t|}{|I|+|I_t|}$ ($|I_t| \ll |I|$), which is very small, and brings many difficulties to obtain effective attack strategies. Moreover, for a T-length attack trajectory, there are total $|A|^T$ possibilities in the search space [18], [20]. $|A|^T$ is a huge number, and it is difficult for PoisonRec to converge under such a kind of setting.

In this part, we propose an important optimization technique to solve the issues discussed above. We reformulate the action space in PoisonRec through a Biased Complete Binary Tree (BCBT). There are mainly two improvements:

- Priori Knowledge. It could be seen as a priori knowledge that it is necessary to sample target items in the attack strategies. We propose a two-stage sampling process, in which PoisonRec first decides which item set (I or I_t) to click, and then chooses one item from the selected set. It can be seen that we built a two-layer tree for sampling items, so that we obtain a biased sampling probability distribution on items in I and I_t. The probability of sampling target items could be increased to about 0.5 ($\gg \frac{|I_t|}{|I|+|I_t|}$) at the beginning of the training.
- Hierarchical Structure. We argue that a hierarchical action space [34] is beneficial for PoisonRec. It not only largely reduces the sampling time complexity on the action space (Section III-E2), but also effectively speeds up the model convergency (Section III-E3).

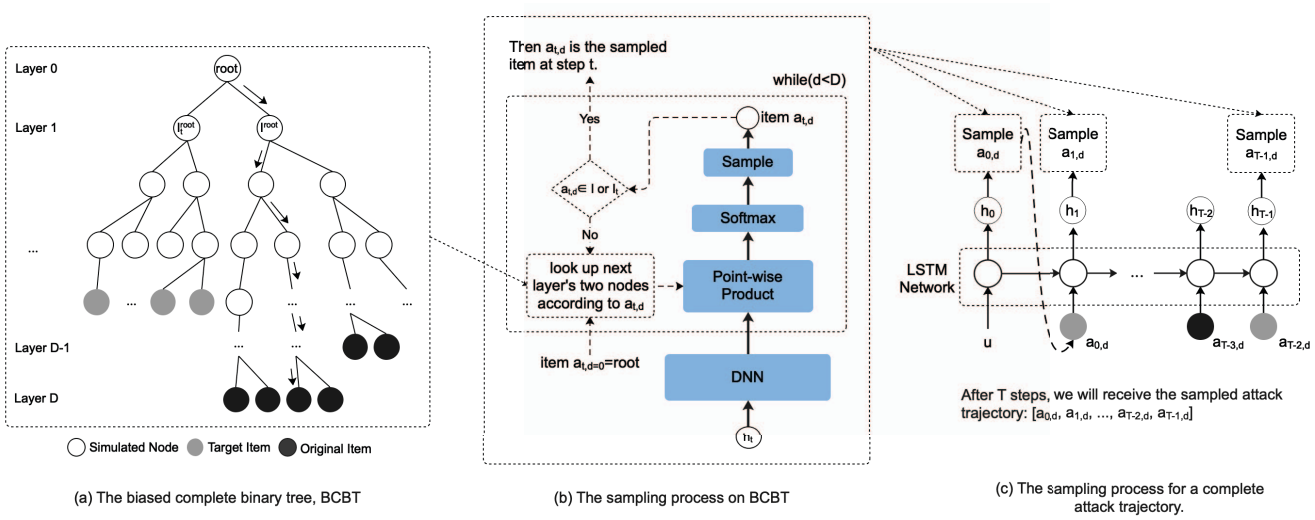
Algorithm 2 The Sampling Process on BCBT

Input: Step t, the hidden variable h_t, the built BCBT.
output: The sampled leaf node (real item) on BCBT.
$d \leftarrow 0$ and $a_{t,d} \leftarrow root$.
while $a_{t,d}$ is not a leaf node ($a_{t,d} \notin I$ *or* I_t) **do**
    Look up next layer's two nodes on BCBT according to $a_{t,d}$.
    Compute point-wise product between $\mathcal{D}(h_t)$ and the two nodes' features, and get $[o_1, o_2]$.
    Sample the next node: $o \sim$softmax$([o_1, o_2])$.
    $d \leftarrow d + 1$.
    $a_{t,d} \leftarrow o$.
end while
Return $a_{t,d}$

In the following, we introduce the construction of BCBT, the sampling on BCBT, and the new update rule of PoisonRec with BCBT in sequence.

1) Construction of BCBT: We adopt the complete binary tree for the hierarchical structure. When constructing BCBT, we first build a complete binary tree (Hierarchical Structure) for I_t and I, respectively, and then merge these two trees with a new root node (Priori Knowledge).

More specifically, we introduce the construction of the complete binary tree for I in detail (the same for I_t). A complete binary tree is a binary tree, and each layer except the last has 2^d nodes (d is the depth of the current layer). Furthermore, the nodes at the last layer are required to be left-aligned. When building a complete binary tree, we need to repeatedly insert nodes from left to right in one layer of the tree. If a layer is full, we insert nodes into the next new layer. The building process is stopped when the number of leaf nodes is equal to $|I|$. All non-leaf nodes are simulated nodes that are used to assist the sampling process on the tree (Section III-E2). For leaf nodes, we assign real items in I to them (details will be discussed later). Since only leaf nodes (real items) have their own features, we randomly initialize



(a) The biased complete binary tree, BCBT

(b) The sampling process on BCBT

(c) The sampling process for a complete attack trajectory.

Fig. 3. The generation of a complete attack trajectory in PoisonRec with BCBT.

trainable feature representation for all simulated no-leaf nodes, whose dimension is equal to that of real items. There is an example of BCBT illustrated on the left of Figure 3.

2) Sampling on BCBT: With a built BCBT, we can sample items following the tree structure more efficiently. The original sampling process in Equation 6 is split into a series of actions on BCBT. At each step t, we need to sample a complete path on BCBT from the root node to a leaf node, and the time complexity of sampling an item can be reduced from $\mathcal{O}(|I \cup I_t||e|)$ to about $\mathcal{O}(\log(|I|)|e|)$. We give the complete pseudo-code of sampling an item on BCBT in Algorithm 2, and there is also an example illustrated in the middle of Figure 3.

3) Updating Rule of PoisonRec with BCBT: For a series of nodes sampled from different layers of BCBT at step t: $[a_{t,1}, a_{t,2}, ..., a_{t,d}]$, we need to update all of them, because one original sampling step is split into multiple ones. The update rule given in Equation 7 is redefined as follows:

$$\theta = \theta + \alpha \min \left(\frac{p_\theta(a_{t,d}|s_t)}{p_{\theta'}(a_{t,d}|s_t)} \hat{R}^i(\boldsymbol{\tau}), \right. \\ \left. cilp(\frac{p_\theta(a_{t,d}|s_t)}{p_{\theta'}(a_{t,d}|s_t)}, 1-\epsilon, 1+\epsilon) \hat{R}^i(\boldsymbol{\tau}) \right) \nabla \log \pi_\theta(a_{t,d}|s_t) \tag{9}$$

where $a_{t,d}$ is the sampled node at step t and level d on the built BCBT, and $d \in [1, 2, ...]$.

Then we discuss how BCBT benefits the model convergency. Each sampled path on BCBT consists of several simulated non-leaf nodes and a real item (leaf node), where all the non-leaf nodes could be viewed as the ancestors of the real item. Following Equation 9, if $p_\theta(a_{t,d}|s_t)$ of each $a_{t,d}$, $d \in [1, 2, ...]$, in a sampled path is updated, the sampling probabilities of other real items that have ancestors in this path will also be affected. To ensure the effectiveness, we emphasize that the design of BCBT should be meaningful and reasonable. Here, we make an assumption:

Assumption 1: Items with close popularity have similar attack properties in a recommender system. For a sampled attack trajectory set, replacing the items with similar ones will not obviously affect its attack performance.

When building BCBT with Assumption 1, there will be clear requirements on the trees built for I and I_t. Taking the tree for I as an example, we need to sort the items in I by their popularity, and then assign them to the leaf nodes by order (from left to right). In such a way, items with closer popularity will share more ancestors. After updating a sampled path on BCBT, the sampling probabilities of items that have higher similarity with the real item in this path will be more affected. These impacts among items are usually positive and can help PoisonRec greatly improve the model convergency. The effectiveness of BCBT will be further discussed and proved in detail in our experiments.

The right of Figure 3 illustrates the generation process for a complete attack trajectory. At step t, PoisonRec models $(u, a_{0,d}, a_{1,d}..., a_{t-1,d})$ through an LSTM network to get h_t (Equation 5), and then samples $a_{t,d}$ with the built BCBT.

F. Complexity Analysis

We analyze the time and space complexity of PoisonRec (Algorithm 1) in this part.

The operations in Algorithm 1 mainly include 1) the attack trajectory sampling based on the policy network $\pi_\theta(\cdot)$, 2) the normalization of a batch of rewards, and 3) the optimization of $\pi_\theta(\cdot)$. We use e to represent the feature representation of users and items, and the size of each layer in the LSTM network and the DNN is set to be equal to $|e|$. To sample T-length trajectories with $\pi_\theta(\cdot)$ for N attackers, PoisonRec needs to perform the calculations in Equations 5 and 6. Specifically, $\mathcal{O}(NT(|e|^2 + |e|^2 + |I \cup I_t||e|))$ time complexity is required (only the major parts are remained), where $\mathcal{O}(|e|^2)$, $\mathcal{O}(|e|^2)$ and $\mathcal{O}(|I \cup I_t||e|)$ are for the calculation in the LSTM network [33], the DNN and the multinomial distribution on the action

space A $(= I \cup I_t)$, respectively. With BCBT, since the original sampling process in Equation 6 is split into a series of actions as discussed in Section III-E2, we do not need to calculate the multinomial distribution on A. The sampling complexity is reduced to $\mathcal{O}(NT(|e|^2 + |e|^2 + \log(|I|)|e|))$. For the normalization of a batch of rewards, it requires $\mathcal{O}(B)$ time complexity, where B is the batch size. For the optimization of $\pi_\theta(\cdot)$ on a batch of examples, PoisonRec needs to calculate the gradient and update the parameters of the two neural networks and the features of related items. With BCBT, real items are assigned to the leaf nodes on it. When sampling items, only about $\log(|I|)$ nodes on BCBT are involved each time. The time complexity for optimization is $\mathcal{O}(BNT(|e|^2 + |e|^2 + \log(|I|)|e|))$. Considering the outer iterations in Algorithm 1, the overall time complexity could be viewed as $\mathcal{O}(S(M + KB)NT(|e|^2 + \log(|I|)|e|))$, where S is the number of training steps.

As for the space complexity, the features of nodes on BCBT and the parameters in the LSTM network and the DNN are stored in the main memory, while others like the space for storing training examples are simply ignored. The leaf nodes (real items) on BCBT have $\mathcal{O}(|I \cup I_t||e|)$ space complexity, and the simulated non-leaf nodes require additional $\mathcal{O}(|I \cup I_t||e|)$. For the two neural networks, both of them need about $\mathcal{O}(|e|^2)$. The overall space complexity is $\mathcal{O}(|I \cup I_t||e| + |e|^2)$.

IV. Experiment

In this section, we first conduct experiments to show the effectiveness of the optimization, BCBT, for PoisonRec. Then we analyze the learned attack strategies on different recommender systems. Finally, we compare PoisonRec with other existing attack methods.

A. Settings

We conduct experiments to evaluate PoisonRec on different real-world datasets, including Steam[1], MovieLens-1m[2] and Amazon[3].

Datasets. Steam, MovieLens-1m and Amazon are all available public datasets.

- **Steam.** Steam is one of the world's most popular game hubs. There are about 6,000 games and 200,000 interactions between users and games. For each user u, we use $\{b_1, b_2, ..., b_k\}$ to represent his/her historical behaviors, and the recommender system can use the first $(k-1)$ behaviors to predict his/her k-th interaction on items. We filter users to guarantee that each user u has at least 3 behaviors $\{b_1, b_2, ..., b_k\}$ $(k \geq 3)$.
- **MovieLens-1m.** MovieLens is collected and made available from the MovieLens web site. MovieLens-1m contains 1 million ratings from 6000 users on 4000 movies. In our experiments, we follow the same pre-processing strategy as [3], which regards all user-item rating pairs as positive samples. In addition, we also remove users with sparse rating behaviors.
- **Amazon.** It is a public widely-used e-commerce dataset collected from Amazon. We select the sub-datasets related to the categories of **Phone** and **Clothing**. For each user u that has k behaviors, we can use his/her historical behaviors to predict the next click item.

For all datasets derived from Steam, MovieLens and Amazon, the features we use are user id and item id. The learned id embedding [3], [35] are regarded as the feature representation of users and items. For item popularity, we simply estimate it through the statistical results in the datasets. We split each dataset into train set, validation set and test set, in which for a user's k behaviors $\{b_1, b_2, ..., b_k\}$, b_k is in the test set, b_{k-1} is in the validation set, and others are in the train set [6], [28]. The statistics are shown in Table II.

Dataset	# Users	# Items	# Samples
Steam	6,506	5,134	180,721
MovieLens	5,999	3,706	943,317
Phone	27,879	10,429	166,560
Clothing	39,387	23,033	239,290

TABLE II
STATISTICS OF ALL DATASETS.

Recommender Systems. For Candidate Generation, we adopt a random method for evaluation efficiency in our experiments, as ranking all items and selecting hundreds of them as the candidates each time through a candidate generation model are time-consuming. Randomly selecting some candidate items and observing whether target items get relatively high scores among them in Ranker can reasonably reflect how well the target items are promoted. For Ranker, we choose 8 different widely-used representative recommendation algorithms as our testbeds, including ItemPop [5], CoVisitation [12], PMF [35][4], BPR [5][5], NeuMF [3][6], AutoRec [27][7], GRU4Rec [6][8], and NGCF [9][9]. ItemPop is a traditional recommendation algorithm, in which items are recommended based on their popularity. CoVisitation is an item-based CF recommendation algorithm. Others have been described in Section II. All of them are popular in the literature of recommender systems.

For experimental evaluation, we need to specify the recommendation process of the recommender system. For a user u, we first produce a candidate item set, and then rank items in it. More specifically, we use randomly-selected 92 original items in I and the 8 target items in I_t to make up a candidate item set each time. For the recommendation results, we assume that each user only views k $(= 10)$ items, and define the top-k items with the highest estimated preference scores in Ranker as L_u. If target items frequently appear in users' recommendation results, there will be a high $RecNum$.

[1]https://www.kaggle.com/tamber/steam-video-games/
[2]https://grouplens.org/datasets/movielens/
[3]http://jmcauley.ucsd.edu/data/amazon/

[4]https://github.com/fuhailin/Probabilistic-Matrix-Factorization
[5]https://github.com/gamboviol/bpr
[6]https://github.com/hexiangnan/neural_collaborative_filtering
[7]https://github.com/gtshs2/Autorec
[8]https://github.com/hidasib/GRU4Rec
[9]https://github.com/xiangwang1223/neural_graph_collaborative_filtering

Attack baselines. We compare PoisonRec with 6 different attack methods. Popular Attack, Random Attack, Middle Attack and PowerItem are heuristic methods, while ConsLOP and AppGrad are learning-based attack methods. Note that, PowerItem and ConsLOP require the system log in their attack background knowledge, and we involve them as our competitors to better illustrate the advantages of PoisonRec.

- **Popular Attack.** Popular Attack refers to injecting fake click behaviors on target and popular items. We select the top $k\%$ (*e.g.*, k=10) of the most popular items from I into a popular item set I_p. In our experiments, we require randomly clicking a target item in I_t and a popular item in I_p alternately.
- **Random Attack.** Similar to Popular Attack, Random Attack means randomly clicking an item in I and a target item in I_t alternatively.
- **Middle Attack.** We further give Middle Attack, in which we randomly select an item from I_t, I_p or $(I \setminus I_p)$ at each step. An important feature of this method is that it may click several target items continuously.
- **Power Item Attack.** Power Item Attack [13] (PowerItem for short) selects influential items first when attacking a recommender system, and we use the *in-degree centrality* [13] for the power item selection, which performs well in the original paper. In our experiments, we require clicking power items and target items alternately in PowerItem.
- **ConsLOP.** ConsLOP is built upon the specific non-personalized recommender system CoVisitation [12]. It models the attack as a *constrained linear optimization problem* (ConsLOP), and can determine: 1) which items the attacker should inject fake co-visitations to, and 2) how many fake co-visitations should be injected to each item. The $N * T$ fake user behaviors in this paper are redefined as $\frac{N*T}{2}$ co-visitations, where a co-visitation means one click behavior on a target item and one following click on a selected original item. ConsLOP is a single item promotion attack method, so we only remain one target item in I_t to perform the attack.
- **AppGrad.** AppGrad [17] initializes a rating matrix $\widetilde{\boldsymbol{M}}$ for N attackers through a generative adversarial network that is trained on the collected user-item rating data. Then $\widetilde{\boldsymbol{M}}$ is iteratively updated by approximately calculating the gradient of $f(\widetilde{\boldsymbol{M}})$ until the model converges, where $f(\widetilde{\boldsymbol{M}})$ is the pre-defined loss function for attack. We set $f(\widetilde{\boldsymbol{M}}) = -RecNum$. In addition, to apply this method in our experiment settings, we make three necessary changes. First, we consider attacking black-box recommender systems with implicit feedback. Thus, we do not generate rating scores, but use discrete behaviors that are sampled with our priori knowledge to initialize $\widetilde{\boldsymbol{M}}$. Second, we use $\widetilde{\boldsymbol{M}}_{ij}$ to represent that user i takes $\widetilde{\boldsymbol{M}}_{ij}$ clicks on item j, and only remain T behaviors for each attacker for fair comparison (the same setting with all competitors). Third, AppGrad does not model the order of behaviors, and we randomly decide it.

Implementation Details. For all the recommendation algorithms and the compared attack methods, without additional statements, we use the default parameters given in their papers. For PoisonRec, we set the size of all layers to 64, and use Adam as the optimizer with a learning rate $\alpha = 2e-3$. When running Algorithm 1, we set $M = B = 32$, $K = 3$. For the number of attackers and the length of each attack trajectory, we set $N = 20$ and $T = 20$ by default.

B. Effectiveness of BCBT

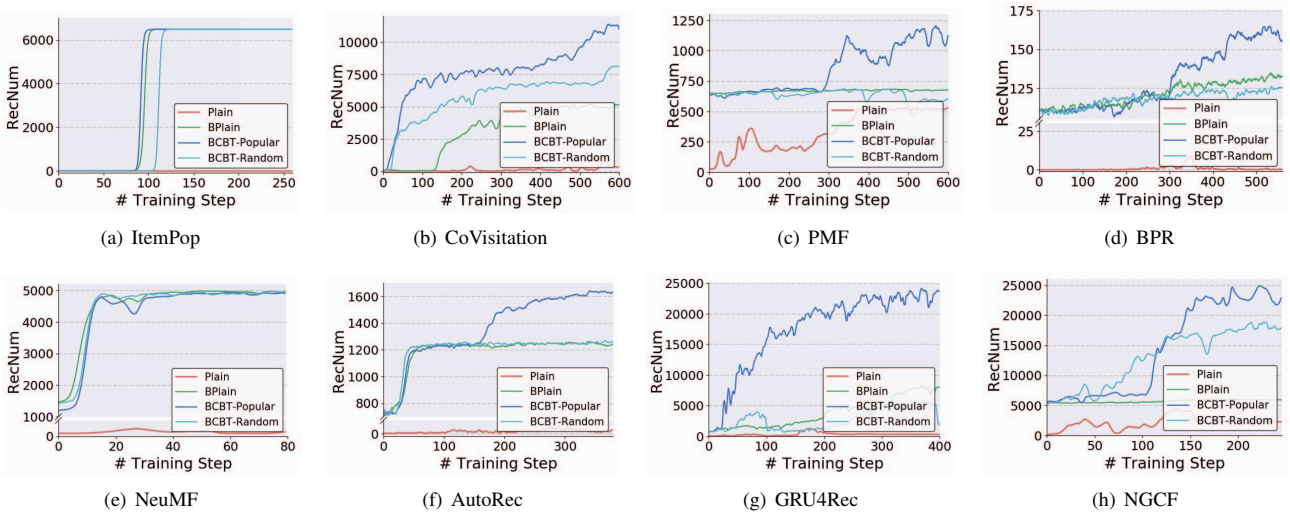
We study the effectiveness of BCBT. Specifically, we compare four different types of designs.

- Plain. It is the basic design of the action space given in Section III-C, with which we need to directly sample items from A as illustrated in Equation 6.
- BPlain. We give BPlain to show the advantages of the priori knowledge, in which we first choose an item set between I_t and I, and then determine which item to click in the selected item set.
- BCBT-Popular. We propose to reformulate the action space through BCBT (with the priori knowledge and the hierarchical structure) for better attack performance in Section III-E. With Assumption 1, we put items with close popularity together, and use BCBT-Popular to represent it.
- BCBT-Random. In BCBT-Random, we randomly assign real items to the leaf nodes on BCBT, and use it to test the rationality of Assumption 1.

At first, compared with the basic Plain, we test the efficiency advantage of BCBT, which greatly reduces the time complexity of Algorithm 1 as discussed in Section III-F. We focus on the time used by PoisonRec and perform experiments on different sizes of item sets, from $3,000$ to $30,000$. As we expect, PoisonRec with BCBT always runs faster than that with Plain, especially on large item sets because of its logarithmic time complexity related to $|I|$. Specifically, on an item set of size $3,000$, PoisonRec with BCBT spends about $1.41s$ to perform a complete training step, while PoisonRec with Plain takes $1.93s$. When the size of the item set reaches $30,000$, the time cost of them increases to $2.33s$ and $15.69s$, respectively. PoisonRec with BCBT runs more than 6 times faster than PoisonRec with Plain, which obviously shows that BCBT has advantages on efficiency.

Then we test the benefits of BCBT-Popular on attacking different recommender systems through equipping PoisonRec with these four different designs on the action space, respectively. Taking the results on Steam as an example, we illustrate how the model converges with the training process in Figure 4.

Advantages of the Priori Knowledge. We say that when performing the item promotion attack in a recommender system, some interactions on the target item set I_t are always needed. The basic design in Plain leads to very a small probability of sampling the items in I_t. Consequently, the generated attack strategies always perform poorly, and there is only a small chance of receiving obvious reward signals.



(a) ItemPop (b) CoVisitation (c) PMF (d) BPR

(e) NeuMF (f) AutoRec (g) GRU4Rec (h) NGCF

Fig. 4. The attack performance when attacking different recommender systems on Steam with the training process.

Within a limited number of training steps, PoisonRec with it performs not well. We build a two-layer tree structure in BPlain, with which we can decide which item set to click (I_t or I) at first. Under such a kind of setting, PoisonRec can find more effective attack strategies at the beginning of the training. From Figure 4, we can see that PoisonRec with BPlain always has a better initialization for the training, and can achieve higher $RecNum$.

Advantages of the Hierarchical Structure. We also introduce a hierarchical structure for the action space. The original item sampling process is split into a series of actions on it, and the update rule in Equation 7 is also redefined as Equation 9, which brings potential to speed up the model convergency as discussed in Section III-E3. With Assumption 1, the impacts among items are meaningful and helpful. In our experiments, PoisonRec with BCBT-Popular gets obvious advantages on the model performance and convergency efficiency.

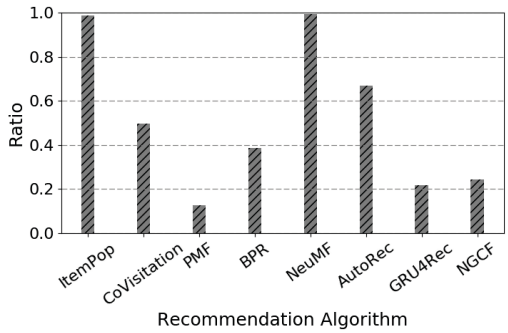


Fig. 5. The ratios of the clicks on I_t to the total clicks for the learned attack strategies on different recommendation algorithms.

We also observe that BPlain gets similar results with BCBT-Popular when attacking ItemPop and NeuMF. We make further investigation, and calculate the ratios of the clicks on I_t to the total clicks for the learned attack strategies of PoisonRec with BCBT-Popular on different recommendation algorithms in Figure 5. For ItemPop and NeuMF, we even receive ratios close to 1.0, which means that clicking items in I_t only is already a good attack strategy on them. Compared with BPlain, BCBT-Popular is further equipped with the hierarchical tree structure, which mainly benefits the exploration on the items in I that have different popularity. If we only need to click the items in I_t that consists of 8 new items, it will be hard for BCBT-Popular to utilize the advantages of the hierarchical tree structure, so BPlain and BCBT-Popular achieve similar performance on ItemPop and NeuMF. When attacking other recommender systems, there are more than 20% clicks on target items most of the time, which experimentally demonstrates the necessity of building a biased probability distribution on items in I_t and I with the priori knowledge.

Besides, to test the rationality of Assumption 1, we give BCBT-Random. In our experiments, PoisonRec with it performs worse than PoisonRec with BCBT-Popular, which clearly illustrates the importance of building a meaningful and reasonable hierarchical tree structure. Gathering items with close popularity (Assumption 1) may not be the optimal solution, but it can bring us considerable advantages. In the following, we adopt PoisonRec with BCBT-Popular by default.

C. Attack Strategies Analysis

In this part, we analyze the learned attack strategies on different recommendation algorithms in detail. Here, we also take the results on Steam as an example. We first visualize all items based on the learned id embeddings in different recommendation algorithms through t-distributed stochastic neighbor embedding (t-SNE) [36][10], and then illustrate the learned attack strategies by circling the clicked items in Figure 6, while ignoring the click order for simplicity. Nodes with different colors are items that have different popularity,

[10]t-SNE is a common technique for dimensionality reduction, and is particularly well suited for the visualization of high-dimensional data.

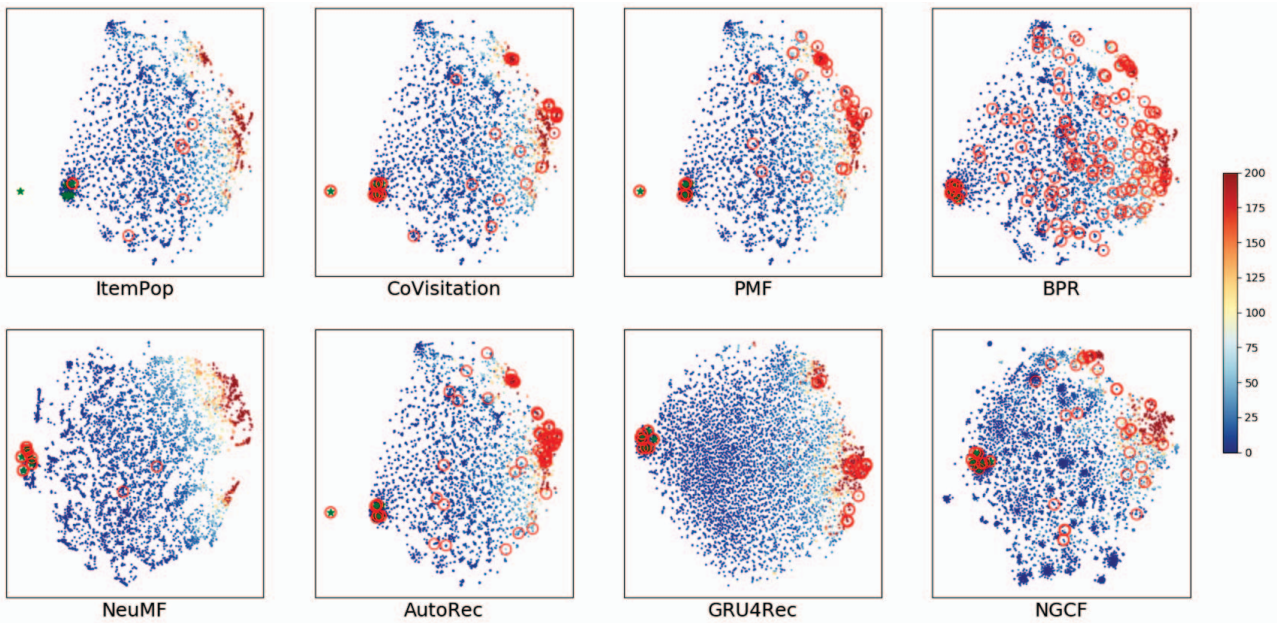


Fig. 6. The visualization of the items and the learned attack strategies on different recommendation algorithms.

and target items are represented by green stars (generally distributed on the left side of each sub-figure). In such a way, we can clearly see the distributions of clicked items in the learned attack strategies on different recommendation algorithms. For ItemPop, CoVisitation and AutoRec, we use the learned item id embedding in PMF for them.

For ItemPop, it is a traditional recommendation algorithm, in which items with higher popularity are more likely to be recommended. We can see that the learned attack strategy of PoisonRec on it involves only one target item. PoisonRec indeed learns to repeatedly click the one selected target item in I_t, so that the target item looks like a popular one, and can receive much more exposure than before. CoVisitation is an item-based CF algorithm. It uses users' sequential behaviors to construct an item-to-item graph, and then calculates item similarities based on this graph. When attacking it, PoisonRec prefers to click target items and original items simultaneously. The learned attack strategies on ItemPop and CoVisitation are obviously different. Putting aside the visualization in Figure 6, when we directly observe the learned attack strategies, we further find that PoisonRec learns to click target items and original items alternately on CoVisitation.

For other recommendation algorithms, PoisonRec also shows the preference for clicking target items and original items at the same time. On PMF, AutoRec and GRU4Rec, PoisonRec prefers to click target items and popular items. However, with limited cost to perform the attack, clicking the most popular items is not always a good solution. More popular items are generally more robust, which means that more fake data may be required to poison them. When attacking BPR and NGCF, clicking some of the slightly popular items receives better performance. As for NeuMF, clicking target items only seems an effective way to poison it.

Overall, there are notable differences among the learned attack strategies on different recommender systems. We need to determine: 1) the items to click, 2) the times to click on each selected item, 3) the order of clicks (*e.g.*, on GRU4rec), etc. **Fixed attack strategies cannot always achieve good results.** We model the sequential attack trajectory as the MDP, and PoisonRec can adaptively learn effective attack strategies on different recommender systems.

D. Comparison with Other Attack Methods.

We conduct the comparison of different attack methods, and Table III reports the results of $RecNum$ on all test datasets and recommender systems.

Random Attack, Popular Attack, Middle Attack and PowerItem are heuristic attack methods. To compare these 4 methods, we use the 8 recommendation algorithms and the 4 datasets to construct 32 testbeds, and then count the number of times that each attack method performs the best. Note that, since there is a high average item frequency ($943317/3706 \approx 254$) in MovieLens, it is difficult for all methods to build a fake popular item with limited cost in this dataset. All methods receive $RecNum = 0$ when attacking ItemPop on MovieLens, so we do not include it in the statistical results.

The specific results are shown in Table IV. We find that we cannot simply say which heuristic method is the best. Different methods receive obvious different attack results on different recommendation algorithms and datasets. Most of the time, Popular Attack and Middle Attack perform better, but they also receive lower $RecNum$ than Random Attack and PowerItem in some cases.

For ConsLOP, it is particularly designed for attacking CoVisitation. With full knowledge about the recommender

Dataset	Method	ItemPop	CoVisitation	PMF	BPR	NeuMF	AutoRec	GRU4Rec	NGCF
Steam	Random	7	278	653	114	1,362	667	783	2,203
	Popular	6	1,895	541	106	599	738	1,331	1,093
	Middle	2	530	609	116	449	643	1,347	798
	PowerItem	6	1,794	534	107	588	661	1,401	852
	ConsLOP	8	4,715	633	121	648	683	2,401	1,699
	AppGrad	5,421	135	686	122	2,914	1,256	5,052	8,094
	PoisonRec	**6,496**	**10,917**	**1,211**	**163**	**4,994**	**1,643**	**24,319**	**25,013**
MovieLens	Random	0	492	2,282	2,012	412	11,117	236	6
	Popular	0	1,420	4,237	1,927	10	10,471	1,367	13,015
	Middle	0	120	2,415	2,055	10	10,896	282	12
	PowerItem	0	1,136	4,286	1,972	545	10,691	1,264	11,230
	ConsLOP	0	**2,162**	4,246	1,624	2	11,578	714	11,493
	AppGrad	0	118	3,580	2,044	2,604	12,124	4372	24
	PoisonRec	0	1,552	**7,050**	**2,442**	**2,742**	**12,472**	**18,525**	**21,577**
Phone	Random	2,020	464	10,432	4,282	4,794	2,822	2,826	8,784
	Popular	2,409	2,368	9,939	3,846	1,290	3,885	2,454	8,048
	Middle	4,946	208	9,050	3,565	5,981	2,627	3,699	9,552
	PowerItem	2,358	1,824	10,880	3,779	1,978	3,046	944	7,408
	ConsLOP	2,074	**6,234**	10,787	4,099	1,648	4,694	2,858	9,136
	AppGrad	61,792	131	11,238	4,187	26,800	4,700	4,072	10,852
	PoisonRec	**82,032**	5,683	**12,195**	**4,530**	**28,646**	**4,873**	**8,513**	**12,324**
Clothing	Random	54,820	413	1,848	2,827	4,656	11,270	7,786	7,376
	Popular	53,265	1,262	1,704	2,803	2,424	12,032	11,827	9,468
	Middle	61,156	125	1,699	3,077	4,733	9,768	12,005	5,672
	PowerItem	57,508	686	1,810	2,678	2,525	11,664	7,234	8,592
	ConsLOP	52,921	**3,312**	1,814	2,842	2,294	11,981	15,490	7,524
	AppGrad	180,432	62	3,216	3,816	8,808	13,472	13,424	11,090
	PoisonRec	**218,275**	2,239	**3,363**	**4,656**	**12,592**	**14,245**	**22,013**	**14,391**

TABLE III
RECNUM OF DIFFERENT ATTACK METHODS WHEN ATTACKING DIFFERENT RECOMMENDER SYSTEMS ON 4 DATASETS.

Method	Steam	MovieLens	Phone	Clothing	All
Random	4	1	1	1	7
Popular	2	3	2	3	10
Middle	1	1	4	4	10
PowerItem	1	2	1	0	4

TABLE IV
THE NUMBER OF TIMES THAT EACH ATTACK METHOD ACHIEVES THE BEST PERFORMANCE AMONG THE FOUR HEURISTIC METHODS.

system, there is no surprise that it can achieve high attack performance on CoVisitation. Specifically, ConsLOP achieves the best results on MovieLens, Phone and Clothing, and performs better than other attack methods except PoisonRec on Steam. For other recommendation algorithms, we use the learned attack strategies of ConsLOP on CoVisitation to attack them. The performance is not as good as before. ConsLOP is particularly optimized on CoVisitation, and the learned attack strategies just achieve limited attack performance on other different recommendation algorithms.

Indeed, ConsLOP is an attack method designed for promoting a single target item [12]. In our experiments, we randomly select one target item from I_t, and then use ConsLOP to calculate the near-optimal solution. However, the constraint of performing attacks on a single target item limits the performance of it. In PoisonRec, we design a relatively comprehensive search space. It is directly optimized on multiple target items, and has opportunities to allocate the limited attack cost ($N*T$ fake user behaviors) on different target items more effectively, leading to better attack performance. PoisonRec also shows the ability to promote multiple target items simultaneously when attacking other recommender systems. For example, on ItemPop, PoisonRec successfully learns to promote 3 and 6 target items at the same time on Phone and Clothing, respectively, and achieves high performance.

For AppGrad, PoisonRec also shows obvious advantages over it. Though AppGrad achieves comparable results with PoisonRec on ItemPop and NeuMF sometimes, PoisonRec performs much better on all other recommender systems. On ItemPop and NeuMF, clicking target items only is already a good attack strategy (Figure 5). We randomly sample discrete behaviors with our priori knowledge to initialize the matrix $\widetilde{M}$ in AppGrad, and nearly half of the clicks are performed on target items at the beginning of the training. When attacking these two recommender systems, AppGrad easily converges to click target items only, and achieves relatively high performance. However, on other recommender systems, the performance of AppGrad is obviously inferior to that of PoisonRec. AppGrad does not incorporate the sequential information on behaviors, which is a serious limitation. When attacking recommender systems that are sensitive to the order of behaviors like CoVisitation and GRU4Rec, it performs not well. In addition, PoisonRec directly explores the attack strategies in the discrete search space, while AppGrad optimizes $\widetilde{M}$ in the continuous numerical space. At each iteration, the elements in $\widetilde{M}$ need to be rounded to the closest integers for obtaining discrete click behaviors [17], which affects AppGrad's performance. More importantly, both PoisonRec and AppGrad need to explore attack strategies in a very large search space, which is related to the number of items in the recommender system. Compared with AppGrad, PoisonRec is further enhanced by BCBT, and

can find high-performance attack strategies more efficiently.

V. Conclusions

In this work, we perform a systematic study on attacking different black-box recommender systems. With very limited knowledge about the attacked system, we exploit the reinforcement learning technique and propose a flexible adaptive data poisoning attack framework, PoisonRec, to efficiently find effective attack strategies. In addition, an optimization technique BCBT is devised to strengthen PoisonRec's attack ability. We conduct extensive experiments on 8 representative recommendation algorithms and 4 different public datasets. The results show that BCBT brings obvious advantages on the model performance and convergency efficiency, and PoisonRec can achieve better attack performance in most cases compared with existing attack methods.

Acknowledgment

This work was partially supported by NSFC under Grant No.61832001, and Alibaba-PKU joint Program.

References

[1] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre, "Collaborative recommendation: A robustness analysis," *ACM Transactions on Internet Technology (TOIT)*, vol. 4, no. 4, pp. 344–377, 2004.

[2] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[3] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 2017, pp. 173–182.

[4] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM conference on recommender systems*. ACM, 2016, pp. 191–198.

[5] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 2009, pp. 452–461.

[6] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[7] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2016, pp. 353–362.

[8] H. Zhao, Q. Yao, J. Li, Y. Song, and D. L. Lee, "Meta-graph based recommendation fusion over heterogeneous information networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 635–644.

[9] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," *arXiv preprint arXiv:1905.08108*, 2019.

[10] S. K. Lam and J. Riedl, "Shilling recommender systems for fun and profit," in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 393–402.

[11] 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 (TOIT)*, vol. 7, no. 4, p. 23, 2007.

[12] G. Yang, N. Z. Gong, and Y. Cai, "Fake co-visitation injection attacks to recommender systems." in *NDSS*, 2017.

[13] C. E. Seminario and D. C. Wilson, "Attacking item-based recommender systems with power items," in *Proceedings of the 8th ACM Conference on Recommender systems*. ACM, 2014, pp. 57–64.

[14] C. E. Seminario and D. C. Wilson, "Assessing impacts of a power user attack on a matrix factorization collaborative recommender system." in *FLAIRS Conference*, 2014.

[15] M. Fang, G. Yang, N. Z. Gong, and J. Liu, "Poisoning attacks to graph-based recommender systems," in *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 2018, pp. 381–392.

[16] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," in *Advances in neural information processing systems*, 2016, pp. 1885–1893.

[17] K. Christakopoulou and A. Banerjee, "Adversarial attacks on an oblivious recommender," in *Proceedings of the 13th ACM Conference on Recommender Systems*. ACM, 2019, pp. 322–330.

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[19] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[21] Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv, "Adaptive dynamic bipartite graph matching: A reinforcement learning approach," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1478–1489.

[22] Z. Li, J. Song, S. Hu, S. Ruan, L. Zhang, Z. Hu, and J. Gao, "Fair: Fraud aware impression regulation system in large-scale real-time e-commerce search platform," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1898–1903.

[23] K. Lolos, I. Konstantinou, V. Kantere, and N. Koziris, "Adaptive state space partitioning of markov decision processes for elastic resource management," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 191–194.

[24] Y. Hu, Q. Da, A. Zeng, Y. Yu, and Y. Xu, "Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application," *arXiv preprint arXiv:1803.00710*, 2018.

[25] F. Garcin, B. Faltings, O. Donatsch, A. Alazzawi, C. Bruttin, and A. Huber, "Offline and online evaluation of news recommender systems at swissinfo. ch," in *Proceedings of the 8th ACM Conference on Recommender systems*. ACM, 2014, pp. 169–176.

[26] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[27] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 111–112.

[28] C. Zhou, J. Bai, J. Song, X. Liu, Z. Zhao, X. Chen, and J. Gao, "Atrank: An attention-based user behavior modeling framework for recommendation," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[29] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based Recommendation with Graph Neural Networks," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, ser. AAAI '19, vol. 33, no. 1, 2019, pp. 346–353.

[30] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[31] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[32] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 2017, pp. 66–83.

[33] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth annual conference of the international speech communication association*, 2014.

[34] R. Takanobu, T. Zhuang, M. Huang, J. Feng, H. Tang, and B. Zheng, "Aggregating e-commerce search results from heterogeneous sources via hierarchical reinforcement learning," in *The World Wide Web Conference*. ACM, 2019, pp. 1771–1781.

[35] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *International Conference on Neural Information Processing Systems*, 2007.

[36] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.