Exploring the Impact of Black-Box Adversarial Behavior in Graph-based Recommenders

Alan Flores-López, Divyahans Gupta, Leo Mehr
Stanford University
{alanf94,dgupta2,leomehr}@stanford.edu

ABSTRACT

In this work we explore how modern graph-based recommender systems react under the presence of white and black-box adversarial manipulation. We implement a graph-based recommendation system based on a system deployed at Pinterest [6] and evaluate the efficacy of adversarial agents whose goal is to boost the popularity of a target item – such attacks are known in the literature as "shilling" attacks. Unlike previous studies that focus solely on *white-box* attackers who have visibility into the recommendation algorithm, we also lean into the more realistic scenario of *black-box* attackers. We find that 1) keeping the number of fake item reviews low is of paramount importance to the efficacy of an adversary, that 2) a black-box attacker we construct performs as well as a principled white-box attacker, and that 3) even principled attackers (white-box or black-box) seem to perform no better than baseline approaches, suggesting strong robustness properties in the graph-based recommendation mechanisms we employ.

1 INTRODUCTION

Modern content discovery applications such as Amazon, Netflix, YouTube, Twitter, and Pinterest rely on recommender systems to provide content to users and increase engagement. Many of these systems leverage information from the graph of users and content in order to make better recommendations [3, 6, 10].

Such *graph-based* recommendation systems work well in the presence of honest existing relationships between users and content (e.g. a user's ratings for a piece of content or a user's choice to interact with some content). However, most modern applications with user-based recommendations suffer from *fraudulent activity*. In the case of Amazon, for example, a fraudulent user could create multiple fake accounts, give fake reviews to some set of items, and influence the recommender system. A study from 2015 estimated that 16% of Yelp reviews were fake or suspicious [16].

In this work we study how **graph-based recommender systems** react under *shilling* attacks where a malicious user creates fake profiles and adds fake reviews to boost the popularity of an item. We implement a modern graph-based recommendation system based on Pixie, a real-time recommendation system deployed at Pinterest. Unlike other works which implement only *white-box* attackers that have in-depth knowledge of a recommender system, we focus also on the more realistic scenario of *black-box* or low-knowledge attackers.

Scope. We do not aim to explore the problem of fake user *detection* – there is a rich history of research in this problem already, as we describe in the related work section. We do not consider the case of more traditional collaborative filtering recommender systems either – we only consider graph-based recommendation systems, most of which involve random walks.

Contributions. We contribute three observations. First, keeping the number of fake item reviews low is of paramount importance to the efficacy of an adversary. Second, a black-box attacker we construct performs as well as a principled white-box attacker suggesting that in practice the gap between white-box and black-box attackers may not be too significant. Third, we find that even principled attackers (white-box or black-box) seem to perform no better than baseline approaches, suggesting strong robustness properties in the graph-based recommendation system we employ.

Overview. In the next section we briefly discuss previous work in recommender systems, attacks and defenses against recommender systems, and the specific landscape of graph-based recommender systems. We also provide the necessary background for the rest of the article. After that we present our threat model, where we describe the powers and goals of the shilling attacker we consider. In the next two sections we discuss our methods for constructing recommenders, attackers, and showcase the results of several experiments. We then suggest future work and conclude.

Code. All of our code is public on GitHub [1].

2 BACKGROUND AND RELATED WORK

2.1 Recommender Systems

Recommender systems have a strong history of research dating back to the early 1990s. Several textbooks have emerged in recent years that systematize this knowledge [1, 12, 13, 19]. Recommender systems are usually built upon *collaborative*

[1] https://github.com/alanefl/graph-based-recommender-attacks

filtering or *content-based* approaches. In the former approach, user-item interactions in the past are used to predict (i.e. recommend) new user-item interactions in the future. In the latter approach, specific attributes of users or items are used to generate recommendations. Hybrid approaches that combine both recommendation paradigms are also actively used [1]. Recent years have seen the rise of Deep Learning approaches for the recommendation task, with good promise [22].

2.2 Graph-based Recommender Systems

In this work we focus on *graph-based* recommendation systems. Graph-based recommender systems [2, 3, 6, 8, 21] use a bipartite user-item graph to represent how a given user rates an item. An edge exists between a user and an item only if that user has rated that item and its weight is the rating. Graph-based recommendation systems are based on *random walks*. To recommend an item to a user, the recommender system initiates a random walk starting from the user, following links in the graph according to some specified rule. Recommendations are then based on the nodes seen the most in their random walks.

2.3 Adversaries in Recommender Systems

There has been a substantial body of work regarding adversarial behavior against recommender systems [4, 5, 14, 17, 18]. Most works describe both shilling attacks and defenses against those attacks in traditional collaborative filtering recommender systems. Two survey papers outline the field as a whole [9, 20], providing a good overview of the types and techniques behind most shilling attackers proposed in the literature. The efficacy of attackers varies significantly depending on the dataset and algorithm in question.

Less work has focused solely on studying attackers for strictly *graph-based* recommender systems. Fang et al provide the what seems to be the first systematic work in poisoning attacks to graph-based recommender systems [7]. The authors claim to construct an optimizing attacker that, because of its unique design targeting graph-based recommenders, outperforms more general attackers. However, the optimizing method Fang et al concoct involves knowledge of the *entire* bipartite graph. We extend this line of work by broadening our scope to more realistic attackers that may only have pieces of knowledge about the recommender and its underlying network, and thus may only approximate a principled white-box approach.

2.4 Terminology

For the rest of this report, we use the term *entity-item* network to refer to a typical user-item bipartite graph (because in some cases the entities to which we "recommend" items are not users). We treat an entity-item network $G = (E, I, E')$ as an undirected, weighted bipartite graph. The weight of an edge $(e, i) \in E'$ corresponds to the rating that entity $e \in E$ assigned to item $i \in I$. Throughout the rest of the paper, we interchange *user* with *entity*, *review* with *edge*, and network-specific items names such as *beer* or *movie* with *item*.

3 THREAT MODEL

We consider *shilling* attackers on graph-based recommender systems.

Goals. The ultimate goal of a shilling attacker is to *push* some target item i^* or have it be the top recommendation for all $e \in E$. The success of our shilling attackers is measured by the *hit-ratio* of i^*, which is the percentage of users to which the target item is recommended. A successful attack would see the hit-ratio of i^* rise significantly relative to its original value.

Powers Our attacker can inject N entities and M edges coming from each of the N entities. Besides the M edges, each entity also adds an edge to the target item i^*. In our experiments we choose modest values for N and M, since a real-world attacker incurs the high cost of being caught if the graph is disturbed substantially. As we will see, it benefits the attacker to keep the value of M low.

Knowledge We separate our attackers into two classes, white-box and black-box, according to the prior variables they use to conduct the attack. Our white-box attackers have total access to the recommender graph while the black-box attacker does not have any more insight into the internals of the recommender algorithm than what is immediately obvious from public data on a popular e-commerce site, e.g., the average rating of an item, the number and identity of users that have rated an item, and the recommendations that a recommender gives to attacker-made profiles. The white-box attackers offer a benchmark against which we can measure the efficacy of our black-box attacks.

4 METHODS

In this section we describe our datasets, our recommender, and our attackers. We justify the quality of our recommender as well as our choices of white-box and black-box attackers.

4.1 Datasets

We use three publicly available datasets, MovieLens100k, MovieLens1M [11], and BeerAdvocate data. The MovieLens datasets are standard datasets for recommender systems which contains users (entities) that review movies (items), with each rating between 1 and 5. The BeerAdvocate dataset is composed of users that rate beers on a scale of 0 to 5.

	MovieLens100k	MovieLens1M	BeerAdv.
Entities	943	6,040	33,387
Items	1,682	3,952	66,051
Edges	100,000	1,000,209	1,571,251
Fill	0.0145	0.010	0.0002
Diameter	5	5	8
Cluster Coef.	0	0	0

Figure 1: The three datasets we use and some network characteristics.

We include statistics from these three networks in Figure 1. Note that

$$\text{Fill} = \frac{\#\text{edges}}{\#\text{total possible edges}} \tag{1}$$

The *diameter* is sampled from 500 nodes in the graph, meaning it is a lower bound on the true diameter of the graph. The *clustering coefficient* is always 0 for bipartite graphs, because no triangles exist.

We use MovieLens1M and BeerAdvocate to evaluate our recommender system and MovieLens100k to evaluate our attackers. In order to provide a null-model for our recommender, we also evaluate it on Erdos-Renyi graphs with the same number of entities, items, edges (chosen uniformly at random) as the MovieLens1M and BeerAdvocate datasets. We constructed the weights of the Erdos-Renyi graphs by sampling from the distribution of edge weights of their corresponding "source" datasets.

4.2 Recommender System

Algorithm 1 MiniPixie Random Walk

```
procedure MiniPixie( e, G, α, N, n_p, n_v )
    totSteps = 0, V = {}
    nHighVisited = 0
    while totSteps < N and nHighVisited ≤ n_p do
        entity = e
        steps = SampleWalkLength(α)
        for i = [1: steps] do
            if i ≠ 0 then
                entity = item.randWeightedNeighbor()
            item = currEntity.randWeightedNeighbor()
            V[item]++
            if V[item] == n_v then
                nHighVisited++
        totSteps += steps
    return V
```

The recommender system we use to test adversaries is based on Pixie, a random-walk based algorithm deployed at Pinterest [6]. The idea of the algorithm is quite simple. When given an entity e to recommend items for, the algorithm carries out random walks on the graph beginning from e – the items with the highest visit count on these random walks are returned as recommendations to e. We refer the reader to the original Pixie paper for more detail on the algorithm.

MiniPixie vs. Pixie. There are several differences between Algorithm 1 and algorithm presented in [6].

- MiniPixie uses a custom `SampleWalkLength` function since the authors of Pixie do not disclose their method. We describe our approach for this function below.
- MiniPixie chooses a weighted edge at random in its random walk, unlike Pixie which chooses a *personalized neighbor* at each step.
- Pixie considers recommending to a *set* of entities – we do not consider that approach.
- Pixie first *prunes* the graph before carrying out recommendations on it. We leave the graph as is.

Sampling Walk Length. MiniPixie recommends items to an entity by carrying out weighted random walks with restarts starting at that entity. Before starting each random walk, we need to choose what its length will be (i.e. the number of steps before restarting). The authors of Pixie do not disclose their method for parameterizing this choice based on the value α, so we do the following.

$$\text{currSteps} \sim \mathcal{N}(\mu, \sigma^2) \tag{2}$$

$$\mu = \alpha N \tag{3}$$

$$\sigma = \beta \tag{4}$$

We round each sample to the nearest integer and use that as our current walk length. The motivation is that $\alpha \in [0, 1]$ functions as a knob that tunes the length of individual random walks, while still preserving variability in those choices. Choosing α close to 0 means that the random walks will be very small compared to N. Choosing α close to 1 means that random walks will be very close to N, and thus maybe only one or two restarts may occur. We keep the variance β a tunable parameter (see Section 5 for details).

Evaluating the Recommender. Despite having key differences from the original Pixie algorithm, we argue that MiniPixie is a reasonable algorithm with which to study adversaries. We evaluate MiniPixie against two baseline recommenders: a *random recommender* that returns items uniformly at random from the set of items in the graph, and a *popular item recommender* that selects items at random from the most popular items, where an item's popularity is its weighted degree.

Parameter Settings. We set the popular item recommender to recommend from the top 1000 most popular items. For MiniPixie, we set $n_p = 30$, $n_v = 4$. We use a random walk of length 1000 with $\alpha = 0.01$ and $\beta = 20$. We chose n_p and n_v based on parameters shown to work well in [6]. The total number of steps in our random walks is 1,000 because in [6] the ideal walk lengths are in the 100,000s of steps, but our graphs are orders of magnitude smaller. We choose α and β so that each individual random walk has length 10 on average, with a standard deviation of 20 steps. Note from Figure 1 that the diameters of these graphs are less than 10, so each random walk has a non-zero probability of reaching any item.

Evaluation Metric. We evaluate the recommenders using a recommendation metric we call the *prediction ratio*:

$$P = \frac{1}{|E|} \sum_{e \in E} \frac{1}{|N_w(e)|} \sum_{i \in N_w(e)} 1\{i \in R(G \setminus \{e \to i\}, e, k)\} \quad (5)$$

E is the set of entities in the graph, $N_w(e)$ is the set of neighbors of that entity that have an edge of weight w or more, k is the number of items that the recommendation routine R returns, and the notation $G \setminus \{e \to i\}$ means that we consider the graph G without the edge that connects e and i. Because computing the prediction ratio over the entire set of entities is computationally expensive, we calculate prediction ratios over a subset of entities. We were able to calculate prediction ratios for 150 randomly sampled entities by running experiments on a Google Cloud n1-standard-16 VM instance (16 cores and 60 GB of memory) over a couple of days.

Results. The results of 36 experiments are shown in Figure 2. We only consider predicting links in the graph that have weight 4 or more. MiniPixie dominates among the Top 10 recommendation setting and considerably surpasses the Popular recommender for MovieLens1M. The Popular recommender dominates the Top 1000 recommendation setting. This suggests MiniPixie is able to give more targeted recommendations to users than the other recommenders. Note it is expected that the Popular recommender will achieve a higher success the more recommendations are given, since popular items are most often rated highly and with high frequency. Note also that all three recommenders fare about the same in the corresponding Erdos-Renyi graphs. This implies that MiniPixie takes advantage of the graph structure of the underlying Entity-Item graph for making informed recommendations.

From this discussion we conclude that the MiniPixie algorithm is suitable enough for us to utilize in our attacking experiments. We insist that our purpose here is not to provide a state-of-the-art recommender system, only to construct a suitable one with which to carry out adversarial analysis.

	Top 10	Top 100	Top 1000	Dataset
Random	3.80 %*	5.15%	6.09%	BeerAdvocate
	3.36%*	3.40%	4.17%	ER-BeerAdvocate
	2.40 %	3.90%	25.87%	MovieLens1M
	1.17%	3.08 %	25.86%	ER-MovieLens1M
Popular	4.40%	**11.65**%	**64.80**%	BeerAdvocate
	3.29%	3.50%	4.84%	ER-BeerAdvocate
	2.61%	10.07%	**86.84**%	MovieLens1M
	1.25%	3.18%	29.06%	ER-MovieLens1M
MiniPixie	**6.41**%	5.52%	36.03%	BeerAdvocate
	3.44%*	3.42%	4.04%	ER-BeerAdvocate
	6.83%	**22.66** %	55.83%	MovieLens1M
	1.14%	3.02%	21.26%	ER-MovieLens1M

Figure 2: Evaluation of recommenders. Given an entity with a missing entity-item link, measure how well each recommender includes that missing link in the top k recommendations. Results are shown for 150 randomly sampled entities. Those marked with an asterisk used slightly fewer samples.

4.3 Attackers

We now describe our white-box attackers and our proposed black-box attacker. Note that the sole metric that our attackers aim to maximize is the *hit-ratio* of the target item after the attack; that is, the percent of real users in the graph who receive the target item among their list of recommendations. Each attacker is configured to create N fake users and M fake reviews per fake user.

Random Attacker *(black-box).* Each reviewed item is uniformly chosen at random. Each fake review rating is sampled from a normal distribution fitted to the real ratings. Every fake user gives the target item the maximum possible rating. A similar baseline was implemented in prior literature [7, 15]. The random attacker is not perfectly "black-box", but only requires knowledge of which items exist and the maximum rating – that is, it does not require any knowledge of the network structure.

Average Attacker *(black-box).* Each reviewed item is uniformly chosen at random. Each fake review rating is sampled from a normal distribution with mean equal to the average rating for that item and a standard deviation of 1.1 (roughly rating range/5). Every fake user also gave the target item a max rating review. A similar baseline was implemented in prior literature [7, 15]. The average attacker is suitable for the black-box setting since it only assumes knowledge of the average review rating, which is likely accessible to every user.

High-Degree Attacker *(black/white-box).* The high-degree attacker sorts all the items by degree, which is equal to the number of reviews for each item. The top $N(M-1)$ highest-degree items are given a fake review with the max rating. The reviews are assigned to the N fake users in round-robin fashion. Every fake user also gave the target item a max rating review. This attacker may be considered either a white box or a black box attacker (since knowledge of item degree is often provided openly on platforms, i.e. number of reviews an item has).

Neighbor Attacker *(white-box).* The neighbor attacker chooses an item to review by sampling uniformly from the set of items that are neighbors of the target item in the folded item graph. Every fake user also gives the target item a max rating review. This baseline presumes that recommendations are the product of random walks in the graph. The hypothesis behind this baseline was that installing edges that create more paths between the target item and its "neighbors" would increase the stationary probability of the target item to users overall, especially those who have reviewed the neighbors of the target but not the target item.

Hill-Climbing Attacker *(white-box).* The hill-climbing attacker follows the standard greedy hill climbing algorithm as appropriate for a bipartite graph. At the i-th step, the attacker greedily adds the item that maximizes its total influence set, which is the set of users that are neighboring the set of $i-1$ previously chosen items. The attacker sorts the items in the order that the items are added. The top $N(M-1)$ items are given a fake review with the max rating. The reviews are assigned to the N fake users in round-robin fashion. Every fake user also gave the target item a max rating review. The intuition behind this scheme is that the attacker seeks to optimize her allocation of fake users and fake reviews to those items that have the most influence in the recommender graph.

Random-Walk-Reachability (RWR) Attacker *(white-box).* The RWR attacker computes a reachability score for each item in the recommender graph. Because the recommendation list is generated via random walks, the reachability score attempts to correlate with the likelihood that the item appears in a user's recommendation list. The reachability score of item I is calculated in a Pixie-like fashion: it is the number of unique users reached via MiniPixie weighted random walks from I. However, instead of determining the length of each walk via sampling, we use an exponential decay function $e(P)$ to anneal the influence of the item and stop the walk when it falls below a threshold:

$$e(P) = \exp\left(-f w_P P_{len}\right) \tag{6}$$

where

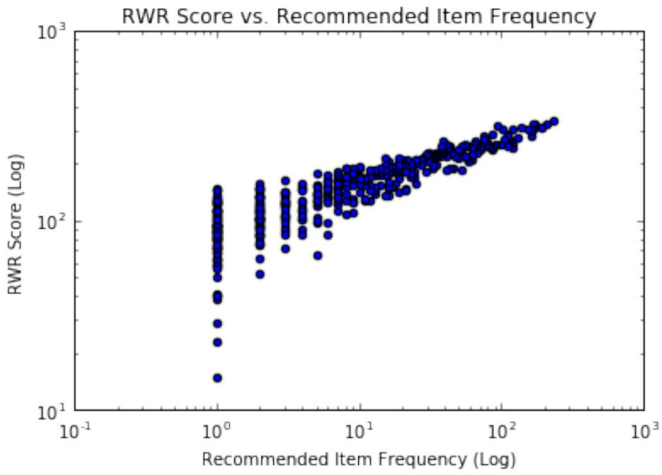

Figure 3: RWR Score is positively correlated with recommendation frequency, so it is an appropriate approximation.

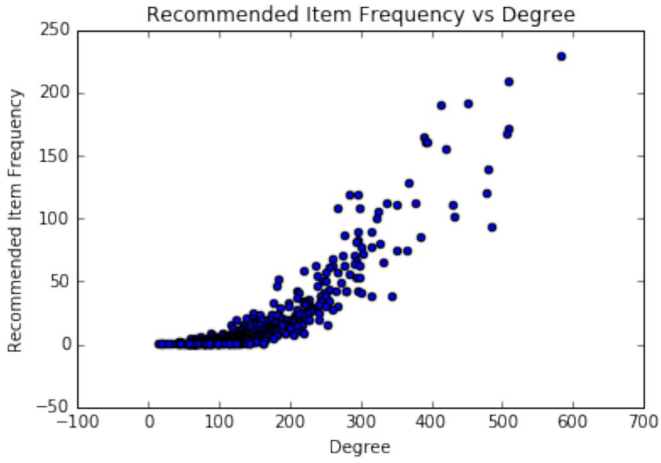

Figure 4: Most recommendations come from low degree nodes in the local subgraph of the user as indicated by the dense cluster in the lower left. The highest-degree nodes have relatively low influence on recommendations in aggregate.

$$w_P = \sum_{(i,j)\in P} r_{\max} - \text{weight}(i,j) \tag{7}$$

In these equations, P_{len} is the number of steps taken in the current random walk P, $r_{\max}$ is the highest rating allowed in the recommender (5 in our case), and f is a scaling factor. The random walk is halted when $e(P) < T$, where T is a stopping threshold.

We modified the MiniPixie algorithm in this way to turn it into a measure of *reachability*. Pixie's walk length sampling is appropriate for discovering item similarity. To measure reachability, however, we want to measure to what extent an item's influence flows out to the graph. The exponential decay function $e(P)$ approximates this magnitude, which decays as the path increases in length or encounters weak edges. Thus, the algorithm measures how many users are connected to an item in its local subgraph via high-rating edges. Those users are likely to receive the item in their recommendation list. Figure 3 illustrates the log-linear relationship between our RWR reachability score and the recommendation frequency. The choice of an exponential decay function is further corroborated by Figure 4, which illustrates the fact that recommendations are not a product of global factors like degree, but rather by local structure.

The attacker sorts the items by their reachability score. The top $N(M-1)$ items are given a fake review with the max rating. The reviews are assigned to the N fake users in round-robin fashion. Every fake user also gave the target item a max rating review. This is a white-box attacker.

Degree-Weighted RWR (DRWR) Attacker *(white-box).* In order to suppress the effect that degree has on inflating the RWR score and favor local structure, this attacker normalizes the RWR score by dividing by the item's degree.

$$\text{DRWR Score} = \frac{\text{RWR score}}{\text{degree}}$$

The attacker sorts the items by their DRWR score. The top $N(M-1)$ items are given a fake review with the max rating. The reviews are assigned to the N fake users in round-robin fashion. Every fake user also gave the target item a max rating review. Items with a high DRWR score spread their influence among a concentrated subgraph of high edge weight, indicating that they are more susceptible to poisoning. Items with high degree will dilute influence across many edges during the Pixie random walk. Therefore, poisoning will be less effective. This is a white-box attacker.

RWR Approximator (RWR-A) Attacker *(black-box).* The RWR approximator attacker is our most sophisticated black-box attacker. It leverages M fake user nodes and one *scout* user. For each of the top 100 items by rating-weighted degree, the *scout* 1) adds a single edge to that item, 2) requests 10 recommendations from the recommender, 3) computes the sum of the degree of all the recommendations and uses the sum as a proxy value for RWR. The other M fake user nodes use the ordering generated by the scout to add edges like the High-Degree Attacker.

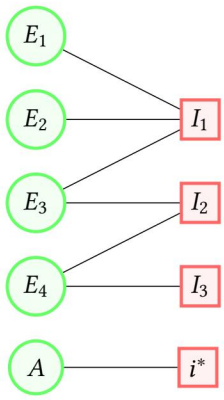

Figure 5: A simple Entity-Item graph, including an attacker and a target item.

Item Set	Hit-ratio
I_1	0.50
I_1, I_2	0.49
I_1, I_3	0.45
I_1, I_2, I_3	0.41
I_2	0.36
I_2, I_3	0.32
I_3	0.29
$\varnothing$	0

Figure 6: Hit ratios for illustrative example.

5 ILLUSTRATIVE EXAMPLE

We provide a small example of an attacker in a hand-crafted network in order to illustrate the key concepts introduced so far in this paper and to highlight some of the interesting key phenomena that we later describe in more detail.

Consider the entity-item graph displayed in Figure 5. There are 4 normal users (E_{1-4}), one Attacker (A), 3 normal items (I_{1-3}), and one target item (i^*). Recall, the goal of A is to add one or more edges to other items such that i^* is recommended to the largest fraction of entities – that is, to maximize the *hit-ratio*. We run 8 trials, one for each of the 2^3 possible sets of the three items that A can add an edge to. In order to account for randomness and for variability in the effect of walk length, we run these trials across 33 different walk lengths averaging at length 20. The table below illustrates the hit rate for each set of items that A adds an edge to:

As expected the hit-ratio for adding no edges is 0 – this is because the random walks in MiniPixie will only ever reach nodes connected to the a given user, and every E is disconnected from i^*. Next, observe that it is better to add an edge to I_1 that not – this makes sense given I_1's high connectivity to E_{1-4}. However, what is quite surprising is that once

Attacker / M	1	2	3	5	10
Random	0.0494	**0.0668**	0.0537	0.0373	0.0280
Average	0.0539	**0.0734**	0.0617	0.0367	0.0322
Neighbor	0.0503	**0.0655**	0.0496	0.0356	0.0248
High Degree	**0.0543**	0.0382	0.0348	0.0297	0.0242
Hill Climbing	**0.0564**	0.0430	0.0397	0.0278	0.0257
RWR	**0.0539**	0.0478	0.0356	0.0314	0.0248
DRWR	**0.0594**	0.0433	0.0356	0.0337	0.0240

Figure 7: Average hit-ratio from 5 attacks with M fake reviews per user and 10% of the users being fake on the MovieLens dataset

you have a good edge, adding edges to less connected items *decreases* the effectiveness of the attack! This is one of the essential properties of attacks on graph-based recommenders, and one that we observe consistently in our experiments.

6 EXPERIMENTS

6.1 The Effect of the Number of Fake Reviews

Our empirical testing confirmed a strong positive correlation between the number of fake users N and hit-ratio. But the more crucial parameter in each attacker is M, the number of fake reviews created per fake user. To investigate the effect that this parameter has on the efficacy of the attackers, we calculated the post-attack hit-ratio for each attacker under different values of M.

Parameter Settings. We let $M = \{1, 2, 3, 5, 10\}$. In each trial, we fixed the number of fake users N to be 10% of all users in the MovieLens100k dataset. The MiniPixie recommender was configured to return 10 items in its recommendation list using the parameters from the sections above.

Evaluation Metric. The aggregate hit-ratio of a trial, uniquely identified by the tuple (*Attacker*, M), was an average of 5 hit-ratios from attacks on 5 different target items which were the same across all the trials. We calculated the hit-ratio for 5 different items to minimize the variability of the results due to the choice of item. We do not report the hit-ratio before the attack because it was always zero or negligible.

Results. The results of the experiment are in Table 1. We see that all of the attackers performed roughly the same and there were no extreme outliers. All of the attackers performed optimally with either $M = 1$ or $M = 2$ fake reviews per fake user and the data suggest that adding more reviews beyond that degrades performance. The Average Attacker achieved the highest hit-ratio among all the attackers. None of our novel attackers (Hill Climbing, RWR, DRWR) outperformed any our baseline attackers (Random, Average, Neighbor, High Degree).

Attacker / N%	1%	5%	10%	20%
Random	0.0083	**0.0350**	0.0668	0.1474
Average	0.0079	0.0322	**0.0734**	**0.1548**
Neighbor	**0.0108**	0.0305	0.0655	0.1410
High Degree	0.0044	0.0220	0.0543	0.1457
Hill Climbing	0.0044	0.0299	0.0564	0.1412
RWR	0.0053	0.0229	0.0539	0.1474
DRWR	0.0047	0.0346	0.0594	0.1438
RWR-A	0.0036	0.0250	0.0564	0.1459

Figure 8: The average hit-ratio from 5 attacks with with N as a percent of the total users on the MovieLens dataset (maximum across M)

6.2 Evaluating Attackers

Having found the optimal range of M, we evaluated our white-box and black-box attackers against one another on the MovieLens100k dataset by measuring the hit-ratio after the attacks as we vary N. The results are in Table 2.

Parameter Settings. Each attacker adds N% fake users to the MovieLens100k dataset, where N is a percent of the total number of users in the graph. We configure the recommender to return 10 recommended items to each user. The attacks were conducted on the MiniPixie recommender, configured using the parameters described in the sections above.

Evaluation Metric. As mentioned earlier, the sole metric we use to measure attackers is the hit-ratio after the attack is conducted. Our independent variable is N, the number of fake users added (as a percent). We are interested in the maximum possible hit-ratio given N across all M. Our prior experiment revealed that the optimal M is usually very low.

Results. The results in Table 2 mirror those in Table 1. Namely, our novel attackers perform worse than the baseline attackers for every N. That isn't to say, however, that our novel attackers perform poorly; they are well within range of the maximum value. There does not appear to be a universally best attacker, though the Average Attacker is the closest to being it.

6.3 Discussion

How many fake reviews should I add? From our experiment in Section 6.1, it is clear that there is a globally optimum number of fake reviews to create per fake user. Every attacker performed optimally when it created 1 or 2 fake reviews per fake user. An intuitive informal explanation for this phenomenon is that the optimal role for a fake user is to serve as a *sink* that directs random walks that reach it toward the target item. If the fake user reviews many other items, then random walks that reach the fake user will walk toward the target item with a lower probability than before, effectively

diluting the target item's influence. This phenomenon has been mentioned in prior literature, as well [7].

How well do the attackers do, and how does the black-box attacker compare to white-box approaches? It appears that the graph structure may be too complex for the assumptions in our novel attackers to be true. The Random Attacker and Average Attacker are black-box attackers yet perform better than our principled white-box attackers. This fact suggests that the graph is home to a variety of different local structures, which could be explored further via motif-matching.

Because our empirical results suggested that a high M degrades the hit-ratio, we were surprised to see the High Degree Attacker perform so well. We expected that the high degree would cause random walks that reached the target item to occur with very low probability as there are many other edges connected to each poisoned item. Its performance suggests, however, that decreasing the shortest path length to the target item for a large number of users (by poisoning the highest degree items) is a great enough boon for the attacker. It is also difficult to say that our DRWR attacker out-performed our RWR attacker. The adequate performance of the High Degree Attacker suggests that high degree is not as significant of a downside as we imagined, so it follows that DRWR does not perform much better than RWR. It remains to be seen why exactly the Average Attacker out-performs the other attackers as we increase N.

One very compelling result to note is that our black-box approximation of the RWR attacker (RWR-A) *matches* the performance of its white-box counterpart (RWR). We were quite surprised to see that a simple approach to approximating RWR could produce a similarly performant attacker — again suggesting that full knowledge of the recommendation system is not necessary to devise a powerful attacker.

7 LIMITATIONS AND FUTURE WORK

Limitations. There are a handful of limitations with the presented work. First, we only evaluate our attackers on the MovieLens 100k dataset, which is both small and may not capture the structure of a wider variety of deployed graphs (e.g. Amazon or Pinterest recommendation graphs). Second, we don't have access to realistic numbers for attackers and for recommenders. How many fake profiles can a shilling attacker create on a site like Yelp, Amazon, or TripAdvisor without being blocked? How many fake reviews can an attacker add without raising suspicion? Insight into these questions can help researchers construct realistic attackers in their studies. Thirdly, we don't have access to massive computation power.

Future Work. Future work could also benefit from understanding more deeply how the particular characteristics and motif profile of a network affect both recommender and attacker performance. Is there anything specific to a recommendation graph that may let us claim *a priori* how successful a black-box attacker may be? Such a study could shed light into why the Average Attacker out-performs our other attackers. We also note that we don't claim to have produced an optimal white-box attacker. Future work could adopt the same approach of approximating a white-box approach with a black-box attacker but with a provably optimal white-box attacker. Similarly, future work could consider many more classes of black-box attackers as have been presented in extensive surveys [9, 20].

8 CONCLUSION

In this report we have explored how black-box shilling attackers can *push* a target item in a graph-based recommender. We implemented MiniPixie, a slimmed-down version of the Pixie recommender system deployed at Pinterest, and we compared its performance against two baseline recommenders on four datasets to corroborate its fitness to study attackers. We then subjected the MiniPixie recommender to abuse by different classes of attackers: several baseline white-box and black-box attackers, a more principled *degree-weighted RWR* attacker, and a black-box approximation of it.

We find that the effectiveness of shilling attackers is determined more by the expressiveness of the edges that are added rather than their volume. We present experiments that corroborate this argument – it seems a shilling attacker that adds to many fake reviews acts as a *bridge* for random walks that pass through it, instead of as a *sink* that directs random walks to the target item. Additionally, we find that our approximate black-box attacker does about as well as principled white-box attackers, and that even principled white-box attackers fare no better than baseline white-box approaches, suggesting a characteristic *robustness* in the recommendation system we utilize.

REFERENCES

[1] Charu C Aggarwal et al. 2016. *Recommender systems.* Springer.

[2] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 635–644.

[3] Shumeet Baluja, Rohan Seth, D Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. 2008. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the 17th international conference on World Wide Web*. ACM, 895–904.

[4] Robin Burke, Bamshad Mobasher, Runa Bhaumik, and Chad Williams. 2005. Segment-based injection attacks against collaborative filtering recommender systems. In *Data Mining, Fifth IEEE International Conference on*. IEEE, 4–pp.

[5] Paul-Alexandru Chirita, Wolfgang Nejdl, and Cristian Zamfir. 2005. Preventing shilling attacks in online recommender systems. In *Proceedings of the 7th annual ACM international workshop on Web information and data management*. ACM, 67–74.

[6] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1775–1784.

[7] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. 2018. Poisoning Attacks to Graph-Based Recommender Systems. *arXiv preprint arXiv:1809.04127* (2018).

[8] Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on knowledge and data engineering* 19, 3 (2007), 355–369.

[9] Ihsan Gunes, Cihan Kaleli, Alper Bilge, and Huseyin Polat. 2014. Shilling attacks against recommender systems: a comprehensive survey. *Artificial Intelligence Review* 42, 4 (2014), 767–799.

[10] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 505–514.

[11] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2016), 19.

[12] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. 2010. *Recommender systems: an introduction*. Cambridge University Press.

[13] Gérald Kembellec, Ghislaine Chartron, and Imad Saleh. 2014. *Recommender systems*. Wiley-IEEE Press.

[14] Shyong K Lam and John Riedl. 2004. Shilling recommender systems for fun and profit. In *Proceedings of the 13th international conference on World Wide Web*. ACM, 393–402.

[15] Shyong K. Lam and John Riedl. 2004. Shilling Recommender Systems for Fun and Profit. In *Proceedings of the 13th International Conference on World Wide Web (WWW '04)*. ACM, New York, NY, USA, 393–402. https://doi.org/10.1145/988672.988726

[16] Michael Luca and Georgios Zervas. 2016. Fake it till you make it: Reputation, competition, and Yelp review fraud. *Management Science* 62, 12 (2016), 3412–3427.

[17] Paolo Massa and Paolo Avesani. 2007. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 17–24.

[18] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. 2007. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology (TOIT)* 7, 4 (2007), 23.

[19] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. Recommender systems: introduction and challenges. In *Recommender systems handbook*. Springer, 1–34.

[20] Mingdan Si and Qingshan Li. 2018. Shilling attacks against collaborative recommender systems: a review. *Artificial Intelligence Review* (2018), 1–29.

[21] Ziqi Wang, Yuwei Tan, and Ming Zhang. 2010. Graph-based recommendation on social networks. In *Web Conference (APWEB), 2010 12th International Asia-Pacific*. IEEE, 116–122.

[22] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435* (2017).