Toward Detecting Collusive Ranking Manipulation Attackers in Mobile App Markets

Hao Chen
School of Computer Science and Software Engineering
East China Normal University
spikechenhao@163.com

Daojing He
School of Computer Science and Software Engineering
East China Normal University
djhe@sei.ecnu.edu.cn

Sencun Zhu
The Pennsylvania State University
szhu@cse.psu.edu

Jingshun Yang
School of Computer Science and Software Engineering
East China Normal University
sei_yjs2013@126.com

ABSTRACT

Incentivized by monetary gain, some app developers launch fraudulent campaigns to boost their apps' rankings in the mobile app stores. They pay some service providers for boost services, which then organize large groups of collusive attackers to take fraudulent actions such as posting high app ratings or inflating apps' downloads. If not addressed timely, such attacks will increasingly damage the healthiness of app ecosystems. In this work, we propose a novel approach to identify attackers of collusive promotion groups in an app store. Our approach exploits the unusual ranking change patterns of apps to identify promoted apps, measures their pairwise similarity, forms targeted app clusters (TACs), and finally identifies the collusive group members. Our evaluation based on a dataset of Apple's China App store has demonstrated that our approach is able and scalable to report highly suspicious apps and reviewers. App stores may use our techniques to narrow down the suspicious lists for further investigation.

Keywords

App Stores; Ranking Fraud; Collusion Groups

1. INTRODUCTION

To help users discover popular apps, app stores like Apple iTunes and Google Play provide the ranking charts on their front pages. The higher ranking an app has, the more easily it can be discovered by users, which in turn can trigger a larger number of downloads and hence higher revenues. Hence, app developers often try various strategies (e.g., advertisement, optimization) to boost their apps' chart rankings as much as possible. Some developers, however, are even incentivized to take fraudulent methods to promote their apps rankings. They hire fraudulent ranking boost service providers to perform the promotion. A search on Google with keywords such as "app promotion services" can show that numerous companies are engaged in such a business on the Internet. Ranking manipulation can greatly harm the mobile app ecosystem. Since apps can be promoted to high ranks merely by paying money to ranking boost companies, it will lead to unfair competition among the developers in app stores, especially discouraging those honest developers. What's more, such fraudulent methods also mislead app users to install low-quality apps, which sometimes are spyware or even malware [1]. For these reasons, Apple App Store warns the developers not to cheat the user reviews with fake or paid reviews in Clause 3.10 of "App Store Review Guidelines" [2]; GooglePlay requires the developers not to manipulate product ratings or reviews with unauthorized means like fake or paid reviews in its "Google Play Developer Program Policies" [3]. In recent yeas, Apple iTunes has also cracked down some dishonest app developers by removing their apps from the store.

Previous research [4] [5] [6] [7] [8] [9] [10] [11] or collusive spammers [12] [13] [14] [15] has focused on discovering single spammer and spam on online shopping websites. For example, most of their spammer or spam detection was done on datasets from Amazon, DianPing, Yelp, etc. Spammers in those websites give spam (i.e, fake reviews) on products to mislead customers into buying them. In contrast, much less effort has been focused on fraud detection in mobile app markets. In this work, we aim to discover attackers that form collusion groups to promote apps' rankings. For effectiveness of promotion (e.g., boosting an app into top 20), fraudulent campaigns often involve large groups of collusive users. In this work, we call such groups of collusive attackers *collusive promotion group*, and apps promoted by collusive promotion groups are called *targeted* apps.

To discover collusive promotion groups, the most difficult task is to identify targeted apps and group members behind them. Currently, app stores like Google Play and Apple iTunes both have over 2 million apps [16] and over 800 millions of users [17]. To identify members of collusive promotion groups, directly looking through all users and all apps in the store is inefficient and unrealistic. Fortunately,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASIA CCS '17, April 02-06, 2017, Abu Dhabi, United Arab Emirates
© 2017 ACM. ISBN 978-1-4503-4944-4/17/04…$15.00
DOI: http://dx.doi.org/10.1145/3052973.3053022

during promotion periods, targeted apps have different ranking, rating, and review number change patterns, compared to normal apps. For example, promoted apps are likely to have drastic ranking increase or decrease in a short time, or show strong rating deviation. In addition, we find that, for a group of collusive attackers, they often promote multiple targeted apps in a batch. As a result, such targeted apps are likely to demonstrate similar unusual change patterns. Specifically, we call apps that are co-promoted by collusive attackers *targeted app cluster(TAC)*.

Based on the above observation, we propose a novel approach to identifying attackers of collusive promotion groups. We first identify targeted apps from the massive number of apps in an app store by mining apps' unusual ranking change patterns. Then, we extract indicative characteristics of TAC, and convert them into pairwise features to signify the relations among cluster members. By measuring feature similarity between apps, we identify app pairs from the same TAC. We further design an algorithm to generate app clusters containing TACs. Finally, applying a data mining technique called frequent itemset mining on such app clusters, we report members of collusive promotion groups.

Our approach has been evaluated with a dataset of Apple's China App store collected from March 13, 2016 to April 13, 2016. It first reported 461 suspicious apps with strong evidences of being promoted. Based on these apps, our approach is able to identify 7 app clusters with size over 20 (apps). Finally, it exposes many collusive promotion groups, some of which having hundreds of members. Our further investigation supports that these users are collusive reviewers because they exhibit other unusual behavior such as reviewing over 500 apps in total, reviewing over 10 apps in one day, or only giving 5 stars.

Our approach can detect highly likely collusive promotion groups. App stores may use additional evidences, which we do not have (e.g., information about reviewer accounts), to further pinpoint the promoted apps and the collusion groups. Although tested against a dataset of Apple's China App store, our proposed approach is based on very general principles, and hence it can also be applied to find collusive attackers in any other mobile app markets (such as Google Play) where ranking fraud exists.

2. PRELIMINARIES

2.1 Attack Model

The ranking charts in an app store provide the ordered lists of the most popular apps and are updated periodically (e.g., daily). No matter what ranking algorithm the app store uses, in general an app's real-time ranking is greatly affected by its number of downloads, number of reviews, and ratings in the past. Since users in the store can review and rate apps conveniently after installing them, to promote an app's ranking effectively, a common way is to launch a *fraudulent burst campaign*, where a promotion service provider organizes a large group of users to download the app, give high ratings and write good reviews in a short time period. Consequently, the chart ranking of the app may be drastically boosted. The app may also be placed very top in the "What's Hot" section in the "Featured" subcategories, as well as highly ranked in search results. Such ranking changes will increase the app's exposure in the app store and hence attract more users to download it.

As app stores like Apple iTunes have tightened security check on its account system, it is hard for fraudulent burst campaigns to register and use bot accounts to promote apps automatically. Currently, burst campaigns tend to hire and organize several groups of real users through social networks such as QQ group and WeChat group. During each burst campaign, a busy promotion service provider may target at multiple apps at the same time by using collusion groups. In return, users in collusion groups will get paid for their services.

Specifically, an app under ranking manipulation always targets at an expected rank, such as top 50 in the leaderboard for some specified periods. The campaign service provider may offer several kinds of services [18], such as top 50 for one week, top 100 for two weeks, and charge money based on chosen services. To achieve apps' ranking goals, a service provider often organizes different user groups to separately promote target apps within different periods. Moreover, multiple apps requesting similar promotion services (e.g., top 50 for one week during a similar time period) are often combined and assigned to different collusive promotion groups during each promotion campaign. We call such a combined app group as a target app cluster (TAC). This attack model is further illustrated in figure1.

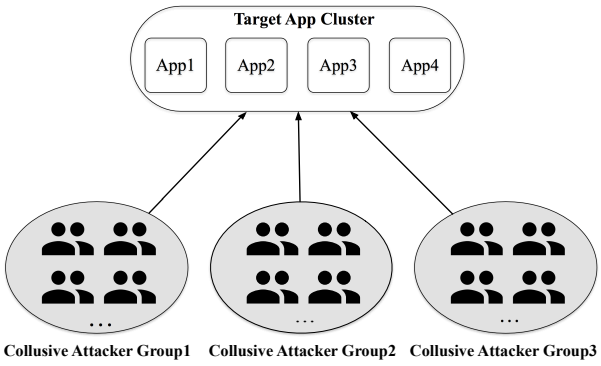


Figure 1: TAC promoted by collusive attacker groups

Note that in this work, we aim to discover groups of attackers who collectively manipulate rankings of *multiple* apps (i.e., TAC). We do not focus on detecting attackers who only promote a single app. Drastic change of ranking is not rare even for some normal apps (e.g, Flappy bird), so detecting attackers based on a single app is likely to cause high false positives. Our work is different from previous work such as [19] and [20], which focus on identifying collusion groups that provide fake app ratings. However, app rankings are also greatly affected by factors like daily number of downloads, not just ratings. Our work complements the previous work by leveraging app ranking change information to identify suspicious apps and attackers.

2.2 Ranking Float Types

Because the app ranking chart is updated periodically (e.g., daily), each app has its past ranking records in the store. Specifically, the ranking records for app a can be represented as a time series, $\psi_a = \{\psi_a^1, ..., \psi_a^i, ..., \psi_a^n\}$, where $\psi_a^i \in \{1, ..., K, +\infty\}$ is the ranking of app a at the time

stamp i; n denotes the total number of ranking records. The smaller value ψ_a^i is, the higher ranking app a has. Let $\Delta\psi_a^i = \psi_a^{i+1} - \psi_a^i$ denote the ranking change of app a during a period $[i, i+1]$ (e.g., 24 hours in our experiment). Negative $\Delta\psi_a^i$ indicates the rise of app a's ranking, and vice versa. Based on apps' ranking change, we propose the ranking change type t_i^a for app a during the period $[i, i+1]$ and categorize it into three types:

Ranking change type	Condition
drastic float up	$\Delta\psi_a^i < -T_\psi$
drastic float down	$\Delta\psi_a^i > T_\psi$
slight float	$-T_\psi \leq \Delta\psi_a^i \leq T_\psi$

where T_ψ is the threshold to define unusual ranking change (e.g., 150 in our experiment). For any app a, if $\mid \Delta\psi_a^i \mid > T_\psi$, it will be considered as having unusual ranking change (i.e., *drastic float up* or *drastic float down*) during the period $[i, i+1]$. Hence, historical records of ranking change type for app a can be denoted as a time series, $T_a = \{t_1^a, t_2^a, ..., t_i^a, ..., t_{n-1}^a\}$.

Previous work [21] has revealed that, compared with normal apps, ranking fraudulent apps often have different ranking patterns during promotion periods. Specifically, through our observation, it has the following characteristics.

- First, when an app is promoted by collusive attackers, its ranking is likely to rise drastically in a short period.
- Second, such app's ranking could also drop rapidly. Ranking manipulation for a top position in the chart is quite expensive. According to the online report [18], to get into the top-10 free apps chart costs 11,200 USD, and keeping it there will cost developers another 65,000 USD per week. Clearly, developers cannot afford to promote their apps all the time. Therefore, unlike normal apps, after reaching the expected ranking goal, a promoted app often cannot keep the same position successfully and would be crowded out quickly once the fraudulent burst campaign is finished.
- Third, after seeing the ranking of his app drops quickly, some developers may decide to launch a second campaign (or more) through the same service provider, which will organize different groups of attackers to perform the manipulation task (a user cannot repetitively rate one app in the store).

Based on the above three characteristics, compared with normal apps, promoted apps in TACs may exhibit drastic increase or decrease in chart rankings when they are promoted multiple times. More specifically, a promoted app often has volatile ranking change, that is, *drastic float up* and *drastic float down* during promotion periods. In contrast, for normal apps, their ranking change type records are *slight float* at most of the time. Figure 2 shows the comparison of ranking change pattern between a normal app and a highly suspicious promoted app during its promotion periods from Apr 11 to Apr 25 in 2016.

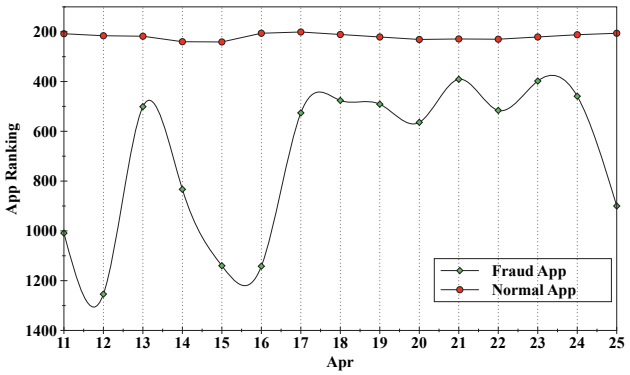


Figure 2: Ranking change patterns of a normal app and a highly suspicious promoted app

3. COLLUSIVE ATTACKERS DISCOVERY

3.1 Overview

As mentioned, in order to impact the ranking of an app significantly, fraudulent burst campaigns organize groups of users to commit ranking frauds collectively (i.e., collusive promotion group). Reviewers in these promotion groups take very similar actions (e.g., downloading, rating). Compared with normal reviewers who rate apps independently, members of collusive promotion groups have a much higher probability to rate the same set of apps. However, in general, co-rating and co-review (i.e., rating and writing reviews to the same set of apps) is not common. According to [19], with the increase of the number of commonly rated apps, co-rating probabilities decrease sharply. The authors collected 553,005 reviewers in the Apple App store and found only 0.163% pairs of the reviewers had two commonly rated apps, and this percentage decreases to 0.012% for 3 commonly rated apps. Therefore, when several users co-rate more and more apps, the relations of group members are increasingly strengthened, and eventually their relations become very close. Hence, to discover group members is to detect such close relations.

User accounts in app stores like iTunes are bound with hardware devices (e.g., smartphones or tablets), so collusive attackers are only likely to account for a small portion among all user accounts. Thus, directly detecting relation among all reviewers in an app store is inefficient. On the other hand, if we can first capture TACs in the store, members of collusion groups can be found much more efficiently. This is because the collusive attackers leave ratings and reviews on apps in TACs. By checking review information of apps in one TAC directly, collusion behaviors (i.e., co-review and co-rate) can be easily detected and exposed.

Specifically, by filtering out a large amount of normal apps and mining targeted app clusters (TACs) in an app store, we can obtain a much-narrowed-down suspect list for collusive promotion groups discovery. Here we outline three challenges to be solved.

- First, how to distinguish between normal apps and targeted apps promoted by colluders?
- Second, how to identify cluster members (i.e., apps from the same TAC) and capture TACs in the wild?
- Third, how to capture cluster member apps in one TAC as complete as possible?

To deal with the above three challenges, we propose the following framework. We first differentiate promoted apps

from massive normal apps in the store. Then, we utilize three pairwise features to find collusive pairs (app pairs from the same TAC). An algorithm will be proposed to build app clusters that cover TACs based on these pairwise features. Finally, frequent itemset mining will be applied on each app cluster to discover members of promotion groups .

3.2 Capture Suspicious Promoted Apps

As mentioned earlier, promoted apps in TACs exhibit volatile ranking change during promotion periods, whereas for normal apps, their ranking changes are much slower and smoother at most of the time. Hence, to find promoted apps in an app store, we can focus on those apps with *drastic float up* or *drastic float down* ranking change patterns.

Not surprisingly, some normal apps may too have drastic ranking changes in certain cases. For example, store vendors sometimes recommend some high quality apps and display them in the first page of the store, which may trigger a huge number of downloads of those apps and cause their rankings to rise rapidly. In another example, apps providing limited-time discount may also experience drastic ranking increases.

However, for normal apps, such drastic ranking changes rarely happen and for each app, it rarely happens multiple times. Hence, we can identify promoted apps by measuring their drastic ranking change frequency (which we call *DRCF*) within certain periods of time. Specifically, we first quantify the ranking change types for any app a as follows:

$$V_{rank}(t_i^a) = \begin{cases} 0 & \text{t}_i^a \text{ is slight rank float} \\ 1 & \text{t}_i^a \text{ is drastic float up} \\ -1 & \text{t}_i^a \text{ is drastic float down} \end{cases} \quad (1)$$

,where t_i^a is the ranking change type during the period $[i, i+1]$. Then, based on quantified ranking type records, the app's drastic ranking change frequency can be calculated as follows:

$$f_a = \frac{\sum_{i=1}^{n} \mid V_{rank}^a(t_i^a) \mid}{M} \quad (2)$$

, where M is the detection period(e.g., 30 days in our experiment). If the fraudulent burst campaign organizes collusive groups to promote apps in a TAC within the detection period, these apps are likely to have much higher DRCF values than normal apps. Specifically, we propose a DRCF threshold T_f to tell ranking fraud apps from normal apps. For any app a , if $f_a > T_f$, it will be regarded as a suspicious promoted app (SPA), which means it has a high possibility of ranking fraud. By measuring apps' DRCFs, we can obtain the set of suspicious promoted apps (SPAs) $\{\alpha_1, \alpha_2, \alpha_3, ...\}$.

3.3 Collusive Ranking Fraud Measurement: Pairwise Features

To maintain targeted apps' high rankings, during promotion periods, the promotion groups take actions on the TAC several times. Each time, members of a promotion group co-review and co-rate apps in one TAC.

In this section, we present several pairwise features to find collusive pairs (app pairs in one TAC). By comparing data change between app pairs, highly similar data change pattern will reveal their close relation. Obtainable application data such as each app's ranking, rating, and review amount are taken into account. Note that only SPA (i.e., $\{\alpha_1, \alpha_2, \alpha_3, ...\}$) will be considered.

3.3.1 Review Volume Explosion Similarity (RVES)

Following arrangements, a large size of group members collaborate to post reviews for targeted apps in one TAC in a short time interval. This causes apps in a TAC to experience explosion of review numbers in a highly consistent way. What's more, since a TAC might be promoted several times, review explosion could occur repeatedly. The relation of any two apps is strengthened when they show more similar review explosion patterns.

Let's $Q_a = \{q_1^a, q_2^a, ..., q_i^a, ..., q_n^a\}$ denote the records of review number changes for app a in a time series, where $q_i^a \in [0, +\infty]$ is the review number change during the time period $[i, i+1]$ (24 hours in our experiment); n denotes the total number of records. The average review number change of app a can be calculated as:

$$\overline{q_a} = \frac{\sum_{i=1}^{n} q_i^a}{n} \quad (3)$$

To detect review explosion for an app a, we define the review volume explosion signature:

$$V_{volume}(q_i^a) = \begin{cases} 1 & \frac{q_i^a}{\overline{q_a}} > T_{surge} \\ 0 & otherwise \end{cases} \quad (4)$$

where T_{surge} is a threshold (in our experiment, we set $T_{surge} = 1.3$). Then, given an app pair (a, b), $\lambda_{RVES}(a, b)$ defines their review volume explosion similarity (RVES):

$$\lambda_{RVES}(a, b) = \sum_{i=1} V_{volume}(q_i^a) \cdot V_{volume}(q_i^b) \quad (5)$$

The larger the RVES value is, the higher pairwise similarity exist between two apps in terms of review explosion patterns. Specifically, we propose a similarity threshold T_{RVES} (e.g., 5 in our experiment). If $\lambda_{RVES}(a, b) > T_{RVES}$, the pair (a, b) is regarded as having *strong* pairwise similarity in review volume explosion pattern. Otherwise, they will be judged as having *weak* similarity in this feature.

3.3.2 Rating Deviation Similarity (RDS)

An app store also displays an average rating for every app, which ranges from 1 to 5. Normal users rate apps independently based on their own user experience. Hence, statistically speaking, the average rating for a normal app will not change with the number of reviewers. However, when collusive attackers co-rate multiple apps with high ratings (e.g., 5 stars) during the promotion periods, the apps' ratings will be deviated from the ratings received from normal customers. That is, apps in one TAC may show similar rating deviation patterns.

Let $R_a = \{r_1^a, r_2^a, ..., r_i^a, ..., r_n^a\}$ denote the rating records for app a in a time series, where $r_i^a \in [1, 5]$ is the average rating at the time stamp i; $\Delta r_i^a = r_{i+1}^a - r_i^a$ calculates the rating deviation of app a at the time stamp i. Thus, the app's rating deviation records can also be listed in a time series $\Delta R_a = \{\Delta r_1^a, \Delta r_2^a, ..., \Delta r_i^a, ..., \Delta r_n^a\}$; n is the total number of records.

We note that developers often update their apps and provide new versions for downloading after a period of time. Every time when an app's version is changed, its average rating will be recalculated in the store. In other words, the rating score displayed in the store represents the average rating for the current version. The quality of an app

might change due to version updates, so an app's average rating for different version might vary. For example, if an app adds some attractive features in its new version, its average rating, even without promotion attacks, could be improved. In contrast, if the app's new version comes with a lot of bugs, normal users will give low ratings to this version. Hence, when measuring rating deviation for an app, the influence of its version change should be taken into account. Specifically, the corresponding version of each rating $Version_a = \{v_1^a, v_2^a, \ldots, v_i^a, \ldots, v_n^a\}$ should be collected, where v_i^a represents the version for r_i^a. When measuring rating deviation for an app in each period, the version of each rating record should be compared. If $v_i^a \neq v_{i+1}^a$, the value of Δr_i^a will be set to ∞, which means incomparable. In other words, any rating deviation caused by version changes will not be taken into account. Finally, the rating deviation signature [1] for app a during the period $[i, i+1]$ can be quantified by:

$$V_{rating}(\Delta r_i^a) = \begin{cases} 0 & \Delta r_i^a \leq 0 \text{ or } \infty \\ 1 & \Delta r_i^a > 0 \end{cases} \tag{6}$$

The number of ratings differs from app to app in the store. A popular app may have hundreds of raters, while a non-popular app may merely have a few raters. For any app, the more reviewers rate it, the more difficult its average rating can be deviated by the same number of promoters. For example, an app with the average rating 4 stars may include total 180 user ratings. To make its average rating increase to 4.5, it needs at least 180 more users to give 5-star rating. However, to another app which has the same average rating (i.e., 4 stars) with only 18 user ratings, it only takes 18 more users to give 5 stars to boost its average rating to 4.5. As the result, when a group of collusive attackers promote multiple apps in one TAC, the ratings of these apps might not increase in the same manner, even though they receive the equal number of high ratings. To capture the pairwise similarity of two apps in terms of rating deviation, we consider the rating changes in multiple promotion periods. Specifically, given an app pair (a, b), their rating deviation similarity (RDS) can be measured in the following way:

$$\begin{aligned} V_{rating}^b(i, w) = & V_{rating}(\Delta r_{i-w}^b) \cup \ldots \cup V_{rating}(\Delta r_i^b) \cup \ldots \\ & \cup V_{rating}(\Delta r_{i+w}^b) \end{aligned} \tag{7}$$

$$\lambda_{RDS}(a, b) = \sum_{i=1} V_{rating}(\Delta r_i^a) \cdot V_{rating}^b(i, w) \tag{8}$$

where w is the extended time period (e.g., 3 days in our experiment); $V_{rating}^b(i, w)$ is the relative rating deviation signature for app b. As long as app b has rating increase during any of the period in $\{i-w, i-w+1, ...i, .., i+w-1, i+w\}$, it will be regarded as having similar rating deviation as app a.

The more similar pattern two apps exhibit in rating deviation, the larger similarity score they will receive. Specifically, we set the similarity threshold $T_{RDS} = 4$. For any app pair (a, b), if $\lambda_{RDS}(a, b) > T_{RDS}$, app a and app b will be regarded as having strong similarity in rating deviation. Otherwise, they are judged as having weak similarity.

[1] we only focus on rating increase here, since collusive attackers always give high rating in order to boost an app's ranking

3.3.3 Ranking Float Similarity (RFS)

Besides ratings and reviews, apps' rankings are also greatly influenced by some other factors, such as number of download and number of daily active users. Therefore, to promote apps' rankings, group members may also adopt other methods, such as boosting app's download number or increasing number of daily active users. Such statistical information, however, is unknown to a third party – only the app store and the developer can see the app's usage information.

However, as we mentioned, promoted apps always have expected ranking targets and apps in one TAC tend to follow the same ranking promotion schedule by the service provider. Moreover, during the ranking promotion periods, apps promoted by attackers are likely to simultaneously experience drastic increase (i.e., drastic float up) or rapid decrease (i.e.,drastic float down) in rankings. That is, the drastic ranking change pattern of apps in a TAC appears similar. Given the app pair (a, b), we define $\lambda_{RFS}(a, b)$ to evaluate their similarity on drastic ranking change pattern:

$$\lambda_{RFS}(a, b) = \sum_{i=1} V_{rank}(t_i^a) \cdot V_{rank}(t_i^b) \tag{9}$$

For app pairs in one TAC, their RFS value could be large due to common promotion actions. We further set a RFS threshold T_{RFS} (e.g., 8 in our experiment). If $\lambda_{RFS}(a, b) > T_{RFS}$, the app pair (a, b) will be regarded as having highly similar pattern of drastic ranking change.

So far, we have proposed three pairwise features for apps involving fraudulent burst campaigns. Specifically, for any app pair (a, b), they will be judged as a suspicious co-promoted pair if they have strong similarity in at least one pairwise features. In practice, some normal apps in the app store or apps from different TACs might exhibit similar change patterns in one or more features by chance. However, unlike apps in one TAC, which are subject to unified ranking promotion, it is unlikely for normal app pairs to generate high similarity values exceeding set thresholds. Note that here we do not require two apps to have strong similarity in two or even three features. Because our ultimate goal is to discover members of promotion groups behind TACs, we like to include more app pairs as candidates for further investigation and not mistakenly eliminate real suspicious pairs. Even though some app pairs may be misidentified as having strong similarity, such mistake will not prevent our ultimate goal of discovering promotion group members, due to the adoption of our method proposed in Section 3.5.

3.4 Capture Target App Clusters (TACs)

In this section, we will capture TACs for group members discovery. Because of consistent promotion actions taken by collusive attackers, apps from the same TAC will finally form the suspicious co-promoted pairs. To capture TACs in the wild, we will focus on app pairs that are measured as suspicious co-promoted pairs. Specifically, we give the following definitions.

Definition(Co-promoted Pair Cluster): Given any app a, the co-promoted pair cluster(CPC) $c(a)$ contains a and all other apps $\{b\}$ if the pair (a, b) is judged as a suspicious co-promoted pair. Here app a is called the *seed* app, and $c(a)$ is the CPC constructed based on app a.

Definition(Co-promoted Pair Graph): Co-promoted Pair Graph (CPG) is an undirected graph, in which every vertex is an app, and an undirected edge indicates that the two connected apps form a suspicious co-promoted pair. To construct a CPC based on a seed app is to find the seed app and all other apps that are linked to it in the CPG.

Intuitively, in the ideal situation, any two apps from the same TAC are linked together due to collective promotions conducted by group members. For apps that do not belong to the same TAC, their behavior is independent and thus connections are hard to form. Therefore, in an ideal scenario, a TAC will become a complete graph as shown in Figure 3. In this situation, an app's CPC is identical to the TAC it belongs to. For example, in Figure 3, the CPC $c(3)$ includes app3 and all apps linked to it, i.e., app1, app2, app4 and app5. The resulting CPC $c(3) = 1, 2, 3, 4, 5$ is identical to the TAC A.

However, in practice, the situation can be somewhat different. First, apps in one TAC is not always linked to any other cluster members in the CPG for some reasons. For example, apps in one TAC might not follow exactly the same promotion schedule. Although apps in one cluster may request similar services (e.g., top 50 for one week), their beginning days and ending days of promotion might not be exactly the same. Therefore, some app pairs in one TAC might not have strong similarity in feature change patterns to form suspicious co-promoted pairs. As shown in Figure 4, app1 does not link to app3 although they belong to the same TAC A, because they follow the similar but not the same promotion schedule.

Second, different TACs may also be connected for some reasons. In Figure 4, app5 and app6 are connected to each other although they belong to groups A and B, respectively. It is possible that these two apps are promoted by two independent fraudulent campaigns but follow the similar promotion schedule accidentally. As a result, they are judged as suspicious co-promoted pairs through our feature similarity evaluation.

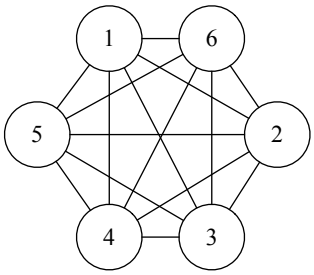


Figure 3: The app graph of TAC $A = \{1, 2, 3, 4, 5, 6\}$ in the ideal situation.

Because of the above reasons, a typical target app in one TAC may be linked to many other member apps but not all of them, and certain members from different TACs may be linked together. Therefore, oftentimes an app's CPC is not totally identical to the TAC it belongs to. However, it is obvious that CPCs based on seed apps from the same TAC have higher similarity among cluster members compared with CPCs constructed by apps from different TACs. For example, in Figure 4, CPC $c(1)$ and CPC $c(2)$ have 5 common apps since app1 and app2 are members in one TAC,

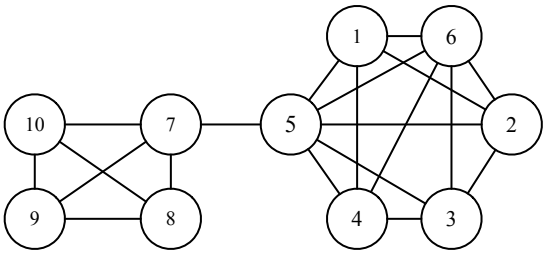


Figure 4: The app graph of two TAC $A = \{1, 2, 3, 4, 5, 6\}$ and $B = \{7, 8, 9, 10\}$ in the general situation

whereas $c(5)$ and $c(7)$ only have 2 common members since the seeds app5 and app6 do not belong to the same TAC. Moreover, for two CPCs constructed by seed apps from one TAC, their union set might cover more members in that TAC. As shown in Figure 4, when combining $c(1)$ and $c(2)$, we can obtain $\{1, 2, 3, 4, 5, 6\}$. This contains cluster members in TAC A more completely than members covered by $c(1) = \{1, 2, 4, 5, 6\}$ or $c(2) = \{1, 2, 3, 5, 6\}$.

Based on the above discussion, if we first find CPCs constructed by seed apps in one TAC, and then combine members of these CPCs, we will obtain a larger cluster. To achieve this goal, we propose an algorithm, which is called $CandidateClusterCapture$. The algorithm runs recursively. In each recursive call, the algorithm first measures the similarity of every two CPCs based on their cluster members. Then, CPCs with member similarity over the desired threshold will be merged. Specifically, in this work, CPCs that can be merged are called *homogeneous* CPCs. The output of $CandidateClusterCapture$ is the set of app clusters, which we call *candidate clusters*, that cover member apps of TACs. These candidate clusters will be used for final collusive attacker discovery in the next section.

The details of $CandidateClusterCapture$ is given in Algorithm 1, which consists of two main steps: (1) CPC construction (2) Cluster mergence. In Step (1), the nested loop will construct CPCs based on every SPA in our dataset. Step (2) involves the cluster merging operation, which is the recursive function called $ClusterMerge$. For any two homogeneous clusters, their cluster members will be combined. Specifically, before homogeneity identification, the inclusion relation (i,e., one cluster is included in the other cluster) between two clusters needs to be checked. If $c_i \subseteq c_j$ or $c_j \subseteq c_i$, the smaller group will be removed from c, since all members in the smaller cluster are contained in the larger group. Otherwise, homogeneity identification between two clusters will be performed. Here we use Jaccard index to measure the similarity of cluster members, which is defined as follows:

$$Sim(c_i, c_j) = \frac{(c_i \cap c_j)}{(c_i \cup c_j)} \tag{10}$$

Algorithm 1 CandidateClusterCapture

Input: The set of SPAs $\{\alpha_1, \alpha_2, ...\alpha_n\}$
Output: The list of candidate clusters

//Step 1: construct CPCs based on SPAs
for $i = 1$ to n **do**
  $c(\alpha_i) \leftarrow \alpha_i$ // insert the initial app
  for $j = 1$ to n **do**
    if (α_i, α_j) is suspicious collusive pair **then**
      $c(\alpha_i) \leftarrow \alpha_j$ // put α_j into $c(\alpha_i)$
    end if
  end for
end for

//Step 2: Merge candidate clusters
Initial the CPC set: put every CPC constructed in Step 1 into the set. Let $C = \{c_1, c_2, ...\}$ denote the sequence of CPCs in C

function ClusterMerge(c)
  $combinable = true$
  for all $c_i, c_j \in G, c_i \neq c_j$ **do**
    if $c_i \subseteq c_j$ **then**
      remove c_i from c
    else if $c_j \subseteq c_i$ **then**
      remove c_j from c
    else if $Sim(c_i, c_j) > T_{js}$ **then**
      $G \leftarrow c_i \cup c_j$
      remove c_i and c_j from c
    else
      $combinable = false$
    end if
  end for
  if $combinable == true$ **then** ClusterMerge(c)
  end if
end function
return C evolving candidate clusters $c_1, c_2, c_3...$ for further collusive attackers detection

Empirically, here we set the similarity threshold $T_{js} = 0.6$. c_i and c_j are judged as homogeneous and combined if $Sim(c_i, c_j) > T_{js}$. Once two homogeneous clusters are combined ($c_i \cup c_j$) and form C, the original clusters c_i and c_j will be removed from C. Note that the combined clusters in C might not be the final candidate clusters, since the algorithm runs recursively. Here, *combinable* is the flag used to indicate whether $ClusterMerge$ should end or start the next recursion. The algorithm will finish when no clusters in C can be measured as homogeneous and be merged. Note that some apps may be misidentified as co-promoted pairs, some candidate clusters may be formed coincidentally. However, the size of such randomly formed candidate clusters will not be very large. In this work, we set the cluster size threshold T_{size} (e.g., 20 in our experiment), so only candidate clusters with size larger than T_{size} will be used in collusive attackers discovery.

3.5 Collusive Attackers Discovery

Compared with the kinds of users who post reviews on apps individually and independently, members of one collusion group co-review and co-rate several apps (i.e., TAC) together. Since we have obtained candidate clusters covering the member apps of TACs, to discover groups of collusive attackers is equivalent to find groups of users who have co-reviewed and co-rated multiple apps in those candidate clusters. Because co-reviewing and co-rating happen at the same time, here we only need to focus on one.

Here we use the data mining technique called *frequent itemset mining(FIM)* for collusive attackers discovery. This technique has also been applied in [22] [12]. In our context, user ids are regarded as items, and each transaction is the set of users who have reviewed a specific app. By FIM, groups of users who have co-reviewed multiple common apps can be found. Specifically, FIM will be applied in each candidate cluster separately. Because we focus on the worst collective ranking promotion activities, here we use maximal frequent itemset mining (MFIM) to discover collusive attacker groups with maximal size. By applying MFIM on one candidate cluster, we can get a set of itemsets. As shown in Figure 5, each itemset is a mapping from a group of reviewers to a set of co-reviewed apps. Specifically, we only generate itemsets that consist of at least 20 reviewers who have co-reviewed at least 3 apps. Any smaller groups will not be taken into account.

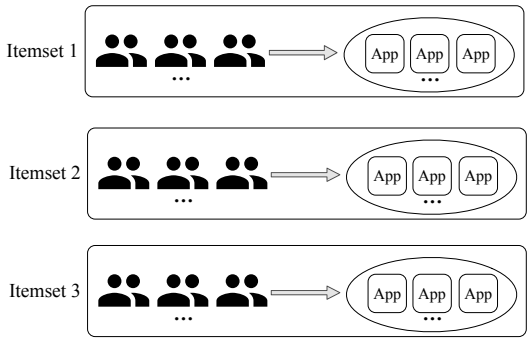


Figure 5: Itemsets obtained by applying MFIM on one candidate cluster

It must be noted that, in our work we do not claim the detection results are 100% accurate, due to the difficulty of getting the ground truth. However, our algorithm output can provide a much-narrowed-down suspect list for further investigation by app store. All apps in each candidate cluster have been previously labeled as suspicious promoted apps and apps in one candidate cluster have highly similar unusual feature change patterns. This indicates that user groups which take co-review and co-rate action behind these candidate clusters are highly likely collusive promotion groups. App stores may use additional evidences which we do not have (e.g., information about reviewer accounts) to further pinpoint the fraud apps and the collusive attacker groups.

4. EVALUATION

4.1 Building The Reference Dataset

Compared with normal apps, fraudulent apps tend to show unusual ranking change patterns (drastic float up or drastic float down in a short time period). To evaluate our method, we focused on collecting apps which have shown such types of ranking change patterns. Specifically, we built a special

data collect system, which consists of two components: **Suspicious App Monitor** and **Metadata Collector**.

The **Suspicious App Monitor** finds apps which have shown unusual ranking changes. Though app stores do not provide a single app's ranking change information, some ASO websites (i.e., websites that offer app store optimization services) provide ranking charts and also list apps that have the most drastic ranking changes over the past time period (e.g., 24 hours). Here we choose one such website, aso100.com, which keeps updating ranking float information for apps in Apple App store. Our monitor collects apps from "top 200 floating up apps in the top free chart", "top 200 floating down apps in the top free chart", "top 200 floating up apps in the top paid chart", "top 200 floating down apps in the top paid chart", respectively, from this website. In this experiment, we calculate the ranking float information based on the ranking charts for the Apple's China app store. Besides an app's basic ranking change information, the monitor also retrieves other information such as app's id and its current chart ranking.

The **Metadata Collector** retrieves the real-time metadata information for each app collected by **Suspicious App Monitor**, through iTunes Search API [2]. For each app, its metadata includes real-time app rating, number of rating, number of reviews, current version, and date.

As earlier mentioned, some normal apps might occasionally experience drastic ranking changes and hence be collected into our dataset. However, later on such apps will be mostly filtered out by setting the threshold for drastic ranking change frequency(DRCF).

Our data collection system ran for a month, from March 13, 2016 to April 13, 2016, at 9:00AM (GMT) each day. The dataset finally contains 37200 records with totally 6651 apps. Then, among these collected apps, by measuring each of their drastic ranking change frequency (DRCF), we filtered out those with low DRCF values, which are likely normal apps. Specifically, in our dataset, the average DRCF of all apps is 0.13 (i.e., 3.9/30). In other words, within a month, the average number of days when apps have showed unusual ranking changes in our dataset is 3.9. In order to filter out the normal apps, we need to set a threshold T_f. When the DRCF threshold T_f increases from 0.033(i.e., 1/30) to 0.2(i.e., 6/30), the number of satisfiable apps decreases sharply from 6651 to 260. When T_f increases continuously (until reaching the largest T_f value (i.e., 30/30=1)), the number of satisfiable apps also keeps decreasing (until to 0).

In this work, we set $T_f = 0.13$ (i.e., 4/30), which resulted in 461 apps with DRCF values above the threshold. These apps will be in included in the set of suspicious promoted apps (SPA) for further investigation.

4.2 Effects of Pairwise Features

In our detection system, RVES, RFS, and RFS are the three core pairwise features to identify suspicious co-promoted pairs among SPAs. Figure6 shows the app pair distribution of RVES, RDS, and RFS, respectively. In each histogram, there are totally 106,030 (i.e., $\binom{461}{2}$) app pairs.

The higher similar data (i.e., rankings, ratings, and review numbers) change pattern any app pair (a, b) demonstrates, the larger similarity score they will get in the corresponding feature. From Figure6, we can see that for all kinds of pairwise features, as the similarity value increases, the number of eligible app pairs decrease sharply. For any pairs from separate TACs, their data changes are independent, so it is hard for them to have highly similar data change patterns. Hence, when the similarity threshold of each feature increases, only app pairs from the same TAC may remain.

We can also see from Figure6 that app pairs in the histogram of RFS have a more dispersed distribution than pairs in RDS and RVES, with more pairs in the high similarity score region. While app pairs in histograms of RDS and RVES show distributions in a more concentrated region, in which apps dominate the area of low similarity scores. Compared with rating and review features, the ranking feature can better reveal relation between apps in one TAC, because collusive attackers might take some unknown actions other than co-rating and co-reviewing to promote apps' rankings. Specifically, in Figure 6(b), when the similarity value of RDS is larger than 8, no any satisfied pair exists. Likewise, in Figure 6(a), no any pair locates in the area where similarity value is larger than 12.

Table 1: detail information of each candidate cluster

#	Cluster size	Review number
1	41	105055
2	38	53421
3	20	226121
4	43	136638
5	130	534023
6	20	163953
7	30	186704

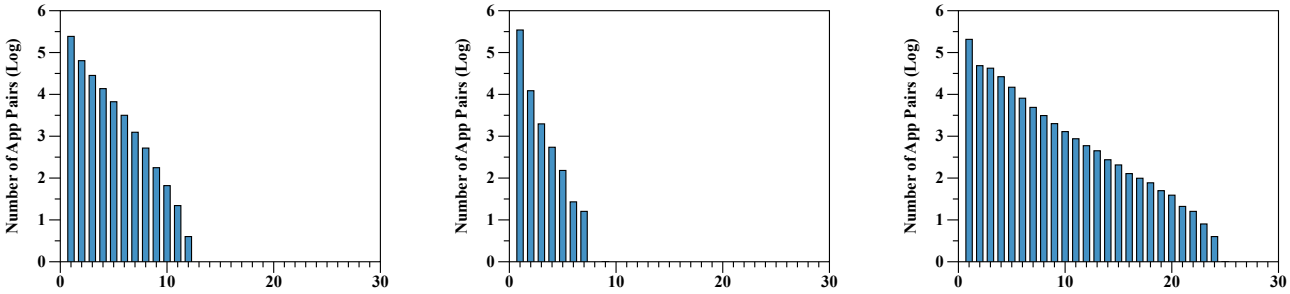
4.3 Collusive Attackers Identification

Based on the above pairwise features, we found 2484 suspicious co-promoted pairs among all suspicious promoted apps. Then we applied the *CandidateClusterCapture* algorithm, and obtained totally 69 candidate clusters. The average size of candidate clusters is 11.4. As discussed in Section IV.D, to filter out clusters that are formed by chance, here we set $T_{size} = 20$. Finally, there are 7 candidate clusters with cluster size larger than 20.

Then, for frequent itemset mining, we collected each app's historical review records from every candidate cluster. Table 1 shows the detailed data of each candidate cluster. Specifically, **Cluster size** is the total number of apps in one cluster, and **Review number** is the total number reviews of all apps in one cluster.

Then, we applied MFIM to each candidate cluster. Specifically, since we did not know exactly how many apps a collusion group promoted each time, here we set the minimum co-reviewed app threshold to 3, and then increased the threshold gradually (e.g., 4 and 5). Tables 2, 3, and 4 in appendix A show the result of each candidate cluster under the corresponding threshold. For example, Table 2 shows the result of all itemsets in each candidate cluster. Each itemset includes at least 20 reviewers who have co-reviewed at least 3 apps.

[2]https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api/



(a) RVES (b) RDS (c) RFS

Figure 6: Histograms of pairwise features. The x-axis denotes the similarity scores and the y-axis denotes the corresponding number of app pairs (on logarithmic scale)

Specifically, in each table, # is the candidate cluster id. **Reviewer number** represents the total number of reviewers of all itemsets in one candidate cluster. **Targeted app number** is the total number of apps from all itemsets in one cluster. **Avg reviewer group** is the average size of all reviewer groups who have co-reviewed apps. **Max reviewer group** is the largest reviewer group in one candidate cluster.

It is worth noting that since fraud campaigns may not organize the same collusion group to promote all apps in one TAC during each promotion period, some itemsets in one candidate cluster have highly similar but not completely identical member apps. For example, in Table4, the Cluster4 has three itemsets, the co-reviewed app set in each itemset is {1075817245, 681580740, 953061503, 981760446, 979100054}, {1075817245, 681580740, 953061503, 979100054, 983956499}, {1075817245, 681580740, 953061503, 981760446, 983956499}. We can see that these three targeted app sets are highly overlapped. In fact, there are six different targeted apps in this cluster, which are highly likely from the same TAC in reality.

In Cluster5, we find several itemsets including app sets with totally different members. This indicates that Cluster5 is likely to cover several app clusters from different TACs in reality, similar to Cluster6. Note that since we focus on identifying members of collusion groups, we will not carefully discriminate different TACs in each candidate cluster.

Specifically, among apps in resulted itemsets, we found 5 apps (1071503483, 831385890, 1098518776, 1094606117, 1091966305) that have been removed from the Apple App China store . This could indicate that the store administrator has spotted ranking promotion activity of these apps and removed them from the store as punishment. However, in reality, many promoted apps can still be available in the store. This is because fraudulent ranking burst campaigns also endeavor to hide the promotion activities, but app store administrators may not have effective ways to discover them timely for the lack of strong evidences. What's more, mistakenly removing an app from the store does cause great negative impacts on users, especially those having downloaded the app. Hence, store administrators will not remove an app easily if they are not absolutely sure about its ranking fraud.

As the threshold of minimum co-reviewed app increases from 3 to 5, the number of satisfiable reviewer groups decreases gradually. Specifically, when the minimum co-reviewed apps threshold reaches 6, we can still find suitable itemsets (i.e., with user group size ≥ 20) in certain candidate clusters. For example, Cluster6 has 11 satisfiable itemsets, with the maximal reviewer group of 79 users. Cluster5 has 171 satisfiable itemsets, with the maximal reviewer group consisting of 327 users. What's more, we can find that users in each itemset had similar rating behavior and most of them posted the highest rating (i.e., 5 stars) to the targeted apps. If we increase our minimum co-review threshold, some reviewer groups might still be obtained. However, setting the threshold too high might filter out some actual collusion reviewer groups. In our future work, we will tune the parameters.

Then, we collect the detailed information of the resulted reviewer groups in itemsets from the Apple App store. Compared with normal reviewers, we find that these reviewers have the following unusual characteristics.

- When we check the historical records of their reviews, we find that these reviewers posted a large number of reviews for various types of apps, which is far beyond normal users' review number. Specifically, most of them have reviewed over 50 apps and some of them have even reviewed over 500 apps.
- They almost always posted the highest rating (i.e 5 stars) to every app they have reviewed despite the apps' real quality (reflected by an app's average rating).
- They reviewed multiple apps together in a short time period. Most of them posted reviews for more than 4 apps in a single day and some of them even reviewed more than 10 apps in certain days. Reviewers from the same groups tend to post reviews on several common apps during the same day.
- Some of their review contents are very generic and are irrelevant with the corresponding apps.

5. RELATED WORK

5.1 Spam and Spammer Detection in Traditional Markets

Various approaches have been proposed on understanding the characteristics of spam and spammers and detecting them in traditional online shopping markets like Amazon,

Yelp, DianPing, etc. For example, works like [8] [9] [23] [24] [25] proposed behavior-based detection methods. In this category, indicative features of spam including user behavior, review content, and product profiles are extracted. Another research thrust [26] [27] has focused on language features. Language used by opinion spammers differs from the language used in genuine reviewers. Graph-based approaches have also been adopted in previous works [6] [7] [13] [5]. Among these works, the problem of fake spammer or spam detection is transformed to a collective classification problem on the graphs. Besides detecting individual spammers, there are also works on identifying spammer groups [12] [14] [4] [28] [22]. Spammers in those websites aim to boost the perceived quality of products by intentionally creating positive comments on these products.

Spam has also been studied in some online social networks like Facebook and Twitter. [29] introduced the concept of "lockstep behaviors", where the actions of multiple accounts are synchronized, to search for spam attacks on Facebook's social graph of over a billion users. [30] presents a system that detects spam on Facebook by identifying messages that are sent in a synchronized fashion. [31] leverages anomalous increases in the follower number of an account to identify collusive follower markets on Twitter. [32] studies other types of collusion ecosystems, in particular the ones that sell likes on Facebook.

5.2 Spam and Spammer Detection in Mobile App Stores

In mobile security, most research concentrates on detecting risky or illegal apps. Previous work like [33] [34] [35] [36] [37] [38] studied malware apps, rebranded apps and garbage apps (Apps that not have a specific functionality or apps with unrelated descriptions and keywords). [1] identified malware apps and search rank fraud in Google Play. [39, 40] focus on detecting collusive attacks where two or more applications interact through Inter-Component Communication (ICC) data flows .

Only a few works focus on spam and spammer detection in mobile app markets. In the mobile app review market, opinion spam is very serious because of huge marketplace and centralized app distribution systems like iTunes, Google Play. The main differences between mobile app markets and traditional markets lie on ranking system, rating mechanism, reviewers' behavior, etc. Hence, previous approaches do not apply directly in mobile app markets.

[20] focuses on unveiling app promotion underground markets by building an automated data collection system to monitor apps in those paid review service providers. The work has greatly unveiled the underground market and statistically analyzed the promotion incentives, characteristics of promoted apps and suspicious reviewers. However, to avoid being detected, fraudulent campaigns are increasingly hiding their traces through other types of secret channels (e.g., messaging apps like WeChat or QQ groups). Indeed, most service provider websites reported in [20] are no longer available and few new sites can be found directly through the Internet.

In our work, we identify collusion group members directly through the app store metadata. What's more, unlike [20] that focuses on detection of rating deviation phenomenon (apps that have deviated rating and users who give deviate rating), we mainly focus on discovery of collusive attackers who aim to promote apps' rankings. In order to promote apps' rankings, fraudulent campaigns often organize collusion attackers to not only boost apps' ratings, but also increase apps' download numbers. To our observation, download numbers seem to influence apps' ranks more significantly. [41] crawled the iOS App Store, compared a baseline Decision Tree model with a novel Latent Class graphical model for classification of individual app spam, whereas our work focuses on collusive spammer discovery.

The GroupTie approach [19] is based on apps' rating features to find collusion groups. The tie graph among users is constructed to capture the collaborative rating promotion or demotion behavior. It however requires that the apps under investigation have at least 8 weeks of lifetime to generate reliable results. In contrast, our approach can also work with apps with short lifetime (e.g., days). Moreover, GroupTie needs to collect co-rating information for *each pair of users* in the app store, which limits its scalability. Since our goal is to discover collusive attackers through app ranking change behavior, we are not required to model the relation between each pair of users in the app store. Instead, our detection merely depends on app meta data. In this sense, our approach can scale better. On the other hand, our approach cannot detect promoted apps that have no drastic ranking changes (e.g., failed promotion), while GroupTie may still be able to detect collusion groups for such apps because it is agnostic to ranking information. Generally speaking, GroupTie and our approach solve the collusion group detection problem from different angles and hence bear different capabilities.

6. DISCUSSIONS

Next we discuss several issues related to the limitations and future improvement of our work. In our work, we set thresholds on pairwise similarity features (i.e., review volume explosion similarity, rating deviation similarity, and ranking float similarity). Any app pair has similarity value higher than the threshold will be considered as suspicious co-promoted pairs and be used to capture TACs for further group members discovery. The difficulty here is to set proper thresholds for pairwise features.

If thresholds are set too high, some real co-promoted app pairs might not be captured. On the contrary, if thresholds are set too low, many innocent apps (or app pairs) might be included accidentally, which will reduce the efficiency of the final collusive attackers discovery. In this work, in order to cover more co-promoted apps, we set the threshold of each feature as the value relative lower than the average similarity value of all app pairs in our suspicious promoted app set (SPA set).

We note that app stores like Apple change their ranking algorithms over time (e.g., once every 1-2 years) while keeping the ranking algorithm secret. To achieve better promotion effectiveness, promotion companies may also change their promotion strategies over time to infer the ranking algorithms.

On the other hand, each app store (Apple App Store, Google Play etc.) runs its own ranking algorithm, so different app stores can react differently to the same promotion attack strategy. Specifically, among these algorithms, important factors (i.e., rating, review amount, download) that affect an app's ranking are almost the same. Since these features do intuitively reflect an app's popularity. To con-

duct ranking fraud in other markets, promotion companies still tend to organize collusive attackers to take similar actions (i.e., organizes a large group of users to download the app, giving high ratings and writing good reviews in a short time period). However, since the weight of each factor might be different in each market's ranking algorithm, to achieve good promotion effectiveness, promotion companies may apply relatively different promotion strategies in each store.

Therefore, in both cases, some thresholds in our method will also need to be adjusted. When an app store adopts our detection method, it may choose and change thresholds over time based on its desirable detection accuracy and acceptable false positive rate.

7. CONCLUSION AND FUTURE WORK

In this work, we proposed a novel approach to identifying attackers of collusive promotion groups in an app store. Our approach exploits the unusual ranking change patterns of apps to identify promoted apps, measures their pairwise similarity, forms targeted app clusters (TACs), and finally identifies the collusive group members. Our evaluation based on a dataset of Apple's China App store has demonstrated that our approach is scalable and able to report highly suspicious apps and reviewers. App stores may use our techniques to narrow down the suspicion lists for further investigation.

We will study the following issues in our future work. First, we will vary the thresholds in our system to evaluate their impacts on detection accuracy. Second, we will use the highly suspicious apps and reviewers reported by our approach as the ground truth, and then train a supervised classification model to learn the thresholds. Third, we will conduct cross-validation of our results with the help of other approaches [19, 41].

Acknowledgment: We thank the reviewers for their valuable comments and our shepherd Danfeng Yao for her guidance on paper revision. The research of Chen, He, and Yang was supported by the National Science Foundation of China (Grants: 51477056 and U1636216), the Shanghai Rising-Star Program (No. 15QA1401700), the CCF-Venustech Hongyan Research Initiative, and the State Grid Corporation Science and Technology Project "The pilot application on network access security for patrol data captured by unmanned planes and robots and intelligent recognition based on big data platform" (Grant No. SGSDDK000KJJS1 600065), and the Pearl River Nova Program of Guangzhou (No. 2014J2200051). The work of Zhu was supported through NSF CNS-1618684 and a gift from Blue Coat System Inc. Daojing He is the corresponding author of this article.

8. REFERENCES

[1] Mahmudur Rahman, Mizanur Rahman, Bogdan Carbunar, and Duen Horng Chau. Fairplay: Fraud and malware detection in google play. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 99–107. SIAM, 2016.

[2] Apple. https://developer.apple.com/app-store/review/guidelines/.

[3] Google. https://play.google.com/about/developer-content-policy.html.

[4] Shebuti Rayana and Leman Akoglu. Collective opinion spam detection: Bridging review networks and metadata. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 985–994. ACM, 2015.

[5] Huayi Li, Zhiyuan Chen, Bing Liu, Xiaokai Wei, and Jidong Shao. Spotting fake reviews via collective positive-unlabeled learning. In *2014 IEEE International Conference on Data Mining*, pages 899–904. IEEE, 2014.

[6] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion fraud detection in online reviews by network effects. *ICWSM*, 13:2–11, 2013.

[7] Geli Fei, Arjun Mukherjee, Bing Liu, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Exploiting burstiness in reviews for review spammer detection. *ICWSM*, 13:175–184, 2013.

[8] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 219–230. ACM, 2008.

[9] Fangtao Li, Minlie Huang, Yi Yang, and Xiaoyan Zhu. Learning to identify review spam. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 2488, 2011.

[10] Jiwei Li, Myle Ott, Claire Cardie, and Eduard H Hovy. Towards a general rule for identifying deceptive opinion spam. In *ACL (1)*, pages 1566–1576. Citeseer, 2014.

[11] Arjun Mukherjee. Detecting deceptive opinion spam using linguistics, behavioral and statistical modeling. *ACL-IJCNLP 2015*, page 21, 2015.

[12] Chang Xu and Jie Zhang. Towards collusive fraud detection in online reviews. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 1051–1056. IEEE, 2015.

[13] Chang Xu, Jie Zhang, Kuiyu Chang, and Chong Long. Uncovering collusive spammers in chinese review websites. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 979–988. ACM, 2013.

[14] Junting Ye and Leman Akoglu. Discovering opinion spammer groups by network footprints. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 267–282. Springer, 2015.

[15] Chang Xu and Jie Zhang. Combating product review spam campaigns via multiple heterogeneous pairwise features. *SDM. SIAM*, 2015.

[16] Number of apps available in leading app stores as of June 2016. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[17] Number of Apple user accounts as of the third quarter 2015. https://www.statista.com/statistics/279689/apple-amazon-and-paypals-user-base/.

[18] Supposed prices for ranking boost services. https://www.techinasia.com/viral-photo-china-shows-manipulate-app-store-rankings-hard.

[19] Zhen Xie and Sencun Zhu. Grouptie: toward hidden collusion group discovery in app stores. In *Proceedings of the 2014 ACM conference on Security and privacy*

in wireless & mobile networks, pages 153–164. ACM, 2014.

[20] Zhen Xie and Sencun Zhu. Appwatcher: unveiling the underground market of trading mobile app reviews. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, page 10. ACM, 2015.

[21] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Ranking fraud detection for mobile apps: A holistic view. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 619–628. ACM, 2013.

[22] Arjun Mukherjee, Bing Liu, and Natalie Glance. Spotting fake reviewer groups in consumer reviews. In *Proceedings of the 21st international conference on World Wide Web*, pages 191–200. ACM, 2012.

[23] Nitin Jindal, Bing Liu, and Ee-Peng Lim. Finding unusual review patterns using unexpected rules. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1549–1552. ACM, 2010.

[24] Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, and Hady Wirawan Lauw. Detecting product review spammers using rating behaviors. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 939–948. ACM, 2010.

[25] Sihong Xie, Guan Wang, Shuyang Lin, and Philip S Yu. Review spam detection via temporal pattern discovery. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 823–831. ACM, 2012.

[26] Myle Ott, Claire Cardie, and Jeff Hancock. Estimating the prevalence of deception in online review communities. In *Proceedings of the 21st international conference on World Wide Web*, pages 201–210. ACM, 2012.

[27] Song Feng, Ritwik Banerjee, and Yejin Choi. Syntactic stylometry for deception detection. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 171–175. Association for Computational Linguistics, 2012.

[28] Chang Xu. Detecting collusive spammers in online review communities. In *Proceedings of the sixth workshop on Ph. D. students in information and knowledge management*, pages 33–40. ACM, 2013.

[29] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 119–130. ACM, 2013.

[30] Qiang Cao. *Understanding and Defending Against Malicious Identities in Online Social Networks*. PhD thesis, Duke University, 2014.

[31] Gianluca Stringhini, Gang Wang, Manuel Egele, Christopher Kruegel, Giovanni Vigna, Haitao Zheng, and Ben Y Zhao. Follow the green: growth and dynamics in twitter follower markets. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 163–176. ACM, 2013.

[32] Emiliano De Cristofaro, Arik Friedman, Guillaume Jourjon, Mohamed Ali Kaafar, and M Zubair Shafiq. Paying for likes?: Understanding facebook like fraud using honeypots. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 129–136. ACM, 2014.

[33] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM, 2011.

[34] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 241–252. ACM, 2012.

[35] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1025–1035. ACM, 2014.

[36] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 221–233. ACM, 2014.

[37] Jonathan Crussell, Clint Gibler, and Hao Chen. Andarwin: Scalable detection of semantically similar android applications. In *European Symposium on Research in Computer Security*, pages 182–199. Springer, 2013.

[38] Suranga Seneviratne, Aruna Seneviratne, Mohamed Ali Kaafar, Anirban Mahanti, and Prasant Mohapatra. Early detection of spam mobile apps. In *Proceedings of the 24th International Conference on World Wide Web*, pages 949–959. ACM, 2015.

[39] Karim O Elish, Danfeng Yao, and Barbara G Ryder. On the need of precise inter-app icc classification for detecting android malware collusions. In *Proceedings of IEEE Mobile Security Technologies (MoST), in conjunction with the IEEE Symposium on Security and Privacy*, 2015.

[40] Amiangshu Bosu, Fang Liu, Danfeng Yao, and Gang Wang. Collusive data leak and more: Large-scale threat analysis of inter-app communications. In *Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2017.

[41] Rishi Chandy and Haijie Gu. Identifying spam in the ios app store. In *Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality*, pages 56–59. ACM, 2012.

APPENDIX

A. RESULT

Table 2: minimum co-review threshold 3

#	Itemset amount	Reviewer number	Targeted app amount	Avg reviewer group	Max reviewer group
1	100	2054	26	62.24	465
2	13	683	6	156	437
3	141	2395	21	96.56	591
4	27	1106	12	110.19	507
5	163	3835	48	230.23	1324
6	133	2253	22	87.06	507
7	21	692	12	163.75	270

Table 3: minimum co-review threshold 4

#	Itemset amount	Reviewer number	Targeted app amount	Avg reviewer group	Max reviewer group
1	51	664	20	38.98	181
2	6	304	6	110	192
3	109	1346	18	61.43	406
4	15	370	8	60.8	181
5	155	2667	41	153.74	882
6	97	1176	18	56.23	222
7	4	211	5	148	178

Table 4: minimum co-review threshold 5

#	Itemset amount	Reviewer number	Targeted app amount	Avg reviewer group	Max reviewer group
1	31	186	14	28.8	48
2	1	89	5	89	89
3	9	204	11	46	88
4	3	96	6	43.3	67
5	113	2554	39	96.9	548
6	49	554	14	45.2	111
7	1	148	5	99.14	148