Crowd Fraud Detection in Internet Advertising

Tian Tian[†], Jun Zhu[†], Fen Xia[‡], Xin Zhuang[‡], Tong Zhang[‡]
[†]State Key Lab of Intelligent Technology & Systems; Tsinghua National TNLIST Lab
Department of Computer Science & Technology, Tsinghua University, Beijing 100084, China
[‡]Baidu Inc., No.10 Shangdi Street, Haidian, Beijing 100084, China
{tiant13@mails,dcszj@mail}.tsinghua.edu.cn, {xiafen,zhuangxin,tongzhang}@baidu.com

ABSTRACT

The rise of crowdsourcing brings new types of malpractices in Internet advertising. One can easily hire web workers through malicious crowdsourcing platforms to attack other advertisers. Such human generated *crowd frauds* are hard to detect by conventional fraud detection methods. In this paper, we carefully examine the characteristics of the group behaviors of crowd fraud and identify three persistent patterns, which are moderateness, synchronicity and dispersivity. Then we propose an effective crowd fraud detection method for search engine advertising based on these patterns, which consists of a constructing stage, a clustering stage and a filtering stage. At the constructing stage, we remove irrelevant data and reorganize the click logs into a surfer-advertiser inverted list; At the clustering stage, we define the sync-similarity between surfers' click histories and transform the coalition detection to a clustering problem, solved by a nonparametric algorithm; and finally we build a dispersity filter to remove false alarm clusters. The nonparametric nature of our method ensures that we can find an unbounded number of coalitions with nearly no human interaction. We also provide a parallel solution to make the method scalable to Web data and conduct extensive experiments. The empirical results demonstrate that our method is accurate and scalable.

Categories and Subject Descriptors

I.5.3 [**Computing Methodologies**]: Pattern Recognition—*Clustering*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Payment schemes, Security*

Keywords

Fraud detection; Internet advertising; crowdsourcing

1. INTRODUCTION

As a way to use the Internet to deliver promotional marketing message to consumers, Internet advertising differs from broadcast and television advertising by its ability to carry out precision marketing. While at the same time it is exposed to greater risks of being influenced by precisely targeted attacks. For search engine advertising, advertisers can present their message to specific individual surfers selected according to their search queries or browsing histories. These advertisements are usually charged by the volume of clicks. However, this pay-per-click billing mechanism increases the risk of fraud since malicious ones can easily raise traffics on specific targets by technological means.

Conventional advertising frauds, such as malwares, auto clickers, etc, usually show specific characteristics on their individual behavior patterns, such as empty cookies, repetitive clicks or hit bursts. These phenomena can be effectively detected by well-studied anomaly detection methods [4, 16, 17]. Unfortunately, when new detection rules are proposed, fraudulent means evolves. Recently, crowdsourcing platforms that can distribute enormous tasks to a large group of web workers provide a more efficient way to handle various tasks including data collection and analysis [8, 10, 21]. Meanwhile, a new type of fraud clicks in Internet advertising has appeared on malicious crowdsourcing platforms[1], where attackers publish micro-tasks of searching and clicking advertisements on a certain search engine, and then web workers accomplish these tasks and are paid per task through the platform.

According to the statistics, expensive adwords on Google, such as insurance, loans, mortgage, etc, could cost an advertiser more than 50 US Dollars per click [11], while malicious platforms only need to pay about 0.1 US Dollars per click to the web workers. This big price gap has lead to a murky business of Internet advertising that attackers publish crowdsourced attacking tasks against competitors to raise their advertising expenses. We refer to these malicious behaviors as *crowd fraud*.

Since simple fraudulent means is well-studied and can usually be filtered out by search engine's anti-fraud mechanisms, crowd fraud has became one of the main sources of customer complaints to search engines and has caused tremendous damage to the marketing environment. There are many differences between the human-generated crowd fraud and the fraudulent behaviors automatically generated by machines, which pose new challenges to crowd fraud detection. For example, crowd fraud often arises from a vast number of attacking sources, but with low fraudulent traffic from each source. The fraudulent behaviors of web workers

[1]For example: http://www.adclickingjobs.com/; http://www.5iads.cn/ (in Chinese).

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741136.

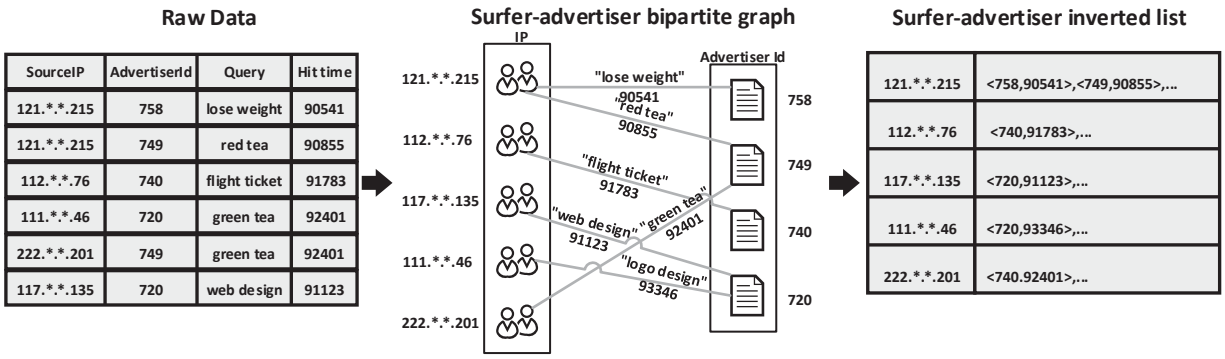


Figure 1: Example of the raw data, the constructed bipartite graph and the surfer-advertiser inverted list.

are noisy and have no distinct regularity, and the fraudulent traffic induced by each worker is often buried in his normal traffic. These differences mean that the short-term or individual behaviors of crowd fraud are almost normal. So conventional anti-fraud systems usually fail to detect crowd fraud. Current commercial crowd fraud detection strategies rely heavily on human interventions, such as the prior knowledge of suspicious queries and complicated artificial filtering rules. Such methods are labor intensive and become invalid rapidly since web workers can easily change their behavior patterns to avoid being detected.

To address the above challenges, we explore the group behaviors of crowd fraud. They are more persistent than the individual behaviors of each worker which may contain substantial randomness. By carefully analyzing the fraudulent behaviors confirmed to be crowd fraud by a commercial detection system, we find some general characteristics about the feature distributions and the network structures of crowd fraud: (1) **Moderateness**: Crowd fraud often aims at advertisers or queries that have moderate hit frequencies; (2) **Synchronicity**: Surfers involving in crowd fraud can usually be grouped into coalitions; in each coalition they usually attack a common set of advertisers; and most of their clicks toward a certain advertiser happen within a common short time period; and (3) **Dispersivity**: Crowd fraud surfers may simultaneously search a series of unrelated queries and click advertisements from different professions.

Based on the above characteristics, we investigate a novel crowd fraud detection method for search engine advertising. From the synchronicity characteristic we can see that crowd fraud has similar properties with coalition attacks. The success of graph based anomaly detection methods on detecting coalitions in many scenarios [13, 15] inspired us to develop our detection method based on the feature-enhanced graph structure of the click-through network. Our method consists of three major stages: (1) **Constructing**: we first remove the logs whose queries have extremely small or large hit frequencies based on the moderateness condition, and then construct a surfer-advertiser bipartite graph as in Fig. 1. Each edge of this graph represents one unique click log, and it contains features such as the hit time and searching query. We also generate the surfer to advertiser inverted list of this bipartite graph for the use of next stage. Each entry of this inverted list represents the click history of one unique surfer. (2) **Clustering**: to find surfer coalitions which exhibit synchronicity, we define a synchronization similarity between click histories and transform the coalition detection problem to a clustering problem. The number of clusters is determined using a nonparametric clustering algorithm, which is inspired by DP-means [12], a small-variance asymptotic version of Dirichlet process (DP) mixtures. (3) **Filtering**: We will find three types of coalitions after clustering, namely, small irrelevant coalitions, fraudulent coalitions, as well as some normal but large coalitions attracted by frequently-presented advertisements. The frequently-presented advertisements in one coalition come from similar business domains such as flight tickets or hospitals. So we examine all large coalitions and remove those with domain centralized advertisers.

Each step of our detection method is scalable with appropriate approximation, and the total number of detected fraudulent clusters is unbounded due to the nonparametric nature of our algorithm. We further implement a parallel version of this method and conduct extensive experiments to show its efficiency and accuracy.

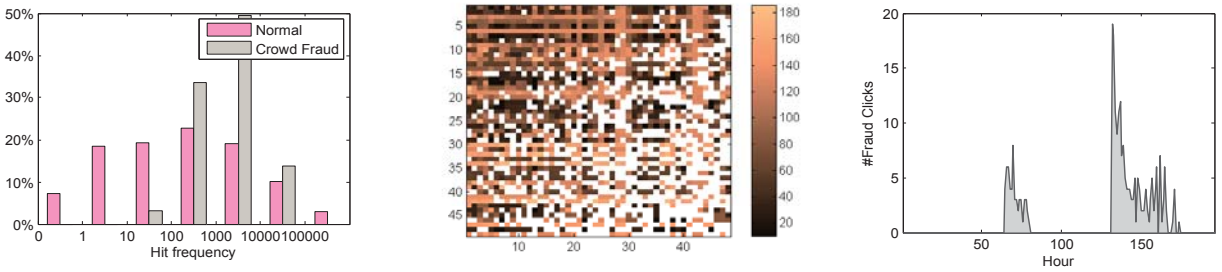
To the best of our knowledge, this paper is the first to formulate the problem of crowd fraud detection in Internet advertising and provides a solution. Our main contributions can be summarized as follows:

1. We formally analyze the crowd fraud problem for Internet advertising, and identify three key characteristics, i.e., moderateness, synchronicity and dispersivity;
2. We present a solution to detect crowd fraud, in which we define a synchronization similarity between click histories and transform the coalition detection problem into a clustering problem, solved by a nonparametric clustering algorithm that automatically decides the number of clusters;
3. We make the method scalable for large-scale search engine advertising by parallelizing the nonparametric clustering algorithm. Extensive experiments are provided to show its effectiveness.

The rest of the paper is organized as: Sec. 2 analyzes the characteristics of crowd fraud; Sec. 3 and Sec. 4 present the detection system for search engine advertising and its parallel implementation; Sec. 5 presents empirical results; Sec. 6 discusses related work; and Sec. 7 provides concluding remarks.

2. CROWD FRAUD CHARACTERISTICS

We define *crowd fraud* as the phenomenon that a group of people driven by economic benefits work together to increase fraudulent traffic on certain targets. Crowd fraud differs from automatically generated fraudulent behaviors in several aspects: (1) The number of web workers involved in an attacking event from crowdsourcing platforms can be very large, while most conventional ways start from much fewer



(a) Traffic moderateness (b) Target synchronicity (c) Temporal synchronicity

Figure 2: Characteristics of crowd fraud: (a) The histogram of the hit frequencies of various queries for both normal traffics and crowd fraud traffics; (b) A corner of the surfer-advertiser bipartite graph built from crowd fraud click relations, where different colors show different hit times and the white color means no click occurred; and (c) A snapshot of the fraudulent click number on one advertiser changes through time.

number of sources; (2) The total fraudulent traffic generated by each web worker in crowd fraud is limited, while other malpractices usually induce hit bursts; (3) Since crowd fraud clicks are generated by real humans, their behaviors have no distinct regularity, while the auto generated traffic usually has detectable repeatability; and (4) Besides the fraudulent traffic, each web worker may also have normal click behaviors, which is also different from automatic clicks. All these differences make the set of conventional features used to detect automatic fraudulent behaviors invalid for crowd fraud detection, which calls for a careful investigation to identify more distinctive and stable characteristics.

This paper focuses on the crowd fraud detection problem in search engine advertising. A main purpose of attackers is to raise fraudulent traffic on competitors to increase their promotional expenses. Thus the attacks usually aim at certain advertisers rather than advertisements.

To compare the differences between normal traffic and crowd fraud traffic, we collect a click dataset from a major Chinese search engine company. The raw click logs on the search engine maintain a history of all click actions that happened within a time period. We investigated the following attributes in the click logs. **Log Id:** We use log Id to identify a unique click log. **IP:** though both cookie Id and IP address can identify different surfers, cookie changes more frequently than IP. So we use the latter in our method, despite the network address translation. **Advertiser Id:** since malpractices mostly aim at certain advertisers, we use advertiser Id to identify their targets. **Hit time:** the time when the click happened. **Query:** the phrase that a surfer searched before his or her click action. We studied a subset of the click logs. After carefully examining the datasets we collected, we find similarities among the crowd fraud traffic on the moderateness, synchronicity and dispersivity characteristics, which provide strong hints to detect crowd fraud effectively. Below, we describe these characteristics in detail.

2.1 Moderateness

In a common search engine session, a surfer searches a query and the engine returns the most relevant results as well as advertisements in search engine results pages (SERPs). Then the surfer may click some links or advertisements which satisfy their requirements or preference. Advertisers can buy keywords from the search engine so that the system will display their message once a surfer's search query matches their keywords. As a result, the advertisers' intentions as well as their advertisement contents are expressed by the corresponding searching queries. We separately count the hit frequencies on each unique query appeared in the crowd fraud dataset and the full dataset, then plot the histogram of hit frequencies in Fig. 2(a).

We can see that most crowd fraud queries are clicked from 50 to 20,000 times during one week, while 50% queries of the full dataset lay outside this range. Thus, crowd fraud traffic differs from normal traffic in that the hit frequencies of its target queries will neither be too small nor too large. It is not too small may be because fraud practices always intend to raise click traffic, so they must click advertisements with considerable frequencies. It is not too large may be because normal traffics on frequently hit queries are very large, while crowd fraud traffics are usually relatively less since the human click efficiency is limited. So the effects of fraudulent traffics will be buried under the normal traffics.

2.2 Synchronicity

We characterize two types of crowd fraud's synchronicity. The first is the target synchronicity. Since crowd fraud starts from malicious crowdsourcing platforms, their behaviors can usually be grouped into sets. Workers on the same platform tend to accomplish similar tasks published on that platform, so their target advertisers are also likely to be similar. The click records can be viewed as a bipartite graph, with surfer vertices and advertiser vertices at two sides, as in Fig. 1, where each edge represents one unique click log, and it contains features such as the hit time and searching query. Then from the view of graph structure, although the click relation between surfers and advertisers forms a very sparse bipartite graph, the links between these crowd fraud workers from the same coalitions to the tasks on the same platform will be dense. To show this phenomenon, we select a dense corner of the bipartite graph built from the crowd fraud dataset, and plot the adjacency matrix of this subgraph in Fig. 2(b), where each cell represents one click from the surfer of the column to the advertiser of the row, and different colors represent different hit times. Our experimental results on real data also verify that these dense structures are common among fraudulent traffics.

The second is the temporal synchronicity. The different colors of cells in Fig. 2(b) show different hit times of the corresponding clicks. We can see that cells in the same row usually have similar colors, which means that the hit times of

Table 1: Example queries, where each row represents the queries from one surfer.

Queries from normal surfers	# Domains
eye cream, cleansing milk, skin care	1
electronic game, card game, game platform	1
headache, dizzy, light-headed	1
Queries from crowd fraud surfers	**# Domains**
franchised outlet, excavator, royal jelly	3
beach BBQ, hospital, calligraphy training	3
toy bear, stainless steel, gym	3

crowd fraud clicks on a same advertiser usually concentrate on a short period. We show this phenomenon more clearly by snapshotting the hit times of fraudulent clicks on one advertiser in Fig. 2(c). We can see that most clicks occurred within short time periods of several hours. This phenomenon also appears on other advertisers, We think this time period may be related to the task's publishing time on malicious crowdsourcing platforms, while their normal traffics usually show periodicity.

2.3 Dispersivity

As a more in-depth examination, we expand the click logs of some frequently appeared individual surfers of both the crowd fraud dataset and the full dataset. After comparison, we find differences between their target business domains. For normal surfers, their click targets during a short period usually focus on one certain business field or a few highly related fields, such as eye cream, cleansing milk, and skin care. However, crowd fraud surfers may click advertisements from many unrelated domains in a short period, such as franchised outlet, excavator and royal jelly. We think this is because web workers do not have real information demand, so if they do many tasks from different advertisers on crowdsourcing platforms, their searching queries as well as their target business domains show dispersivity. Some example searching queries and the number of domains are shown in Table 1.

In summary, the group behaviors of crowd fraud show strong regularities, including moderateness, synchronicity and dispersivity. These regularities differ crowd fraud traffic from normal traffic. Based on these observations, we develop an effective crowd fraud detection system in next section.

3. CROWD FRAUD DETECTION

Our method consists of three major stages: *constructing*, *clustering* and *filtering*, as outlined in Fig. 3.

3.1 Constructing Stage

This stage removes irrelevant data and reorganizes the logs into a surfer-advertiser inverted list.

We first filter out irrelevant logs based on their queries. According to the moderateness condition, click logs with queries whose hit frequency is extremely small or large are barely crowd fraud. So we choose a lower threshold S_L and an upper threshold S_U, and remove the logs with queries whose hit frequency falling outside the range $[S_L, S_U]$. The appropriate threshold values can be found from statistics of fraudulent traffic. Smaller Interval means less data left. This

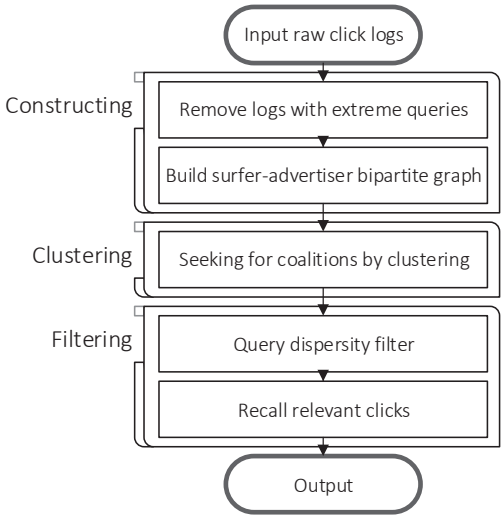


Figure 3: Workflow for crowd fraud detection.

pre-filtering can significantly reduce the data volume. In our experiments, it removes more than 70% click logs.

Then, we reorganize the remaining data as a surfer-advertiser inverted list. As shown in Fig. 1, the click logs represent a surfer-advertiser bipartite graph. The structure of this bipartite graph can be stored as a surfer-advertiser inverted list, which can express the surfers' behaviors more explicitly. In the sequel, we use $\mathcal{G}$ to denote an inverted list; $\boldsymbol{x}_i \in \mathcal{G}$ is the ith entry of $\mathcal{G}$, representing the click history of the ith surfer; each click history $\boldsymbol{x}_i$ consists of an undetermined number of click events; an event $x_{ij} \in \boldsymbol{x}_i$ consists of a target advertiser Id x_{ij}^I and a hit time x_{ij}^T, meaning that surfer i clicked an advertisement at time x_{ij}^T, and this advertisement belongs to an advertiser with Id x_{ij}^I . If surfer clicked advertisements belong to the same advertiser more than one time, we only record one click with the earliest hit time, which is sufficient to represent the intention of all this kind of clicks. A click history is of the form:

$$\{\{Id:74, time:456\}, \{Id:64, time:93\}, ...\}.$$

Suppose we have N unique IPs, M unique advertisers Ids and D click logs after pre-filtering, there will be N lines in $\mathcal{G}$, and each line will contains D/N events on average.

3.2 Clustering Stage

After pre-filtering the logs using the moderateness condition, we proceed to consider the synchronicity condition, which states that surfers involving in crowd fraud usually group into coalitions; in each coalition they usually attack a common set of advertisers; and most of their clicks toward a certain advertiser happen within a common short time period. We formulate the coalition detection as a clustering problem, and solve it by a nonparametric method that can automatically learn the unknown number of coalitions.

3.2.1 Problem Formulation

By the synchronicity property, we need to detect malicious surfer coalitions in which all surfers have similar behavior patterns. We define the *synchronization similarity* (sync-similarity for short) between each pair of click histories and the *coalition center* for every surfers coalition in order to formulize these coalitions.

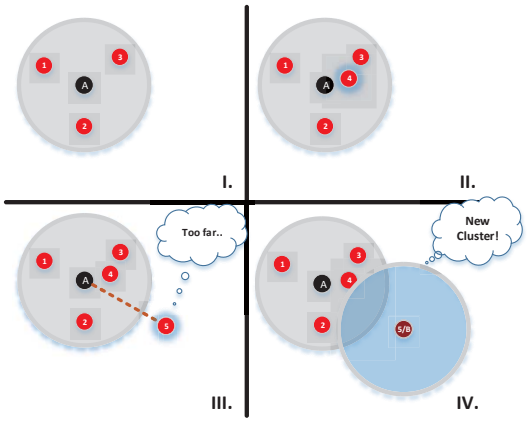


Figure 4: The nonparametric clustering procedure.

Definition 1. **[Sync-Similarity]** Given two click histories $\boldsymbol{x}_i, \boldsymbol{x}_j$ and a time window radius τ, the sync-similarity between them is given by

$$S_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{\substack{x_{is} \in \boldsymbol{x}_i \\ x_{jt} \in \boldsymbol{x}_j}} \mathbb{1}(x_{is}^I = x_{jt}^I,\ |x_{is}^T - x_{jt}^T| < \tau), \qquad (1)$$

where $\mathbb{1}(\cdot)$ is the indicator function.

Numerically, the sync-similarity equals to the number of targets shared by two click histories. Here a common target not only requires a same advertiser Id appearing in both click histories, but also requires the hit times on this advertiser Id in the two click histories fall into a same time period.

Definition 2. **[Coalition Center]** The center $\boldsymbol{\mu_k}$ of coalition k has the same form as the click history. It consists of many click events, and each event μ_{ks} consists of two attributes, i.e., $\mu_{ks} = \{\mu_{ks}^I, \mu_{ks}^T\}$, where the advertiser Id μ_{ks}^I denotes a common attacking target of the coalition and the intrinsic hit time μ_{ks}^T denotes the average hit time of clicks on this advertiser induced by surfers in this coalition.

For all surfers in coalition k, if the sync-similarities between their click histories and the center $\boldsymbol{\mu}_k$ are larger than a similarity threshold T, their behaviors will be regarded as synchronized. Suppose there are K malicious coalitions, z_i denotes the coalition to which the ith surfer belongs, these coalitions should satisfy the following condition:

Condition 1. **[Synchronicity Condition]**

$$S_\tau(\boldsymbol{x}_i, \boldsymbol{\mu}_k) > T, \forall k,\ i,\ s.t.\ z_i = k, \qquad (2)$$

where T is a similarity threshold.

Now our problem is to find coalitions that satisfy Eq. (2), as well as to find their centers. If we consider surfers who reside outside all coalitions as coalitions with only one surfer in it, the original detection problem will became a clustering problem. Each point for clustering represents the click history of one surfer; each cluster represents one coalition; and the cluster center represents the coalition center. For convenient calculation, we assume that all coalitions target on exact w advertisers. This assumption may lead to omit some coalitions which target on less than w advertisers, the reduction of flexibility brought by this assumption can be compensated by a propagation step in the final stage.

3.2.2 *Clustering Algorithm*

There are two main differences between our problem and an ordinary clustering problem. Firstly, the number of clusters is very hard to decide in advance, and it increases about linearly as the number of surfers increases. This is because most surfers behave normally, and then each of them belongs to one new cluster. Secondly, for each surfer in a cluster, the sync-similarity between it and the cluster center should be larger than a radius threshold. So we can not use classic clustering algorithms (e.g., K-means) directly. Inspired by DP-means [9, 12, 23], which is a small variance asymptotics of Dirichlet process mixtures [2], we present a nonparametric method, which solves the following problem:

$$\max_{\mathcal{C}, \{l_c\}_{c=1}^k} \sum_{c=1}^{k} \sum_{i \in l_c} S_\tau(\boldsymbol{x}_i, \boldsymbol{\mu}_c) - \lambda_w k, \qquad (3)$$

where $\mathcal{C} = \{\boldsymbol{\mu}_c\}$ denotes all cluster centers, $l_c = \{i | z_i = c\}$ denotes all indexes of the surfers in cluster c, $\lambda_w = (1-\rho)w$ is a similarity threshold, and $\rho \in [0, 1]$ is a relaxing factor.

Problem (3) admits two properties: (1) The final cluster number is potentially unbounded and automatically determined by the data, and (2) The sync-similarity between each point to its most similar cluster center must be larger than ρw. To see the first property, since the cluster number k is also a variable in this objective, we need to learn the cluster number in real time. So the cluster number will increases when data grows, which is potentially unbounded. To see the second property, once the similarity between a point $\boldsymbol{x}_i$ to its most similar center $\boldsymbol{\mu}_c$ is smaller than ρw, we can create a new cluster whose center is composed by w events in $\boldsymbol{x}_i$, then the similarity between $\boldsymbol{x}_i$ and this new cluster center will be w, and the total change on the objective is $w - S_\tau(\boldsymbol{x}_i, \boldsymbol{\mu}_c) - \lambda_w = \rho w - S_\tau(\boldsymbol{x}_i, \boldsymbol{\mu}_c) > 0$.

These two properties provide us an iterative way to find a local optimum of problem (3). The final algorithm is similar as K-means' two-step update. For each iteration, we first assign each point into its nearest cluster, then update the cluster centers based on the assignments. The main difference between our algorithm and K-means is the assignment step. When a new point $\boldsymbol{x}_i$ comes, we find its most similar cluster center. If the similarity between $\boldsymbol{x}_i$ and this center is smaller than ρw, we generate a new cluster whose center is composed by w events in $\boldsymbol{x}_i$. Then we assign $\boldsymbol{x}_i$ to this new cluster. Fig. 4 illustrates this procedure. When converge, we maintain the remaining clusters whose size larger than a threshold n for the use of the next stage.

The algorithm is outlined in Alg. 1, which consists of a *UpdateCenter* subroutine that solves the subproblem

$$\max_{\boldsymbol{\mu}_c} \sum_{i \in l_c} S_\tau(\boldsymbol{x}_i, \boldsymbol{\mu}_c). \qquad (4)$$

Though the problem can be solved by enumerating all the possible combinations of w events, the time complexity is $O(m^w)$, where m is the average number of unique events in points of this cluster, which is very large. Here we introduce an greedy algorithm, which is approximate but efficient. We sort all advertisers by their occurrence frequencies in all surfers' click histories in this coalition, then use the top w advertisers as the cluster center. The intrinsic hit times are the averages of click times on these advertisers.

As we shall see in experiments, the algorithm with greedy approximation still converges well and finds good solution-

Algorithm 1: Serial solution for coalition detection

Input: $\rho,\ w,\ \tau$
Input: surfer-advertiser inverted list $\mathcal{G}$
$k = 0,\ \mathcal{C} = \varnothing$
while *not converged* **do**
  for $\boldsymbol{x}_i \in \mathcal{G}$ **do**
    $c = \arg\max_{c \in \{1,\cdots,k\}} S_\tau(\boldsymbol{x}_i,\ \boldsymbol{\mu}_c)$
    if $S_\tau(\boldsymbol{x}_i,\ \boldsymbol{\mu}_c) < \rho w$ **then**
      $\boldsymbol{\mu}' = \{\text{Select } w \text{ events in } \boldsymbol{x}_i\}$
      $\mathcal{C} = \mathcal{C} \cup \{\boldsymbol{\mu}'\},\ k = k + 1$
      $z_i = k$
    else $z_i = c$
  for $c \in \{1, \cdots, k\}$ **do** $\mu_c = \text{UpdateCenter}(c)$
Output: Accepted cluster centers $\mathcal{C}$

Procedure UpdateCenter(c)

$\boldsymbol{n} = \boldsymbol{t} = \varnothing$
for $\boldsymbol{x}_i \in \mathcal{G}$ *and* $z_i = c$ **do**
  for $j \in \{1, \cdots, |\boldsymbol{x}_i|\}$ **do**
    $r = x_{ij}^I$
    if $n_r \notin \boldsymbol{n}$ **then**
      Update $\boldsymbol{n}$ and $\boldsymbol{r}$
    $n_r = n_r + 1$
    $t_r = t_r + x_{ij}^T$
for $n_r \in \boldsymbol{n}$ **do** $t_r = t_r / n_r$ $\boldsymbol{y} = \{$the indexes of largest w elements in $\boldsymbol{n}\}$
Output: {events formed by $\boldsymbol{y}$ and times in $\boldsymbol{t}$}

s. By this approximation, the time complexity of *UpdateCenter* is reduced to $O(m \log(m))$, and it can be further reduced to $O(m \log(w))$. If $m \approx D/k$, the overall time complexity for one iteration of Alg. 1 is $O((k + \log(w))D)$. As analyzed above, most items belong to a cluster with size 1, so $k \approx N$. This causes the algorithm still very time consuming. We will describe an approximate and parallel algorithm that we use in practice in the next section.

3.3 Filtering Stage

Usually we can find many large coalitions after clustering, which may contain false alarms. For example, for hot businesses such as games or healthcare, normal surfers with similar information demands may also click a lot of common advertisers that provide similar services. Since the number of these kinds of surfers can be large, it is possible to appear groups of normal surfers which exhibit both the target synchronicity and the temporal synchronicity. This will cause false alarm detections. In order to distinguish these false alarms from real fraud coalitions, we build a query dispersity filter for clusters based on the dispersivity condition.

From our observations, although normal coalitions may exhibit synchronicity, their targets usually focus on one business domain or a few highly related domains, which invalid the dispersivity condition. To evaluate how dispersive a cluster is, we define the *domain coherence coefficient (DCC)* of a set of advertisers.

Definition 3. [**Domain Coherence Coefficient**] Given a set of advertisers $\mathcal{A}$. Let $\mathcal{A}'$ be its largest subset in which all advertisers belong to a same business domain. The domain coherence coefficient of this set is

$$R_{dcc}(\mathcal{A}) = \frac{|\mathcal{A}'|}{|\mathcal{A}|}. \tag{5}$$

Algorithm 2: Query dispersity filter

Input: $\mathcal{Q},\ \lambda,\ w$
Input: candidate cluster centers $\mathcal{C}$
$\mathcal{F} = \varnothing$;
for $\boldsymbol{\mu}_i \in \mathcal{C}$ **do**
  $d_i = \max_{\boldsymbol{q}_j \in \mathcal{Q}} |\boldsymbol{\mu}_i^I \cap \boldsymbol{q}_j|$;
  if $d_i \le \lambda w$ **then**
    $\mathcal{F} = \mathcal{F} \cup \{\boldsymbol{\mu}_i\}$;
Output: fraudulent cluster centers $\mathcal{F}$

Collecting domain information for all advertisers to calculate each cluster's DCC relies on a lot of external interventions. In our method, we choose a more straightforward way. We first build a query-advertiser inverted list $\mathcal{Q}$ from raw click logs. Each entry $\boldsymbol{q}_i \in \mathcal{Q}$ corresponds to a unique query, and the elements $q_{ij} \in \boldsymbol{q}_i$ represent advertisers that have displayed together with query $\boldsymbol{q}_i$. Since these advertisers are related to a common query, their business domains should be related. For cluster i, we use $\boldsymbol{\mu}_i^I = \{\mu_{is}^I | \mu_{is} \in \boldsymbol{\mu}_i\}$ to represent all advertisers targeted by surfers in this cluster, then if we find an entry $\boldsymbol{q}_j$ that satisfies $|\boldsymbol{\mu}_i^I \cap \boldsymbol{q}_j| = p_{ij}$, we can say at least p_{ij} advertisers in $\boldsymbol{\mu}_i$ are related. So by the definition of DCC, we can find a relation that

$$R_{dcc}(\boldsymbol{\mu}_i^I) \ge \max_{\boldsymbol{q}_j \in \mathcal{Q}} \frac{|\boldsymbol{\mu}_i^I \cap \boldsymbol{q}_j|}{|\boldsymbol{\mu}_i^I|}, \tag{6}$$

where $|\boldsymbol{\mu}_i^I|$ always equal to w in our setting. Since larger R_{dcc} means less dispersity, we can choose a threshold λ for R_{dcc} and remove clusters that invalid the dispersivity condition. Alg. 2 outlines the query dispersity filter.

After filtering, the remaining clusters represent the detected fraudulent coalitions. As a final step, we recall click logs that relate to them. For each coalition, the clicks happened between its surfers and the advertisers within its coalition center are recalled as crowd fraud clicks. Here we can also use our fraudulent centers as seeds of a bipartite graph propagation algorithm [14], then iteratively expand the set of suspicious surfers and the set of suspicious advertisers, as well as their intrinsic hit times. The propagation can compensate the reduction of flexibility brought by the assumption that every coalition center consists of w events, and it helps to increase the recall rate of the method.

4. PARALLELIZATION

The real world click logs can be very large, easily rendering a serial algorithm as above inapplicable. So we develop a parallel implementation to make it practical in real scenarios. The constructing stage and filtering stage are easy to parallelize under a distributed computing framework, such as MapReduce [6]. Our discussion will focus on the difficult nonparametric clustering step. We also provide some implementation details of our system.

4.1 Parallel Nonparametric Clustering

The nonparametric nature of our clustering algorithm causes difficulties in its parallelization. The main difficulty

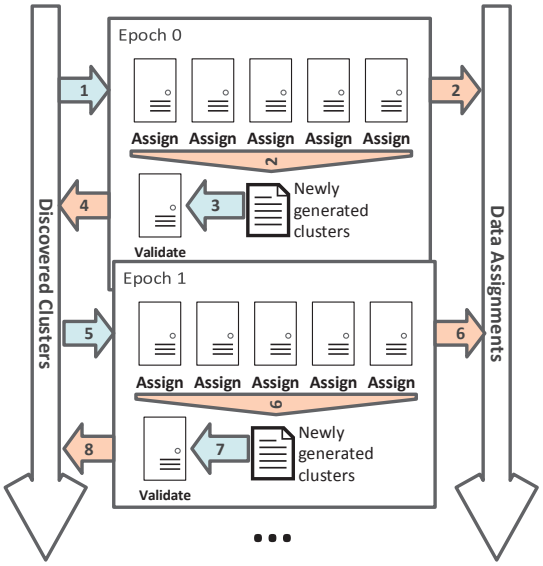


Figure 5: Parallel diagram for the assignment step.

lies in the assignment step, in which newly generated clusters are immediately used by other data points. So we cannot simply divide the assignment tasks to different computing nodes. Here, we use the optimistic concurrency control (OCC) technique [18] to make parallelization possible.

Fig. 4 shows a diagram of parallel assignment step. Using the OOC technique, we divide the assignment step into T epoches and equally partition the whole inverted list $\mathcal{G}$ into T parts denoted by $\mathcal{B}(t), t \in \{1, \cdots, T\}$.

Each epoch only processes one data portion. The newly generated clusters in an epoch are recorded for further use, but not used immediately. That is, each data point processed in this epoch only needs to be compared with clusters generated by the end of the last epoch. Since these clusters' information is known before the start of this epoch, the assignment tasks in one epoch can be distributed without communication. After each epoch, the new clusters generated by different nodes may overlap as they are not compared with each other. So a serial validation step is needed between each two epoches. During validation, running on a master node, every pair of the newly generated clusters are compared; and if their similarity is larger than the similarity threshold, we merge them as well as their associated data points. Pan, et al. [18] has proved this OOC algorithm for DP-means is equivalent to the serial algorithm. Their proof is also suitable for our algorithm.

For our problem, the total number of clusters approximately equals to the number of surfers. So both comparing a point with all cluster centers and delivering cluster center files from slave nodes to master nodes are very expensive. However, since most clusters are induced by normal surfers, they only contain one surfer per cluster and will be removed after clustering. So if the data points are randomly arranged, we can remove smaller clusters and maintain no more than K clusters after each epoch to speed up clustering. This reduction approximation may lead to remove some small fraudulent coalitions, but the algorithm can be speed up significantly. With OCC parallelization and reduction approximation, our final parallel solution for coalition detection is described in Alg. 3.

Algorithm 3: Parallel solution for coalition detection

Input: $T,\ \rho,\ w, K,\ \tau$
Input: $\mathcal{B}(t), t \in \{1 \cdots T\}$
$\mathcal{C} = \varnothing$
while *not converged* **do**
  $k = |\mathcal{C}|, \mathcal{Z} = \varnothing$;
  for *epoch* $t = 1$ *to* T **do**
    $\mathcal{C}' = \varnothing$
    for $\boldsymbol{x}_i \in \mathcal{B}(t)$ **do in parallel**
      $c = \arg\max_{c \in \{1, \cdots, k\}} S_\tau(\boldsymbol{x}_i,\ \boldsymbol{\mu}_c)$
      if $S_\tau(\boldsymbol{x}_i,\ \boldsymbol{\mu}_c) < \rho w$ **then**
        $\boldsymbol{\mu}' = \{$Select w events in $\boldsymbol{x}_i\}$
        $\mathcal{C}' = \mathcal{C}' \cup \{\boldsymbol{\mu}'\}$
        $z_i =$Ref$(\boldsymbol{x}_i)$ //Ref() is the reference function to the coalition center;
      else $z_i = c$
    Validate$(\mathcal{C}', \rho w)$
    $\mathcal{C} = \{K$ largest clusters in $\mathcal{C} \cup \mathcal{C}'\}$
  for $c \in \{1, \cdots, |\mathcal{C}|\}$ **do in parallel**
    $\boldsymbol{\mu}_c =$UpdateCenter(c)
Output: Accepted cluster centers $\mathcal{C}$

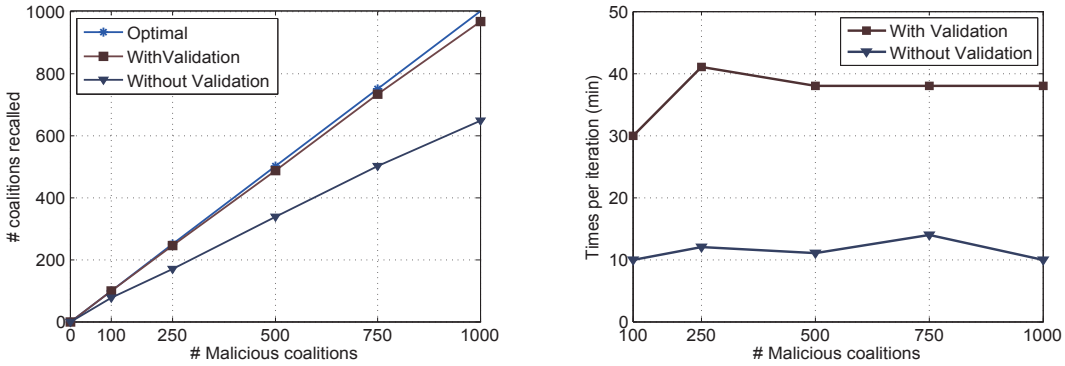
Procedure Validate$(\mathcal{C},\ \lambda)$

for $\mu_i \in \mathcal{C}$ **do**
  $c = \arg\max_{c \in \{1, \cdots, |\mathcal{C}||c \neq i\}} S_\tau(\boldsymbol{\mu}_i, \boldsymbol{\mu}_c)$
  if $S_\tau(\boldsymbol{\mu}_i, \boldsymbol{\mu}_c) > \lambda$ **then**
    Ref$(\boldsymbol{\mu}_i)$=c
    $\boldsymbol{\mu}_i = \varnothing$

Remark 1. Intuitively, the majority of people behave normally, so almost all clusters in our problem are small clusters with only one data point in it, and the mistake induced by the reduction approximation will not be very high. In fact, under a relatively ideal setting that each point comes from an *i.i.d.* distribution, the false negative rate can be estimated as $\frac{n_i}{T}(1 - \frac{TK}{N})(e-1)^{-\frac{n_i}{T}}$, where n_i is the size of a suspicious large cluster i with more than one point in it. We defer the derivation details to Appendix. This result shows that a larger N leads to a larger false negative rate. Since the function xa^{-x} is monotonically decreasing when $x > 0$, a large cluster is less likely to be removed. Moreover, the larger K will lead to a smaller false negative rate, and if $K = N/T$ (i.e., no reduction), the false negative rate is 0.

4.2 Implementation Details

Both the clustering and filtering stages involve a lot of calculations on sync-similarity. To make this computation efficient, we sort every click history when constructing the surfer-advertiser inverted list. The events in a click history are ordered by their target advertiser Ids. When clusters are newly generated or updated, we maintain their orders. So when comparing two click histories, we can array in order all events belonging to them within linear time, and compute their sync-similarity by counting the number of adjacent and similar event pairs. The time complexity for this calculation is $O(d_1 + d_2)$, where d_1 and d_2 are the sizes of two click histories.



(a) Recall

(b) Running speed

Figure 6: Performance on synthetic data: (a) the relation between the number of generated coalitions and the number of discovered coalitions; (b) the per iteration running time for the two clustering settings.

The validation procedure of OCC compares every pair of the newly generated clusters. This is a very time consuming bottleneck of the clustering algorithm. Meanwhile, we empirically found that only very few clusters can be merged during validation. So we can skip the validating procedure to further speed up the algorithm. Through evaluation in the next section, we can see this trick brings significant speed up and still holds an acceptable accuracy.

5. EXPERIMENTS

To verify the effectiveness of our crowd fraud detection system, we conduct a series of experiments on both synthetic and real click data and carefully evaluate its sensitivity, convergence, scalability and accuracy. Most of our algorithms are written in C++ ran under the streaming mode of a modified Hadoop [1] system. All the experiments are conducted on a Hadoop cluster with more than 4,000 nodes, and we use 1,000 mappers and 400 reducers. Raw click logs are stored on the HDFS.

5.1 Synthetic Data

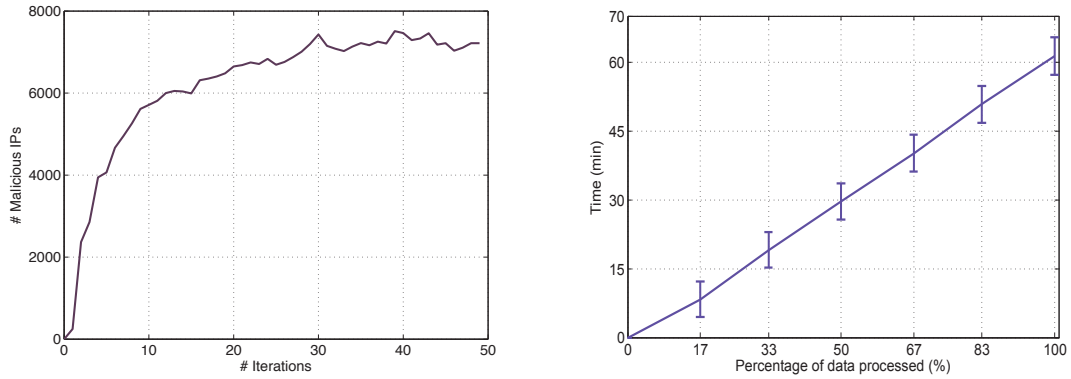
Malpractices for Internet advertising is often hard to distinguish. To judge whether a single click log is malicious or not, an evaluator often needs to expand a lot of logs that correspond to the source IP or target advertiser Id of this log. Even an experienced researcher can only label about 50 logs per hour, so it is too labor intensive to label all the logs in a real large dataset. Thus, it is hard to evaluate the performance of our system, especially its recall rate.

In concern about this issue, we first provide results on synthetic data, where the truth is known, to check the sanity of our algorithms as well as evaluate the recall. Here, we generate several datasets, each of which consists of two parts: normal part and fraudulent part. For the normal part, we simulate 1 million surfers and 100 thousand advertisers. Each normal surfer randomly clicks 10 advertisers, and the hit times for these events are uniformly sampled from $[1, 240]$, which represents a time period of 240 hours. Since advertiser Ids and hit times are randomly sampled, it is unlikely to appear coalitions within this part of click logs. As for the fraudulent part, we generate L coalitions, each of which consists of 200 surfers and 5 advertisers and assigns each advertiser a random intrinsic hit time. Each surfer in a coalition clicks all the advertisers in that coalition, and the clicks target on each advertiser happen within a 6 hours time period centered at the advertiser's intrinsic hit time. In the experiment, we set L at 100, 250, 500, 750 and 1,000, leading to 5 different datasets.

The synthetic experiments only use the clustering stage of the system to evaluate recall rate, and due to the data generating mechanism, the coalition detection precision on this synthetic dataset should be near 100%. So we do not test the result precision on this dataset, and the evaluation on the system's precision is left to the real data experiments. As stated in Sec. 4.2, the validation procedure is the computational bottleneck, and only very few clusters can be merged in this procedure. So we test two types of clustering methods, one with the validation procedure and one without it. The parameters are set as: $T = 4,\ n = 50,\ w = 5,\ \tau = 8,\ K = 10,000$, that is, each iteration is divided into 4 epoch, the minimum number of surfers in one malicious coalition is 50, the minimum number of advertisers is 5, the time window radius is 8 hours, and the upper bound for the number of clusters maintained during learning is 10,000. Other small clusters are discarded on the fly. The discovered cluster numbers and the per-iteration running times for the 5 synthetic datasets with different numbers of fraudulent coalitions are shown in Fig. 6(a) and Fig. 6(b).

Sensitivity Fig. 6(a) shows the relation between the true number of coalitions and the number of coalitions discovered by the algorithm, where the diagonal line shows the optimal performance (i.e., the number of discovered coalitions equals to the true number of coalitions). We can see the number of discovered coalitions for both settings with/without the validation procedure increase about linearly when the number of true coalitions increase. The algorithm with validation achieves a recall rate of nearly 100% on average. This result shows the ability of our algorithm on detecting unknown crowd frauds. The recall rates of the algorithm without validation are relatively lower, e.g., it is about 82% on the dataset with 100 coalitions and 65% on the dataset with 1,000 coalitions. Since the occurrences of coalitions in real click logs are much rarer than the occurrences we defined in the synthetic datasets, the performance of the algorithm without validation is also acceptable.

Efficiency Fig. 6(b) shows the per iteration running time for the two types of clustering algorithms. We can see the



(a) Convergence (b) Scalability

Figure 7: Performance on real click data: (a) shows the number of malicious IPs we found during iterating. (b) shows the the overall running time after each epoch.

average running time for the algorithm with validation is about 3 times larger than that of the algorithm without validation. Considering the overhead time cost induced by the MapReduce implementation, the efficiency gap brought by the different algorithm strategies could be much larger. So when testing on the real click dataset, we use the clustering algorithm without validation for the clustering stage.

5.2 Real World Data

We applied our method to advertisement click logs of a major Chinese commercial search engine. We use the dataset in Sec. 2, which consists of around 400 million logs, and each log is composed by log Id, source IP, target advertiser Id, hit time and searching query.

At the constructing stage, we set $S_L = 50$ and $S_U = 10,000$ to remove queries whose hit frequencies are extremely small or large, these parameters are decided by the statistics of traffic moderateness on the collected crowd fraud dataset, such as in Fig. 2(a). Then we generate a surfer-advertiser inverted list for further use. At the clustering stage, we set parameters as $n = 3,\ w = 8,\ \rho = 0.8,\ \tau = 9$, that is, each malicious coalition needs to contain at least 3 surfers and 8 advertisers, and each surfer needs to click at least 80% of these advertisers during a time period of radius 9 hours. These parameters are selected to maximize the recall rate of coalitions on the given crowd fraud dataset. In the experiment, we run 50 iterations at the clustering stage. Each iteration is divided into 6 epoches and all validation procedures are skipped. As for the reduction approximation, we maintain 10,000 largest clusters at most during clustering, other small clusters are discarded in real time. So $T = 6,\ K = 10,000$. These parameters are decided by our computing resources, which make sure that the program stops running within a reasonable time period. For the filtering stage, we choose domain coherence coefficient threshold $\lambda = \frac{3}{8}$. For the ease of evaluating, the results are not expanded by propagation.

The experimental results are shown as following.

Convergence Fig. 7(a) shows the number of malicious IPs we found after each iteration. Since the numbers of target advertisers in each coalition are the same, this curve actually reflects the objective value of the problem. We can see this curve converges after about 30 iterations, which shows the convergence of our clustering algorithm.

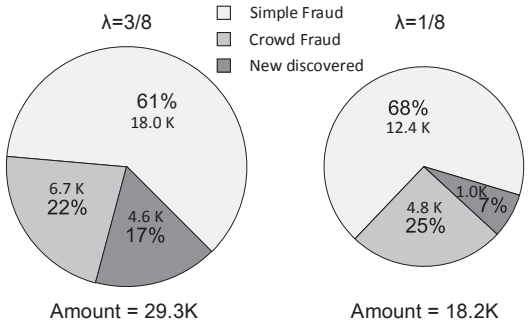


Figure 8: Diagram for the results on real data.

Scalability To show the running time of our system scales up linearly when data size scales up, we record the overall running time after each epoch among 50 iterations. Fig. 7(b) shows the average running times. We can see that the running time increases about linearly when the percentage of processed data increases. This result shows our detection system is scalable when data grows.

Accuracy To evaluate the accuracy of detected frauds, we compare our results with a rule-based system, which can distinguish simple automatically generated fraud and the crowd fraud in the full traffics. After the clustering stage, we found 231 malicious coalitions; after filtering stage, 210 coalitions are removed by query dispersity filter. The remaining 21 coalitions contain 29.3 thousand click logs in total. Through our observation, 20 of them satisfy the dispersivity condition, which correspond to 98.7% of click logs. The test results of these click logs are shown in Fig. 8.

24700 logs meet the results of the baseline system, which account for 83% of our results. 18000 of them are confirmed to be simple fraudulent clicks, they are caused automatically by malwares or spywares; 6700 of them are confirmed to be more hidden fraudulent clicks, they are very likely to be caused by crowd fraud behaviors. The rest 4600 logs are not found by the baseline methods. Since manual evaluating is very expensive, we randomly sampled 200 clicks among them to judge whether they are fraudulent clicks. The final manual evaluating result, provided by several domain experts, confirms that the accuracy on this subset is about 90%.

During manual evaluating, we find that all bad cases of previous trial are induced by one false alarm coalition. So we use a tighter query dispersity filter with $\lambda = \frac{1}{8}$, then all result coalitions satisfy the dispersivity condition. The evaluation of this result is also shown in Fig. 8. It shows that the overall accuracy of this trial is even higher than that of the last one.

6. RELATED WORK

Crowdsourcing is rising field that includes a series of problem solving strategies, business models or activities [7, 8, 20]. Online working platforms, such as Amazon Mechanical Turk (MTurk) [2], provide a new way to distribute enormous tasks to a crowd of workers, and each worker only needs to finish a small part of the entire task, resulting in faster and cheaper solutions. This mode is particularly suitable for large but repetitive tasks like labeling, evaluating, etc. However, the popularisation of crowdsourcing also gives rise to new types of security problems on the web. Crowdturfing [22] is the way to hire web workers through malicious crowdsourcing platforms to initiate fake campaigns. This phenomenon has brought serious problems to social networks and electronic commerce. In this paper, we focus on the fraudulent click problem for Internet advertising which also starts from malicious crowdsourcing platforms.

Anomaly Detection [4] is a well studied topic in various application domains. It aims to find patterns in data which do not conform to the expected behavior. Many methods have been developed for different applications such as fraud detection for credit cards, insurance, health care, intrusion detection for cyber-security, etc. Fraud detection for Internet advertising is an important application of anomaly detection. Rule-based methods such as detecting empty cookie and hit inflations [17] are useful to find simple attacks. Graph-based detection methods are also tried, e.g., people use co-clustering or prorogation methods [13] to find suspicious dense connections between surfers and targets. As for other related areas, CopyCatch [3] is a successful approach which focuses on detecting fake likes on Facebook. They provided a way to find coalitions who demonstrate steplock behavior. In our work, we build a detection system aimed at crowd fraud detection for Internet advertising which is unbounded and scalable.

Clustering is one fundamental problem in data mining. It groups a set of objects so that objects in the same group are more similar to each other than to those in other groups. K-means is one of the most popular clustering methods, but it needs to pre-specify the number of clusters. Many nonparametric methods has been studied to overcome this limitation, such as Dirichlet process mixtures (DPM) [2], which is the nonparametric version of the Gaussian mixture model and can learn the cluster number. Mean-shift clustering is an approach which merges clusters on the fly [5], with wide use in computer vision. DP means [12] is another nonparametric clustering method derived by a small variance asymptotics of DPM. It reserves the nonparametric essence and dramatically increases running speed compared to DPM. Similar small variance techniques have been extended to many other nonparametric Bayesian models [19, 23]. Inspired by DP-means, our work converts crowd fraud detection to clustering by defining the sync-similarity, and proposes a fast and parallel algorithm for real Internet advertising.

[2]http://www.mturk.com/.

7. CONCLUSIONS

We formalize the crowd fraud detection problem in Internet advertising, and carefully analyze the behaviors of crowd fraud to identify three important characteristics, including moderateness, synchronicity and dispersivity. Based on the findings, we further develop an effective detection system, which consists of three major steps, data constructing, clustering and filtering. One key step is convert the original coalition detection problem to a clustering problem. Our nonparametric clustering method ensures to learn the unknown number of coalitions, which is important for real-world applications. Finally, we build a parallel detection system for search engine advertising, and conduct extensive experiments to evaluate its performance on both synthetic and real click data. Results show that our system can find fraudulent clicks with a high accuracy, and it can be scaled up to large datasets.

In the future, we would like to do more analytical work on crowd fraud, and test the performance of our system on datasets of shorter time period. We'd also like to extend current algorithms to other fraud detection problems such as click fraud or social network spams.

Acknowledgments

The work was supported by the National Basic Research Program (973 Program) of China (No. 2013CB329403), National Natural Science Foundation of China (Nos. 61322308, 61332007), and the Tsinghua National Laboratory for Information Science and Technology Big Data Initiative.

APPENDIX

A. FALSE-NEGATIVE RATE ANALYSIS

Let s be the number of large clusters with size larger than 1, d be the number of small clusters with only one point in each cluster, and n_i be the size of the ith cluster. Assume that the N data points are *i.i.d.* sampled from a multinomial distribution with $s + d$ parameters $\epsilon_1, \epsilon_2, \cdots, \epsilon_s, \epsilon_0, \cdots, \epsilon_0$, where $\epsilon_i = \frac{n_i}{N}$ denotes the probability that a point is sampled from cluster i, and $\sum_{i=1}^{s} \epsilon_i + d\epsilon_0 = 1$. Since the number of clusters is large, each ϵ_i is small.

After the tth epoch, we maintain only K largest clusters and remove about $\frac{N}{T} - K$ small clusters, so the probability that a small cluster will be removed is $1 - \frac{TK}{N}$. For the ith cluster, the probability that its current size equals to one is $\frac{N}{T}\epsilon_i(1-\epsilon_i)^{t\frac{N}{T}-1} \approx \frac{N}{T}\epsilon_i \exp\left(-t\frac{N}{T}\epsilon_i\right) = \frac{n_i}{T}\exp\left(-t\frac{n_i}{T}\right)$, where we use the fact that the binomial distribution converges towards the Poisson distribution as the number of trials goes to infinity while the product $N\epsilon_i$ remains fixed. Thus the probability that this cluster will be removed after this epoch is $(1 - \frac{TK}{N})\frac{n_i}{T}\exp\left(-t\frac{n_i}{T}\right)$. Considering all T epoches, the probability that the ith cluster will be removed is $(1 - \frac{TK}{N})\frac{n_i}{T}\sum_{t=1}^{T}\exp\left(-t\frac{n_i}{T}\right) \leq (1 - \frac{TK}{N})\frac{n_i}{T}(e-1)^{(-\frac{n_i}{T})}$. Therefore, the false negative rate is the one as we sated in the Remark 1.

B. REFERENCES

[1] Apache hadoop. http://hadoop.apache.org/.

[2] C. E. Antoniak. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The Annals of Statistics*, pages 1152–1174, 1974.

[3] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. CopyCatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*, 2013.

[4] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[5] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. on PAMI*, 17(8):790–799, 1995.

[6] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011.

[8] J. Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006.

[9] K. Jiang, B. Kulis, and M. I. Jordan. Small-variance asymptotics for exponential family Dirichlet process mixture models. In *NIPS*, 2012.

[10] E. Kamar, S. Hacker, and E. Horvitz. Combining human and machine intelligence in large-scale crowdsourcing. In *AAMAS*, 2012.

[11] L. Kim. The most expensive keywords in Google adwords. http://www.wordstream.com/blog/ws/2011/07/18/most-expensive-google-adwords-keywords, 2011.

[12] B. Kulis and M. I. Jordan. Revisiting k-means: New algorithms via Bayesian nonparametrics. In *ICML*, 2012.

[13] X. Li, Y. Liu, M. Zhang, and S. Ma. Fraudulent support telephone number identification based on co-occurrence information on the web. In *AAAI*, 2014.

[14] X. Li, M. Zhang, Y. Liu, S. Ma, Y. Jin, and L. Ru. Search engine click spam detection based on bipartite graph propagation. In *WSDM*, 2014.

[15] A. Metwally, D. Agrawal, and A. El Abbadi. Detectives: detecting coalition hit inflation attacks in advertising networks streams. In *WWW*, 2007.

[16] A. Metwally, D. A. A. El Abbadi, and Q. Zheng. Hide and seek: Detecting hit inflation fraud in streams of web advertising networks. *Technical Report, University of California at Santa Barbara*, 2006.

[17] R. Oentaryo, E.-P. Lim, M. Finegold, D. Lo, F. Zhu, C. Phua, E.-Y. Cheu, G.-E. Yap, K. Sim, M. N. Nguyen, et al. Detecting click fraud in online advertising: a data mining approach. *JMLR*, 15(1):99–140, 2014.

[18] X. Pan, J. E. Gonzalez, S. Jegelka, T. Broderick, and M. Jordan. Optimistic concurrency control for distributed unsupervised learning. In *NIPS*, 2013.

[19] A. Roychowdhury, K. Jiang, and B. Kulis. Small-variance asymptotics for hidden Markov models. In *NIPS*, 2013.

[20] R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *EMNLP*, 2008.

[21] Y. Tian and J. Zhu. Learning from crowds in the presence of schools of thought. In *SIGKDD*, 2012.

[22] G. Wang, C. Wilson, X. Zhao, Y. Zhu, M. Mohanlal, H. Zheng, and B. Y. Zhao. Serf and turf: crowdturfing for fun and profit. In *WWW*, 2012.

[23] Y. Wang and J. Zhu. Small-variance asymptotics for Dirichlet process mixtures of SVMs. In *AAAI*, 2014.