Big Data and Information Analytics
©American Institute of Mathematical Sciences
Volume **1**, Number **2&3**, April & July **2016**
doi:10.3934/bdia.2016006
pp. **227–245**

DETECTING COALITION ATTACKS IN ONLINE ADVERTISING: A HYBRID DATA MINING APPROACH

Qinglei Zhang and Wenying Feng*
Department of Computing & Information Systems
Department of Mathematics, Trent University
Peterborough, Ontario K9J 0G2, Canada

(Communicated by Wenxue Huang)

Abstract. Coalition attack is nowadays one of the most common type of attacks in the industry of online advertising. In this paper, we attempt to mitigate the problem of frauds by proposing a hybrid framework that detects the coalition attacks based on multiple metrics. We also articulate the theoretical basis for these metrics to be integrated into the hybrid framework. Furthermore, we instance the framework with two metrics and develop a detection system that identifies the coalition attacks from two distinguish perspectives.

1. **Introduction.** As an opportunity for organizations and companies to reach out millions of potential customers at a low cost, online advertising costs for nearly one-quarter of the global ad spent [1]. Figure 1 illustrates the process of online advertising. The advertiser manages the advertising campaigns for various brands, while internet publisher provides the inventory for hosting the ads. Moreover, advertisers and publishers can communicate directly or indirectly. Exchangers such as Google's DoubleClick [2] and Yahoo!'s RightMedia [3] are involved as intermediaries who pair the publishers' ad request with the most profitable advertiser bid. During the life-cycle of an advertisement, an impression (view) means the ad is displayed on the publisher's webpage when a visitor loading the page. Consequently, when the visitor clicks on the ad a click event is fired and the visitor is taken to the brand's landing page. Finally, a conversion is recorded if the visitor performs certain actions on the brand's websites.

In the context of online advertising, a buyer/seller relationship is established between the advertiser and publisher [26]. With accordance to the three basic events during the advertising life-cycle, the three price models in the industry are PPM (Pay-Per-Mille[1]), PPC (Pay-Per-Click), and PPA (Pay-Per-Action) [7, 22, 27], respectively. Since publishers earn revenue based on the number of views, clicks, or actions that are associated with the ads, publishers have strong incentives to maximize those numbers [15, 16]. While publishers can apply legitimate methods to attract more traffic to their websites, malicious publishers attempt to make more money by generating invalid (fraudulent) traffic. Such publisher frauds can be further categorized as impression frauds, click frauds, and conversion (action) frauds [7, 9, 22]. Various types of fraud attacks from both industry and academia

2010 *Mathematics Subject Classification.* Primary: 93E10, 93E35; Secondary: 62M20, 62H30.
Key words and phrases. Data mining, fraud detection, internet publisher, impression, MapReduce, online advertising.

* Corresponding author: Wenying Feng.

[1]Mille means thousand impressions in the context.

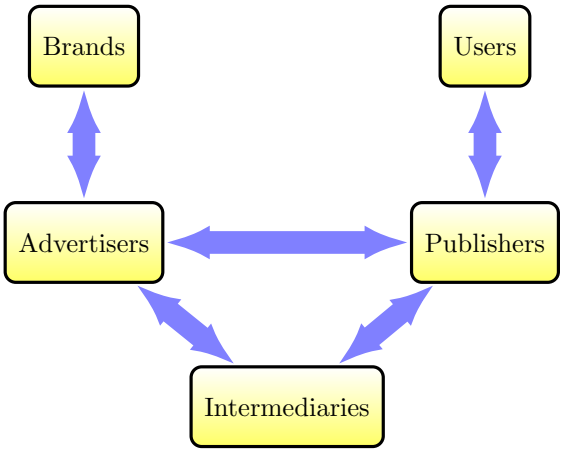


FIGURE 1. The process of online advertising

have been identified in the literature (e.g. [20, 21, 24]). To generate traffic to publishers' websites, attacks can be launched either on publishers' websites or on machines that are controlled by malicious publishers, and attacks can be performed either by real human or bots. More details of some existing online advertising frauds are discussed in [26].

In online advertising industry, the number of views, clicks, and actions of ads is critical to the revenue of advertisers. Advertisers use several metrics such as CPM (Cost Perm Mille), CTR (Click Through Rate), and CR (Conversion Rate) as the KPIs (Key Performance Indicators) to evaluate the performance of their market campaigns. However, the inflated numbers of views, clicks, and actions that are generated by fraudulent publishers hinder the reliability and effectiveness of those performance metrics. Publisher fraud is a severe problem for advertisers and worth the endeavour to detect and prevent.

Publisher frauds are mainly launched with two key entities: fraudulent sites and machines. With respect to correlations between machines and sites, publisher frauds can be classified as coalition attacks and non-coalition attacks. As illustrated in Figure 2, the non-coalition attack corresponds to a one-to-one relationship, and the coalition attack represents a many-to-many relationship. An elementary attack can be easily detected and blocked by identifying repeated views, clicks, or actions from one machine on one site. To avoid being detected, fraudulent publishers attempt to blur the strong correlation between the fraudulent sites and machines. A single fraudulent publisher needs to simulate fake visitors, or frequently change the identification of users to blur the relationship, which significantly increases the difficulty and cost for launching the attacks. On the other hand, several fraudulent publishers can make a collusion and share their resources, which also blur the correlation between a pair of fraudulent sites and machines. Since coalition attacks can be launched without developing sophisticated techniques, they have become more and more popular recently. While coalition attacks can be deployed easily and effectively, the detection of such attacks is more challengeable as an open issue.

Both active and passive approaches can be used to detect the fraudulent problems in general. To verify the validity of the traffic, active approaches usually require the interaction with the web pages and/or visitors (e.g. [20]). Passive detection

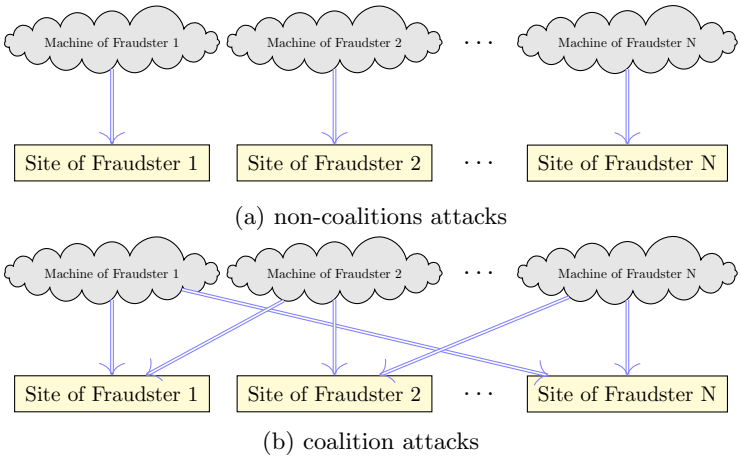


FIGURE 2. Different types of publisher frauds

approaches, on the other hand, apply data analysis/mining techniques to recognize the behaviour pattern of traffics (e.g. [11, 19, 21, 23, 25, 27]). While active detection techniques typically adopt the signature based detection mechanism to mark the fraudulent traffic, passive detection techniques can use both signature based (e.g. [11]) and anomaly based (e.g. [23]) approaches to identify frauds. Moreover, the active detection may change the model of advertising network and compromise the privacy of visitors, which is sometimes impractical. On the other hand, the premise for data mining approach to detect frauds is to identify correlations between publishers and visitors. Due to the flexibility and feasibility of the passive detection measurement, we adopt the data mining approach in this paper. In particular, we propose an algorithm to detect correlations of publishers and visitors that identify coalition attacks.

The paper is structured as follows. Section 2 introduces background and related notations of coalition attacks and parallel computing. Section 3 presents the proposed detection system for coalition attacks. Next, in Section 4 we demonstrate the evaluation results of our new techniques. Finally, some conclusions and future work are discussed in Section 5.

2. **Background.**

2.1. **Identification for coalition attacks.** A classical way to detect publisher frauds is to block suspicious domains when their impressions and CTR exceed a hard (static) threshold. However, fraudsters can easily comprise the classical detection tools by mimicking legitimate criteria. To identify malicious intents of fraudsters, the key is to identify the association between fraudulent websites and machines. In the literature, two types of metrics have been explored. One set of metrics is derived from the graphic topology (e.g. sets of neighbours and paths between nodes). Those metrics are more generic and can be applied to any domain. In our context, real network data indicates that two legitimate sites should have different sets of visitors [17, 18]. Therefore, the degree of overlap among visitors of two sites can be used to measure the collusion of two sites. Different similarity metrics having been proposed for generic link prediction problems, such as Jaccard coefficient [8]

and Adamic [6] can be used to estimate the similarity of visitors for two publishers. For example, Jaccard coefficient is employed for the coalition attack problem in [17]. On the other hand, another set of metrics is associated with attributes of vertices and links, which requires the domain knowledge. In the context of e-commerce, as ROI (Return of Investor) is one of the key concerns, coalition groups incentively have high ROI than normal publishers. Therefore, different variants of ROI can be derived as the metrics for identifying frauds. For example, in [12], a metric called GPR (Gain-Per-Resource) is proposed to detect the coalition attacks.

In this paper, we propose a hybrid approach that combines the metrics from different perspectives to detect the coalition attacks. In particular, the similarity and GPR are selected as two representative metrics in our technique. To systematically and automatically detect coalition attacks, we give formal definitions for identifying coalition attacks based on similarity and GPR in the rest of this section.

The ratio of gain and cost is an estimator of ROI (Return of Investment), which can be formally defined as:

$$GPR(g) = \frac{W(g)}{R(g)}, \tag{1}$$

where $W(g)$ denotes the gain of group g and $R(g)$ denotes the amount of resources for the group. Given the definition of the GPR metric, the GPR value of coalition groups will exceed a threshold, denoted as θ_{gpr}. However, a group that consists of coalitions and normal publishers may still have a high GPR. In order to exclude such cases, a definition of GPR-GROUP is further introduced.

Definition 2.1. A group is called a GPR-GROUP iff

$$\forall(g \mid g' \subset g : GPR(g') < GPR(g)),$$

where g' denotes any subgroup of g.

A coalition group can be defined in terms of GPR.

Definition 2.2 (gpr-based identification)**.** (1) A group of publishers and visitors, denoted as g, is called a coalition iff

$$GPR(g) \geq \theta_{gpr} \text{ and } g \text{ is GPR-GROUP.}$$

(2) A coalition g is a maximal coalition iff

$$\neg\, \exists(g'' \mid g \subset g'' : g'' \text{ is a coalition}).$$

Let S_{p_1} and S_{p_2} denote two sets of visitors to publishers p_1 and p_2, respectively. The Jaccard coefficient, $\frac{|S_{p_1} \cap S_{p_2}|}{|S_{p_1} \cup S_{p_2}|}$, captures the traffic similarity of p_1 and p_2. However, since the number of repeated visitors is also a signal of frauds in online advertising, the set model for publishers' visitors will lose important information for detecting frauds. Therefore, we use a variant of Jaccard coefficient w.r.t a bag model[2] to represent publishers' visitors. Formally, let B_{p_1} and B_{p_2} denote two bags of visitors to publishers p_1 and p_2, respectively. The similarity of a pair of sites p_1 and p_2 is modelled by the formula:

$$\text{Similarity}(p_1, p_1) = \frac{|B_{p_1} \sqcap B_{p_2}|}{|B_{p_1} \sqcup B_{p_2}|}, \tag{2}$$

[2]The elements in bag models can be duplicated.

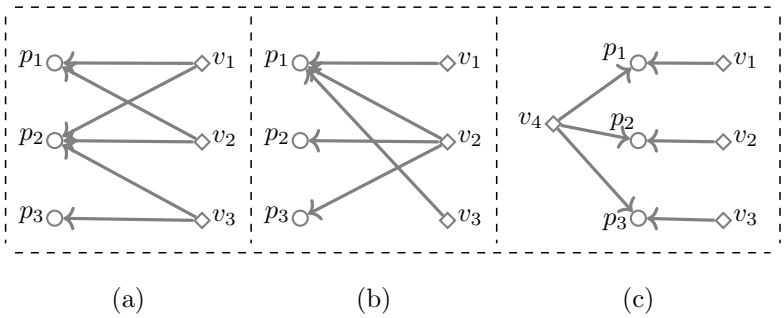


FIGURE 3. Different Traffic Patterns with/without Coalition Attacks

where $\sqcap$ and $\sqcup$ denote the intersection and union of bags. We use a pair (n, e) to represent an element in a bag, where n indicates how many times e repeats in the bag. Then $\sqcap$ and $\sqcup$ can be defined as the following:

$$\begin{aligned}
&\forall((n,e) \mid (n,e) \in A \sqcap B : \quad \exists(m_1, m_2 \mid (m_1,e) \in A \wedge (m_2,e) \in B : n = min(m_1,m_2)))\\
&\forall((n,e) \mid (n,e) \in A \sqcup B : \quad ((n,e) \in A \wedge \neg\, \exists(k \mid k \in N : (k,e) \in B))\\
&\qquad\qquad\qquad\qquad\quad \vee((n,e) \in B \wedge \neg\, \exists(k \mid k \in N : (k,e) \in A))\\
&\qquad\qquad\qquad\qquad\quad \vee(\exists(m_1, m_2 \mid (m_1,e) \in A \wedge (m_2,e) \in B : n = max(m_1,m_2)))).
\end{aligned}$$

With the definition of similarity metric, a pair-wise similar group means that the similarity of each pair in the group is greater than a threshold, denoted by θ_{sim}. Since two legitimate sites only have negligible similarity, it is unlikely that a random group is pair-wise similar. On the other hand, coalition groups share the resources, which would generate similar traffics inherently. Therefore, coalition groups can be identified with regard to the traffic similarity.

Definition 2.3 (similarity-based identification)**.** A group of publishers g is a coalition group iff

$$\forall p_1, p_2 : (p_1 \in g) \wedge (p_2 \in g) \wedge (p_1 \neq p_2) \Longrightarrow \text{similarity}\ (p_1, p_2) \geq \theta_{sim}.$$

2.2. **Examples.** To illustrate the effectiveness of the hybrid approach, we use several trivial examples to show that the two metrics, similarity and GPR, are insufficient and not consistent. Figure 3 illustrates click graphs of different traffic patterns. Publishers and visitors are denoted as $p_{\{1,2,3\}}$, and $v_{\{1,2,3,4\}}$, respectively. Only the case (a) is an example of coalition attacks that the three publishers share the resources. However, the similarity of the group in case (b) is greater than or equal to the value of case (a), and the GPR of the group in case (c) is same as the value of case (a). In other terms, case (b) and case (c) can be respectively misidentified as coalitions according to similarity-based and gpr-based identifications for coalition attacks given in Definitions 2.3 and 2.2.

2.3. **MapReduce computing paradigm.** A key challenge of detecting coalition attacks is the complexity for calculating the metrics for all possible groups among the whole set of publishers. The problem of detecting coalitions is NP-hard in general, and usually involves with processing large set of data. Therefore, in order to apply the proposed metric in practical, we adopt a parallel computing technique called MapReduce.

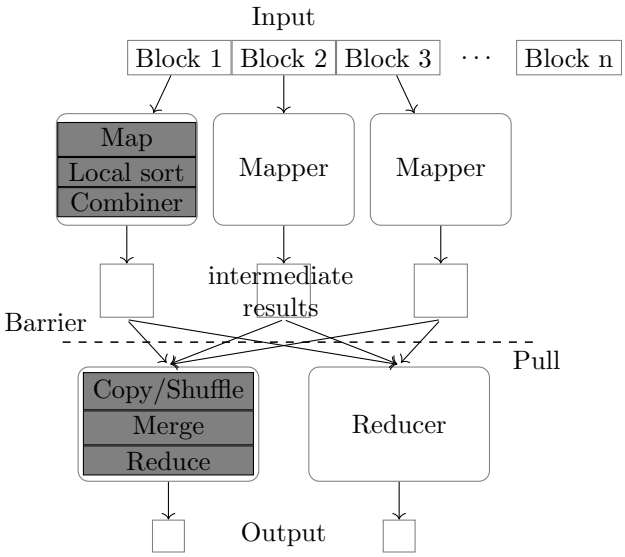


FIGURE 4. The Architecture of Hadoop [14]

MapReduce is a prominent parallel, distributed data processing paradigm that grows rapidly and has gained significant momentum from both industry and academia. Figure 4 illustrates an overview of the Hadoop architecture, which is an open-source Java implementation of MapReduce [5]. In general, each processing job is broken down to as many tasks as input data blocks on multiple clusters. With the MapReduce model, details of parallel execution are hidden and handled by the internal implementation. Users focus only on the task of data processing. A single MapReduce (MR) job simply consists of two primitive stages: Map and Reduce. The Map stage is to map the input data to a list of pairs (key, value), and the Reduce stage calculates results by aggregating the value of keys.

3. **Detection of coalition attacks.**

3.1. **The theoretical basis for the detection technique.** In Definitions 2.2 and 2.3, we have formally defined metrics that can be used to identify the coalition attacks. However, the detection technique to find all of them is not straightforward. For a universal group with N publishers in total, all its possible sub-groups would be 2^N. It is impractical to calculate the proposed metric for all possible sub groups and identify those that satisfy the given definitions. Such brute-force searching scheme is especially impossible for a hybrid approach as the computational complexity would increase exponentially when more metrics are integrated. On the other hand, a priority-style detection scheme would search the space inductively and prune impossible candidate as early as possible, which as a result may reduce the complexity dramatically. Furthermore, the combination of several metrics helps to further prune a significant number of candidates on the fly, and further reduce the computation time.

A well-known property derived from association rule mining is the following definition of anti-monotone.

Definition 3.1 (anti-monotone property[13])**.** Let $P : \mathbb{X} \rightarrow \{\text{true, false}\}$ be a predicate over the domain $\mathbb{X}$, P is said to be anti-monotone iff:

$$\forall(g, g' \in \mathbb{X} \mid g' \subseteq g : P(g) \implies P(g')).$$

We can consider the identification of coalition attacks as a predicate P. Let D denote the set of all publishers and visitors. The domain $\mathbb{X}$ corresponds to the powerset of D in our contex. However, not every metric that is abstracted from the domain knowledge has the anti-monotone property. In other words, to fit the metrics into the inductive detection style, we should derive anti-monotonic predicates based on the original metrics.

To obtain the anti-monotone predicate w.r.t. the similarity-based identification (see Definition 2.3), we use notations of similar graph, and clique. Let G_{sim} denote a similarity graph associated with the underlying traffic. Each vertex in G_{sim} corresponds to a publisher, and edges represent the similarity of visitors between two publishers is greater than or equal to the threshold θ_{sim}. Moreover, a clique is a subset of vertices of an undirected graph G such that its induced subgraph is complete. With accordance to the similarity-based identification, the detection of coalition attacks is abstracted as finding cliques in the associated similarity graph. Formally, we can prove the following lemma according to Definition 2.3.

Lemma 3.2. *Given the similarity graph G_{sim}, a group g is a coalition group iff g is a clique associated with G_{sim}.*

Consequently, we can derive a predicate using the notation of clique. Proposition 1 indicates that the predicate is anti-monotone.

Proposition 1 (similar clique property)**.** $\forall(g' \mid g' \subseteq g : g$ *is a clique associated with* $G_{sim} \implies g'$ *is a clique associated with* $G_{sim})$

With respect to the gpr-based identification given in Definition 2.2, the GPR property cannot be transferred to an anti-monotone predicate directly. We instead use the anti-monotone predicate according to an alternative concept called GPR-CORE [12]. Let G_{click} denote a click graph induced by the traffic data. The graph in Figure 3 is an example of such click graphs. Each vertex in G_{click} corresponds to either a publisher or a visitor. An edge (p, v) in G_{click} means that the visitor v visits the publisher p. Given the definition of GPR-CORE below, we can also have the anti-monotone property of GPR-CORE in Proposition 2.

Definition 3.3. A click graph g is called a GPR-CORE if g is a GPR-GROUP and every connected subgraph of g is also a GPR-GROUP.

Proposition 2 (gpr core property)**.** $\forall(g' \mid g' \subseteq g : g$ *is a gpr-core group* $\implies g'$ *is a gpr-core group* $)$

Consequently, the following lemma articulates the gpr-based identification w.r.t. the anti-monotone property GPR-CORE.

Lemma 3.4. *a group g is a coalition group if g satisfies the following conditions:*

- $\exists(g', g'' \mid g' \cup g'' = g : g'$ *is GPR-CORE and* g'' *is GPR-CORE),*
- *the GPR value of g is greater than or equal to θ_{gpr},*
- *the graph G_{click} induced by g is connected.*

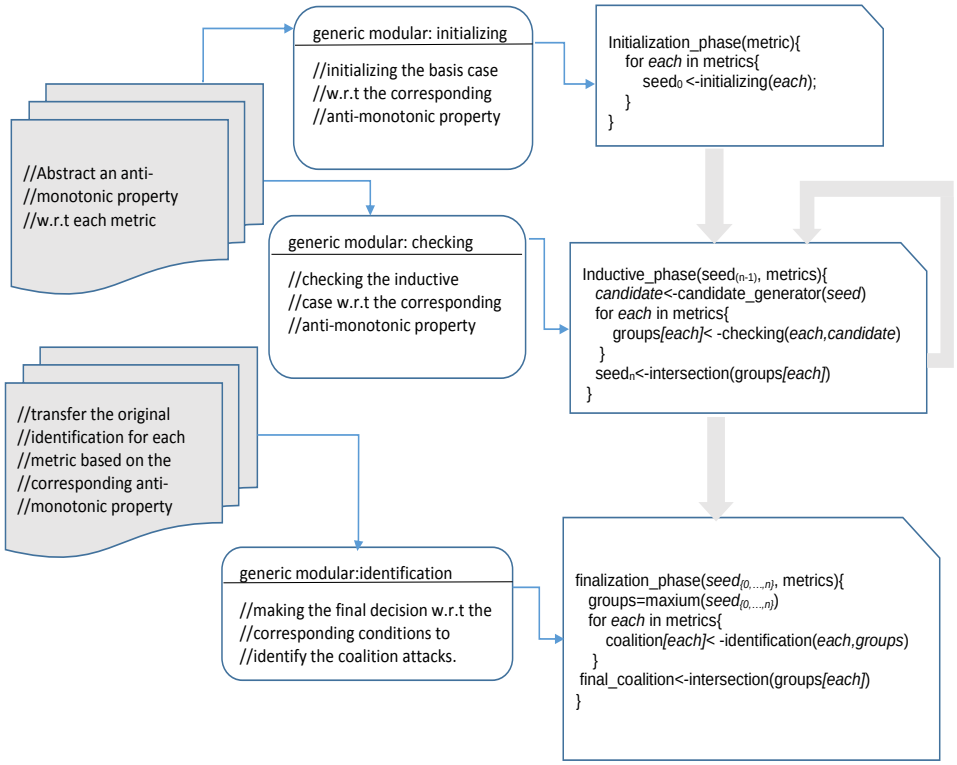


FIGURE 5. Framework of the Hybrid Detection System

3.2. **The framework of the hybrid detection system.** Figure 5 illustrates the framework of the hybrid detection system. There are three major stages of the system: initialization phase, inductive phase, and finalization phase. With such a framework, we adopt an aprior-style to identify the coalition attacks. Moreover, we define generic modules that can be instanced with different metrics. Such a framework enables the efficiency and extensibility of the hybrid approach. Following Figure 5 and the discussions in Section 3.1, implementation of the system is straightforward.

For the similarity-based identification, the *initializing* module responses to generate the similarity graph G_{sim}. In the property *checking* module, we check the predicate $P(g) \equiv \langle g$ is a clique associated with $G_{sim}\rangle$. As discussed previously, the anti-monotone property (see Proposition 1) of the predicate ensures the soundness of the inductive detection mechanism. Algorithms 1 and 2 articulate the pseudo code for the similarity-based *initializing* and property *checking* modules. In Line 2 of Algorithm 1, we call a function `PAIR_INTERSECT_UNION` to calculate the bag union and intersection of visitors for a pair of publishers according to the definitions. In Line 2 of Algorithm 2, a function `COUNT_GROUP_EDGES` is used to count the number of edges in G_{sim} for potential candidate subgroups. To improve the efficiency of the algorithm, these two functions are implemented with the MapReduce paradigm,

Algorithm 1 initializing: similarity-based

```
1: procedure PAIR_SIMILARITY(Data)
2:     pairs ← the list of all pairs of Publishers
3:     pairs_value ← PAIR_INTERSECT_UNION(pairs, Data)
4:     G_sim ← initialize the storage of the similar graph
5:     for each in pairs do
6:         (intersection, union) ← pairs_value[each]
7:         similarity ← intersection / union
8:         if similarity ≥ θ_sim then
9:             Adding each as an edge in G_sim
10:        end if
11:    end for
12:    return G_sim
13: end procedure
```

which will be discussed in the next section. Moreover, the final *identification* module for the similarity based metric is trivial. As given in Definition 3.2, identified similar cliques are identical to coalition groups.

Algorithm 2 checking: similarity-based

```
1: procedure SIMILAR GROUP(candidate,G_sim)
2:     group_edges ← COUNT_GROUP_EDGES(candidate,G_sim)
3:     clique_groups ← initialize the storage of the clique groups
4:     for each in candidate do
5:         complete_edge ← len(each) * (len(each)-1)
6:         if complete_edge ← group_edges [each] then
7:             Adding each to clique_groups
8:         end if
9:     end for
10: end procedure
```

For the gpr-based identification, we calculate the gpr value for single publisher and visitor in the *initializing* module, and check the gpr-core property in the *checking* module. Moreover, Lemma 3.4 is used to identify tasks of the *identification* module in the gpr-based case. The corresponding algorithm is given in Algorithms 3, 4, and 5, respectively. The function `CALCULATE_RESOURCE` that is called in Lines 5 and 14 of Algorithm 3, Line 6 of Algorithm 4, and Line 7 of Algorithm 5 calculates the resource cost of the corresponding group (i.e., $R(g)$ in Definition 1). In Algorithm 3, we initially obtain the boundary value for individual (i.e., either a publish or a visitor) gain by calling the function `INDIVIDUAL_GAIN_BOUNDARY`. The individual gbr value is always 0, and the induced click graph can be considered as connected. Furthermore, the function `GROUP_GBC_VALUES` is called in Line 2 of Algorithm 4, and Line 3 of Algorithm 5 to calculate the gain, gain boundary, and connectivity for all candidate groups. Both functions `INDIVIDUAL_GAIN_BOUNDARY` and `GROUP_GBC_VALUES` are implemented using the map-reduce paradigm. Moreover, we implement another map reduce function `MERGE_GPR_CORE` to merge the GPR-CORE groups to potential GPR groups according to the first item in Lemma 3.4.

Algorithm 3 initializing: gpr-based

```
1: procedure INDIVIDUAL_GPR(Data)
2:     (P_gainb, V_gainb) ← INDIVIDUAL_GAIN_BOUNDARY(Data)
3:     GPR_1 ← initialize the storage for individual gpr values
4:     for p in P_gainb do
5:         resource ← CALCULATE_RESOURCE([p],[])
6:         gprb ← P_gainb[p] / resource
7:         gpr ← 0
8:         connectivity ← true
9:         if gprb ≥ θ_gpr then
10:            Adding (p,gpr,gprb,connectivity) to the GPR_1
11:        end if
12:    end for
13:    for v in V_gainb do
14:        resource ← CALCULATE_RESOURCE([],[v])
15:        gprb ← V_gainb[v] / resource
16:        gpr ← 0
17:        connectivity ← true
18:        if gprb ≥ θ_gpr then
19:            Adding (v,gpr,gprb,connectivity) to the GPR_1
20:        end if
21:    end for
22:    return GPR_1
23: end procedure
```

Algorithm 4 checking: gpr-based

```
1: procedure GPR-CORE(Data,candidate,GPR_(n−1))
2:     gbc_values ← GROUP_GBC_VALUES(Data,candidate)
3:     GPR_n ← initial the storage for the group gain values of size n
4:     for g in candidate do
5:         (publishers,visitors) ← g
6:         resource ← CALCULATE_RESOURCE(publishers,visitors)
7:         (gpr,gprb) ← gbc_values[g] / resource
8:         Adding (g, gpr, gprb) to the GPR_n
9:     end for
10:    subset ← COUNT_VALID_SUBSET(candidate,GPR_(n−1),GPR_n)
11:    gprcore ← initialize the storage of the GPR-CORE groups
12:    for g in candidate do
13:        if subset[g] == n & GPR_n ≥ θ_gpr then
14:            Adding g to the gprcore
15:        end if
16:    end for
17: end procedure
```

3.3. **The map and reduce functions for the detection techniques.** The map reduce paradigm is discussed in Section 2. The input data is mapped into

Algorithm 5 identification: gpr-based

```
1: procedure FIND_GPR_GROUPS(gprcore, GPR_{1...n})
2:     merged_group ← MERGE_GPR_CORE(gprcore)
3:     gbc_values ← GROUP_GBC_VALUES(merged_group)
4:     gpr_groups ← initialize the storage for the gpr groups
5:     for g in merged_group do
6:         (publishers,visitors) ← g
7:         resource ← CALCULATE_RESOURCE(publishers,visitors)
8:         (gain,gain_b,connectivity) ← gbc_values[g]
9:         gbr ← gain / resource
10:        if gbr ≥ θ_gpr & connectivity is true then
11:            Adding g to gpr_group
12:        end if
13:    end for
14: end procedure
```

a collection of pairs (*key*,*values*), and then the result is calculated by aggregating the *values* w.r.t. each *key*. In order to allow the parallel computation on different clusters, the *reduce* operation on the mapped dataset should be commutative and associative. In our context, different metrics for a large collection of groups have to be calculated. All groups form a lattice data structure, which allows the inductive search strategy in the algorithm. With the map reduce paradigm, a set of groups is mapped to a set of partial ordered *keys*, while *values* are emitted with respect to different metrics. All functions for key encoder, decoder and comparison are given in Appendix A. With the inductive style, we generate new sets of keys in each iteration based on the potential candidates from the previous iteration. To reduce the computation time, we also implement the inductive candidate generator with the map-reduce paradigm (see Appendix A for the map and reduce functions). Furthermore, in Algorithm 6, we articulate the pseudo code for function `pair_intersection_union` and `count_group_edges`. functions `individual_gain_boundary`, `group_gbc_values`, `count_valid_subset`, and `merge_gpr_core` are given in Algorithms 7 and 8. As mentioned in the last section, map-reduce computation is adopted in Lines 3, 21 of Algorithm 6, Lines 3, 23 of Algorithm 7, and Lines 10, 24 of Algorithm 8. In particular, we use Disco [4] to conduct the computation and the implementation of those mapper functions are also given in the Appendix.

4. **Evaluation of the detection system.** To evaluate the proposed detection algorithm, we compare the performance of our algorithm with those that only use single metric. The proposed hybrid framework provides a convenient way to extend the system with different metrics. In other term, instead of re-implementing those algorithms separately, we can simply modify the list of metrics in Figure 5 to obtain and compare different algorithms.

We retrieve the motivative examples introduced in Section 2. Both values of θ_{sim} and θ_{gpr} have critical effects on the detection rate and error rate of the algorithm. The detection rate becomes low if the threshold is too high. On the other hand, the precision rate can also decrease when the threshold is too low. In Figure 6, we illustrate effects of θ_{sim} and θ_{gpr} on the traffic that corresponds to the examples

Algorithm 6 functions associated with the similarity metric

```
1: function PAIR_INTERSECTION_UNION(Data,pair)
2:     pairs_value ← {}
3:     for (key,values) ∈ reduce(mapper_similarity_pairs(pairs,Data)) do
4:         \\ "key" is the key for each pairs
5:         \\ "values" are counts of clicks that are associated the pair, respectively
6:         pairs ←KEY_DECODER(key)[0]
7:         if values[0] > 0 & values[1]>0 then
8:             intersection ← min(values[0] ,values[1] )
9:             union ← max(values[0] ,values[1] )
10:        else if values[0] > 0 & values[1]==0 then
11:            intersection ← 0
12:            union ← values[0]
13:        else values[0]==0 & values[1]>0
14:            intersection ← 0
15:            union ← values[1]
16:        end if
17:        pairs_value[key]← [intersection,union]
18:    end for
19:    return pairs_value
20: end function
21: function COUNT_GROUP_EDGES(G_sim,candidate)
22:    group_edges← {}
23:    for (key,value) ∈ reducer(map_similarity_groups(candidate,G_sim)) do
24:        \\ "key" is the encoded key of each group in candidate
25:        \\ "value" is the count of edges in G_sim that are associated with all
   elements in the group
26:        group=key.decoder(key)
27:        group_edges[ group] ← value
28:    end for
29:    return group_edges
30: end function
```

given in Figure 3. The detection rate is 1 when the θ_{sim} less than 0.33, while the detection rate becomes 0 when the value greater than 0.33. The detection rate is 0 with θ_{gpr} greater than 1. With a relative low threshold θ_{gpr}, we may also increase the opportunity of positive false alarms. For example, the precision rate could be as low as 0.3 when θ_{gpr} is less than 0.84. On the other hand, with the hybrid approach, we can obtain a high detection rate while still keep a reasonable precision rate. Figure 7 shows the effect of the two parameters θ_{sim} and θ_{gpr} on the detection rate and the precision of the hybrid approach. In particular, we adjust the value of one parameter with a fix value for the other. With accordance to the dropping point of detection rate in Figure 6, we set θ_{sim} to be 0.32 and θ_{sim} to be 1, respectively. We can see that the precision in Figure 7(b) is always much better than the precision in Figure 6(b), and the average precision in Figure 7(a) is also better than the case of 6(a).

We also applied the proposed technique to practical data. The raw data are daily log including impressions, clicks, and conversions from real traffic. Each record in

Algorithm 7 functions associated with the gpr metric

```
1: function INDIVIDUAL_GAIN_BOUNDARY(Data)
2:     P_gainb ← {}
3:     V_gainb ← {}
4:     for key,value) ∈ reduce(map_vertices(Data) do
5:         \\ "key" is the encoded key of each publisher or visitor
6:         \\ "value" is the sum cost that is associated with the publisher or visitor
7:         (p,v)=key.decoder(key)
8:         if v==“” then
9:             P_gain[p] ← value[1]
10:        elsep==“”
11:            V_gain[p] ← value[1]
12:        end if
13:    end for
14:    return (P_gainb, V_gainb)
15: end function
16: function GROUP_GBC_VALUES(Data,candidate,size)
17:    gbc_values ← {}
18:    for (key,values) ∈ reduce(map_gbc_values(Data,candidate)) do
19:        \\ "key" is the encoded key for each group in candidate
20:        \\ "values" correspond to the gain, gain boundary, and the number of
   connected edges that are associated with the group
21:        gain← values[0]
22:        gain_boundary ← values[1]
23:        if values[2]>0 then
24:            connectivity = true
25:        else
26:            connectivity = false
27:        end if
28:        gbc_values[key] ←(gain,gain_boundary, connectivity)
29:    end for
30:    return gbc_values
31: end function
32: function COUNT_VALID_SUBSET(candidate)
33:    subset ← {}
34:    for (key,value) ∈ reduce(map(Data,candidate)) do
35:        \\ "key" is the encoded key for each group in candidate
36:        \\ "value" count the number of valid subgroup of the group (i.e. satisfies
   the properties of GRP-Core)
37:        subset[key] ← value
38:    end for
39:    return subset
40: end function
```

the log is related to more than 50 features. To prepare the dataset of our algorithm, we index the publisher and visitor, associate the impression and clicks, and then extract the cost for each click. To further evaluate the effectiveness, we use third

Algorithm 8 functions associated with the gpr metric (cont.)

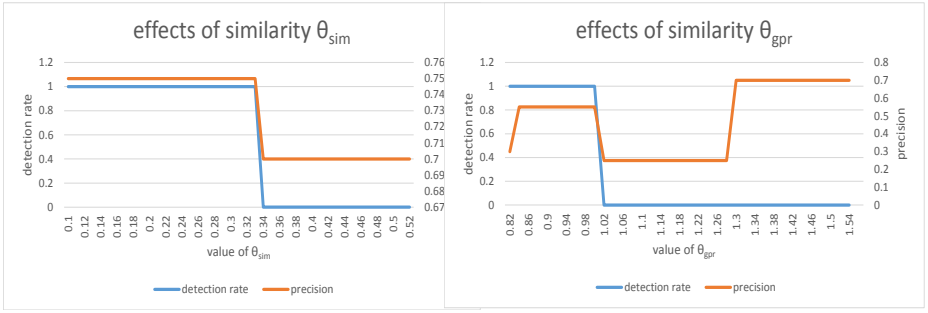
```
function MERGE_GPR_CORE(Data,gprcore)
    i ← 0
    merged_i ← gprcore
    prev ← merged_i
    new ← []
    while prev!=new do
        pairs ← combination(new,2)
        i ← i+1
        separated ← {}
        for (key,value)∈ REDUCE(MAP_MERGE_GPRCORE(pairs)) do
            \\ "key" is the encoded key for the merged group that associated with each pair.
            \\ "value" is the number of overlapped vertices for the pair
            if value>0 then
                Adding key to merged_i
            else
                (merged_key, component_key) ← key.split(":")
                separated[merged_key] ← component_key
            end if
        end for
        unknown_keys ← separated.keys() - merged_i
        for key in unknown_keys do
            candidate[key] ← separated[key]
        end for
        for (key,values) ∈ REDUCE(MAP_GBC_VALUES(Data,Candidate)) do
            if values[2] > 0 then
                Adding key to merged_i
            end if
        end for
        prev ← merged_{i−1}
        new ← merged_i
    end while

end function
```

party service to label the click either as 'normal' or 'coalition attack'. Table 1 shows results for three different datasets.

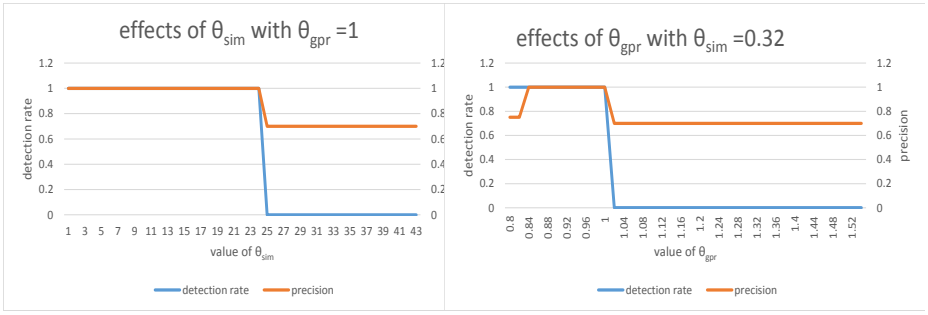
We can see that the algorithm has an average detection rate of 97.67% and precision 96.44%. In experiments, each dataset includes the traffic of 7 days with $300 \sim 500$ publishers. Since we adopt the map-reduce paradigm, our detection technique allows parallel computing. Although the complexity of the algorithm is determined by the number of publishers in the dataset, our detection system inherently has no restriction on the size of the problem. Therefore, with increasing number of clusters, we can further reduce the processing time and handle even larger size of data.

5. **Conclusion and future work.** In this paper, we proposed a detection technique for coalition attacks in online advertising. A hybrid framework that can detect



(a) effects of θ_{sim} (b) effects of θ_{gpr}

FIGURE 6. Independent Parameter Adjustment



(a) effects of θ_{gpr} (b) effects of θ_{sim}

FIGURE 7. Dependent Parameter Adjustment

TABLE 1. Results obtained from practical data

#publishers	detection rate	precision
340	96.12%	95.03%
414	97.12%	99.35%
407	99.18%	94.94%

coalition attacks based on multiple metrics is developed. We articulate the theoretical properties that each metric should satisfy in the framework. In particular, we adopt similarity and gpr as two metrics in the system. The performance of the technique with different datasets is evaluated.

As future work, more datasets will be tested. Since there is no benchmark data sets for coalition frauds in online advertising, obtaining the data for evaluation is usually difficult and expensive. Besides real-world data sets and the third party service, simulation technique can be used to generate synthetic date for both normal and fraudulent traffic. Extensive empirical study on different traffic patterns can be studied. On the other side, to improve the performance of the detection technique, more metrics can be explored to identify coalition attacks [6, 10]. With the hybrid framework, additional metrics can be integrated in the system efficiently. The

ultimate goal of the work is to provide an reliable, extensible, and scalable on-line detection system for the dramatically developing online advertising industry.

Acknowledgments. The project was supported by the Mathematics of Information Technology and Complex Systems of Canada (MITACS) and EQ Works (`http://www.eqworks.com/`).

REFERENCES

[1] *Zenithoptimedia*, Available from `http://techcrunch.com/2014/04/07/`.
[2] *Doubleclick by Google*, Available from: `http://www.google.ca/doubleclick/`.
[3] *Rightmedia from Yahoo!*, Available from: `https://www.rightmedia.com/`.
[4] *Disco MapReduce*, Available from `http://discoproject.org/`.
[5] *Hadoop MapReduce*, Available from: `https://hadoop.apache.org`.
[6] L. Adamic and E. Adar, Friends and neighbors on the web, Social Networks, **25** (2003), 211–230.
[7] V. Anupam, A. Mayer, K. Nissim and B. Pinkas, On the security of pay-per-click and other web advertising schemes, in *The 8th International Conference on World Wide Web*, **31** (1999), 1091–1100.
[8] M. S. Charikar, Similarity estimation techniques from rounding algorithms, in *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, (2002), 380–388.
[9] N. Daswani, C. Mysen, V. Rao, S. Weis, K. Gharachorloo and S. Ghosemajumde, Online advertising fraud, in *Crimeware: Understanding New Attacks and Defenses*, Addison-Wesley Professional, Reading, 2008.
[10] M. A. Hasan, A survey of link prediction in social networks, *Social Network Data Analytics*, (2011), 243–275.
[11] B. Kitts, J. Y. Zhang, A. Roux and R. Mills, Click fraud detection with bot signatures, in *Proceedings of ISI 2013, Seattle, Washington, USA*, (2013), 146–150.
[12] C. Kim, H. Miao and K. Shim, CATCH: A detecting algorithm for coalition attacks of hit inflation in internet advertising, *Information Systems*, **36** (2011), 1105–1123.
[13] C. K. S. Leung, Anti-monotone constraints, in *Encyclopedia of Database Systems* (eds. Ling Liu and M. Tamer Özsu), Springer, (2009), 98–98.
[14] K. Lee, H. Choi and B. Moon, Parallel data processing with MapReduce: a survey, in *The ACM Special Interest Group on Management of Data Record*, **40** (2011), 11–20.
[15] A. Metwally, D. Agrawal and A. EI Abbadi, Duplicate detection in click streams, in *International World Wide Web Conference*, (2005), 12–21.
[16] A. Metwally, D. Agrawal and A. EI Abbadi, Using association rules for fraud detection in web advertising networks, in *Proceedings of the 31st international conference on very large data bases*, (2005), 169–180.
[17] A. Metwally, D. Agrawal and A. EI Abbadi, Detectives: detecting coalition hit inflation attacks in advertising networks streams, in *Proceedings of the 16th International Conference on World Wide Web*, (2007), 241–250.
[18] A. Metwally, D. Agrawal, A. EI Abbadi and Q. Zheng, On hit inflation techniques and detection in streams of web advertising networks, in *Proceedings of the 27th International Conference on Distributed Computing Systems*, (2007), 52–52.
[19] C. Phua, E. Y. Cheu, G. E. Yap, K. Sim and M. N. Nguyen, Feature engineering for click fraud detection, in *ACML Workshop on Fraud Detection in Mobile Advertising*, 2012.
[20] Y. Peng, L. Zhang, J. M. Chang and Y. Guan, An effective method for combating malicious scripts clickbots, in *Computer Security, the Series Lecture Notes in Computer Science*, **5789** (2009), 523–538.
[21] B. K. Perera, *A Class Imbalance Learning Approach to Fraud Detection in Online Advertising*, M.Sc. thesis, Masdar Institute of Science and Technology, 2013.
[22] K. Springborn and P. Barford, Impression fraud in online advertising via pay-per-view networks, in *Proceedings of the 22nd USENIX Security Symposium*, (2013), 211–226.
[23] F. Soldo and A. Metwally, Traffic anomaly detection based on the IP size distribution, in *Proceedings of the INFOCOM International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, (2012), 2005–2013.
[24] C. Walgampaya and M. Kantardzic, Cracking the smart clickbot, in *Proceedings of the 13th IEEE International Symposium on Web Systems Evolution*, (2011), 125–134.

[25] C. Walgampaya and M. Kantardzic and R. Yampolskiy, Evidence fusion for real time click fraud detection and prevention, in *Intelligent Automation and Systems Engineering, Lecture Notes in Electrical Engineering*, Springer Science+Business Media, **103** (2011), 1–14.
[26] Q. Zhang and W. Feng, Detecting coalition frauds in online-advertising, in *Mathematical and Computational Approaches in Advancing Modern Science and Engineering*, (eds. Jacques Bélair, Ian A. Frigaard, Herb Kunze, Roman Makarov, Roderick Melnik and Raymond J. Spiteri) Springer, (2016), 595–605.
[27] L. Zhang and Y. Guan, Detecting click fraud in pay-per-click streams of online advertising networks, in *The 28th International Conference on Distributed Computing Systems*, (2008), 77–84.

Appendix A. Implementation.

The following code are written with PYTHON.

A.1. Key handler.

In this subsection, we list functions for encoding, decoding, and comparing the keys.

```
def key_decoder(publishers,visitors):
   sorted_p=sorted(publishers)
   sorted_v=sorted(visitors)
   key_p=‘‘_’’.join(each for each in sorted_p)
   key_v=‘‘_’’.join(each for each in sorted_v)
   key=key_p+‘‘!!’’+key_v
   return key

def key_decoder(key):
   (key_p,key_v)=key.split(‘‘!!’’)
   publishers=key_p.split(‘‘_’’) if not key_p ==‘‘’’ else []
   visitors=key_v.split(‘‘_’’) if not key_v ==‘‘’’ else []
   return (publishers,visitors)

def key_cmp(key1,key2):
   (p1,v1)=key_decoder(key1)
   (p2,v2)=key_decoder(key2)
   p_in=set(p1).issubset(set(p2))
   v_in=set(v1).issubset(set(v2))
   return p_in and v_in
```

A.2. General mapper reduce functions.

We generalize two mapper and reducer functions that can be reused by other functions.

```
def inductive_generator_mapper(line,params):
   candidate=line.split(‘‘,’’)[0]
   P=params[0]
   V=params[1]
   (publishers,visitors)=candidate.split(‘‘!!’’)
   for p in P:
      if not p in publishers:
         key=key_encoder(publishers+[p],visitors)
         yield (key+‘‘:’’+each,[1])
   for v in V:
      if not v in visitors:
```

```
        key=key_encoder(publishers,visitors+[V])
        yield (key+‘‘:’’+each,[1])

def reducer(iter, params):
   data = {}
   first = True
   prev = None
   for key, value in sorted(iter):
      if key not in data:
         if not first:
            yield prev, data[prev]
            data = {}
         data[key] = value
      else:
         for k, v in enumerate(value):
            if type(v)==set:
               data[key][k]=data[key][k].union(v)
            else:
               data[key][k]+= v
      first = False
      prev = key
   for key, value in data.iteritems():
      yield key, value
```

A.3. **Other mapper reduce functions.** All other mapper reduce functions are given in the rest of this section.

```
def mapper_similarity_pairs(line,params):
   groups=params
   (head,tail,weight,label)=line.strip().split(‘‘,’’)
   for each in groups:
      (p1,p2)=each.split(‘‘_’’)
      key=each+‘‘!!’’+tail
      if head==p1:
         yield (key,[1,0])
      elif head==p2:
         yield (key,[0,1])
      else:
         pass

def mapper_similarity_groups(line,params):
   #tag2group=params[0]
   group_keys=params[0]
   V_key=params[1]
   threshold=params[2]
   (p1,p2,similarity)=line.split(‘‘,’’)
   if float(similarity)<threshold:
```

```
      pass
   else:
      for key in group_keys:
         (p_key,v_key)=key.split(‘‘!!’’)
         new_key=p_key+‘‘!!’’+V_key
         pubs=p_key.split(‘‘_’’)
         if p1 in pubs and p2 in pubs:
            yield (new_key,[1])

def mapper_vertices(line,param):
   (head,tail,weight,label)=line.split(‘‘,’’)
   p_key=head+‘‘!!’’
   v_key=‘‘!!’’+tail
   yield (p_key,[1,float(weight)])
   yield (v_key,[1,float(weight)])

def mapper_gbc_values(line,params):
   groups=params
   (head,tail,weight,label)=line.split(‘‘,’’)
   #print groups
   for key in groups:
      and_edge=0
      or_edge=0
      (key_p,key_v)=key.split(‘‘!!’’)
      publishers=key_p.split(‘‘_’’) if not key_p==‘‘’’ else []
      visitors=key_v.split(‘‘_’’) if not key_v==‘‘’’ else []
      if head in publishers and tail in visitors:
         and_edge=float(weight)
      if head in publishers or tail in visitors:
         or_edge=float(weight)
      #yield (key,[and_edge,or_edge])
      components=groups[key]
      connectivity=0
      for each in components:
         (p_key,v_key)=each.split(‘‘!!’’)
         heads=p_key.split(‘‘_’’)
         tails=v_key.split(‘‘_’’)
         if head in heads and tail in tails:
            connectivity=1
            break
      yield (key,[and_edge,or_edge,connectivity])
```

Received February 2016; revised September 2016.

E-mail address: trina.zhang.q@gmail.com
E-mail address: wfeng@trentu.ca