2013

Mining Indirect Antagonistic Communities from Social Interactions

Kuan ZHANG

David LO
Singapore Management University, davidlo@smu.edu.sg

Ee Peng LIM
Singapore Management University, eplim@smu.edu.sg

Philips Kokoh Prasetyo
Singapore Management University, pprasetyo@smu.edu.sg

DOI: https://doi.org/10.1007/s10115-012-0519-4

Citation

ZHANG, Kuan; LO, David; LIM, Ee Peng; and Prasetyo, Philips Kokoh. Mining Indirect Antagonistic Communities from Social Interactions. (2013). *Knowledge and Information Systems*. 35, (3), 553-583. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/1559

Mining Indirect Antagonistic Communities from Social Interactions

Article *in* Knowledge and Information Systems · June 2012
DOI: 10.1007/s10115-012-0519-4

David Lo
Singapore Management University
311 PUBLICATIONS **4,817** CITATIONS
SEE PROFILE

Ee-Peng Lim
Singapore Management University
446 PUBLICATIONS **9,021** CITATIONS
SEE PROFILE

Philips Kokoh Prasetyo
Singapore Management University
21 PUBLICATIONS **115** CITATIONS
SEE PROFILE

Accepted for publication in Knowledge and Information Systems

Mining Indirect Antagonistic Communities from Social Interactions

Kuan Zhang, David Lo, Ee-Peng Lim, and Philips Kokoh Prasetyo

School of Information Systems, Singapore Management University, Singapore

Abstract. Antagonistic communities refer to groups of people with opposite tastes, opinions, factions within a community. Given a set of interactions among people in a community, we develop a novel pattern mining approach to mine a set of antagonistic communities. In particular, based on a set of user specified thresholds, we extract a set of pairs of communities that behave in opposite ways with one another. We focus on extracting a compact lossless representation based on the concept of closed patterns to prevent exploding the number of mined antagonistic communities. We also present a variation of the algorithm using a divide and conquer strategy to handle large datasets when main memory is inadequate. The scalability of our approach is tested on synthetic datasets of various sizes mined using various parameters. Case studies on Amazon, Epinions, and Slashdot datasets further show the efficiency and the utility of our approach in extracting antagonistic communities from social interactions.

Keywords: Antagonistic group; Frequent pattern mining; Closed pattern; Social network mining

1. Introduction

In social interactions, people tend to share their opinions, either through sentiments or ratings. Current technologies enable us to easily share our opinions on specific objects such as articles, movies, books, people, and softwares. These opinions are important for users to evaluate a particular object, and how the object influences and affects people. It is common that groups of people or communities are formed based on similarity in opinions. For some pairs of groups, we observe social interactions in which two groups of the same pair consistently differ in opinions. We call these two groups of people holding opposite opinions on some objects as *indirect antagonistic communities*. Indirect antagonistic

Received Dec 24, 2011
Revised Mar 31, 2012
Accepted Apr 07, 2012

communities are common in various social settings including commerce, lifestyle, politics, religions, sports, ideology, etc.

Detecting indirect antagonistic communities is one of the first steps in understanding the dynamics of social interactions. Early detection of antagonistic communities could help to avert unwanted tensions among opposing communities. In product evaluation, information about antagonistic communities and their differing opinions could also be used for better product design, market segmentation, recommendation, etc.

Mining antagonistic communities is relatively new research problem. Several studies on finding communities in social network [6, 14, 13] focus on finding cohesive or non-antagonistic communities. While cohesive communities are important, understanding the relations among these communities such as antagonistic relations is also equally important. We enrich past studies on community finding by discovering not cohesive communities but ones with opposing sub-communities. We believe these two sources of information could give more light to the social interactions among users in Web 2.0. In addition, opposing communities and their nature have been studied in the sociology domain [33, 9, 8, 20, 15, 10].

In this paper[1], we design a novel pattern mining algorithm to discover indirect antagonistic community from social interactions. Each antagonistic community is represented as a pattern. The algorithm explores opinions on a set of objects, and finds all antagonistic communities. To increase the efficiency, the algorithm prunes away all candidate communities which do not have enough frequency/support. As a frequent antagonistic pattern would have large number of sub-patterns, we only select closed patterns to output.

We experiment our solution on synthetic datasets of various sizes under various mining parameters to evaluate its scalability. We also conduct case studies on Amazon, Epinions, and Slashdot datasets to show the efficiency and the utility of our approach in mining antagonistic communities.

The contributions of this work are as follows:

1. We propose a new problem of mining indirect antagonistic communities. Mined indirect antagonistic communities could potentially be used to shed better light on social interactions, prevent unwanted tensions in the communities, improve recommendations and marketing strategies, etc.
2. We propose a new algorithm to mine indirect antagonistic communities that is shown to be scalable.
3. We extract indirect antagonistic communities from real datasets demonstrating antagonistic behaviors in real rating datasets.

This paper is organized as follows. We present related work in Section 2. In Section 3, we formalize the concepts, properties, and problem of indirect antagonistic community mining. Our algorithm to discover indirect antagonistic community, *Clagmine*, is described in Section 4. We discuss the experiment and performance result in Section 5. Finally, we conclude our work in Section 6.

[1] This paper is an extension of our conference paper [40] with additional descriptions, case studies, and analysis.

2. Related Work

In this section, we highlight related work on community finding, signed social networks, homophily and inter-group antagonism, and frequent pattern mining.

2.1. Community Finding

Community finding is one of key problems in social network analysis and it has been extensively studied [16, 36, 30, 21, 34, 38, 6, 14, 13]. Previous studies can be divided into three categories: unsigned, signed, and heterogeneous networks. We list and highlight some of these studies below. Our list is by no means complete. Different from these studies, our goal is not to find homogenous communities rather pairs of communities that exhibit antagonistic behaviors.

Unsigned networks. Girvan et al. and Newman worked on algorithms to mine communities from undirected networks [16, 30]. Their work is based on a basic principle that the links connecting to different communities must be few and the shortest paths between any two nodes from the two different communities must pass through such links. Thus, these links carry high "betweenness". Another work by Leicht et al. mined communities in directed graphs [21]. Previous works on unsigned networks only consider link density when dividing the networks into communities. The link density within communities should be as dense as possible and the link density between communities should be as sparse as possible.

Signed networks. For signed networks, we need to take the signs of links into consideration when determining communities. The basic criteria is that for positive links, high link density is desired, but for negative links, the density should be sparse within a community. A two-step method to mine communities from signed network is introduced by Yang et al. [38]. Their algorithm is based on the principle that if an agent starts from any node and transits after a few steps, the probability that it remains in the same community is greater than that of reaching a different community. Their algorithm is more biased on sign of links, less on the density. Traag et al. proposed another solution on mining communities from signed networks [34], which was based on the work by Newman to find communities in unsigned network [30]. Their work only considers communities' internal links, and the links between communities are not taken into consideration.

Heterogeneous networks. Cai et al. proposed an approach to mine communities from multi-relational social networks (i.e., multiple networks on the same set of nodes) [6]. They first formed the target relationship network (i.e., partial information of the hidden relationship network which is inferred from the multi-relational networks) from the labeled nodes, where nodes in the same community have the same label. Next, they combined the original heterogeneous networks to approximate the target relationship networks.

2.2. Signed Social Networks

There have been a few studies that analyze and mine signed social networks, e.g., [12, 22, 5].

In [12], Easley and Kleinberg described some basic properties of signed networks. The authors introduced balanced triangles from social psychology to capture the stable structures, each consisting of three nodes and their links. If every arbitrary three nodes in a signed network forms a balanced triangle, the network is said to be balanced. A well-known balance theorem [7, 18] holds for the balanced network. This theorem says that if a signed complete network is balanced, either all the nodes have positive links with each other, or the nodes can be divided into two camps, within each camp the nodes are friends to each other and across the camps nodes are enemy to each other. Two extensions of balance theorem have also been studied in [12]. The first extension addresses balance structure of non-complete networks. The second extension is about approximately balanced networks with only most of the triangles balanced. Different from the existing studies on balanced network, in this study we do not decide if a network could be divided into two camps, rather we want to extract the set of (potentially many) antagonistic communities from signed networks.

Leskovec et al. predicted the polarity of known links [22]. They proposed a logistic regression classifier for the prediction task using two classes of features. The first class of features is based on indegree, outdegree and their combinations – with the signs of the links taken into consideration. The second class of features is based on "triads" that involve the target link. In their work, they showed the importance of negative links in predicting positive links.

Bonachich and Lloyd proposed to use eigenvector to measure the centrality or status of each node in a signed social network [5]. The basic principles are: If a node is connected positively to a high status node, the node's status increases. If a node is connected positively to a low status node, the node's status decreases. Conversely, if a node is connected negatively to a high status node, the node's status decreases. If a node is connected negatively to a low status node, the node's status increases.

2.3. Homophily and Inter-Group Antagonism

Homophily. Antagonistic communities are also related to the concept of homophily. Members of a pair of antagonistic communities, intuitively share more preferences with those in the same community and share less preferences with others from the opposing community. There have been a number of studies on homophily in social networks, e.g., [28]. In this work, our mined communities not only express similar preferences but also opposing preferences. Antagonistic community captures a kind of homophily behavior in sharing objects the community of users like together, as well as sharing objects they dislike together.

Inter-group antagonism. In sociology, economics, and psychology research communities, the concept of inter-group antagonism has been studied by various works [33, 9, 8, 20, 15, 10]. We extend this interesting research question by providing a computation tool to automatically identify opposing communities from a history of their behaviors. We believe our tool could potentially be used to help sociologists understand the behaviors of communities from the wealth of available user interaction data in Web 2.0.

This paper is an extension of our preliminary study on mining indirect antagonistic community [40] with more comprehensive details on the proposed algorithm, additional experiments on new datasets, and more detailed analysis of the experiment results. In [27], we have also investigated the problem of mining *direct* antagonistic communities from explicit trust networks. Each direct antagonistic community comprises of two sets of people, where people in each set form a strongly connected component with respect to trust links, and people in the opposing set form a bi-clique with respect to distrust links. In this work we focus on mining *indirect* antagonistic communities, where there are two kinds of entities: users and items, and users from opposing sides of an antagonistic community are different based on their views on the items.

2.4. Frequent Pattern Mining

Our algorithm belongs to the family of pattern mining algorithms [3, 31, 24, 35, 11, 37, 29, 17, 19, 23, 25, 26]. There have been a number of pattern mining algorithms including those mining association rules (e.g., [3, 31]), frequent sequences (e.g., [35, 23]), frequent repetitive sequences (e.g., [11]), frequent subgraphs (e.g., [37]), etc. There are many recent studies too, e.g., [24, 29, 17, 19, 23]. The closest to our study is the body of work on association rule mining [3]. Association rule mining employs the concept of support and confidence. It extracts frequent itemsets and relationship between itemsets. On the other hand, we extract two sets of opposing users that share many common interests (or form opinions about a common set of topics, or rate a common set of items) but oppose each other with high likelihood. This problem is different from association rule mining. We show that a similar apriori-like anti-monotonicity property holds but we employ a different algorithm to mine for antagonistic communities. Similar to the work in [31], we do not report all antagonistic communities rather only the *closed* ones.

3. Preliminary

Indirect antagonistic communities are pairs of opposing sets of users derived from their indirect social interactions, particularly through ratings of items such as products, views, events, or even ideas. All rating scores are categorized into three **rating polarity** levels: high, medium, and low rating polarity. For example, in 1-5 rating scale, we categorize rating scores 1-2 as low polarity, 3 as medium polarity, and 4-5 as high polarity. We define some preliminary concepts and the indirect antagonistic community mining problem as follows.

Definition 3.1. (Rating Database) Consider a set of users U and a set of items I. A database of ratings consists of a set of mappings of item identifiers to a set of pairs, where each pair consists of user identifier and rating score. There are three types of rating scores: high (hi), medium (mid), and low (lo). The rating database could be formally represented as:

$$DB_R = \{it_{id} \mapsto \{(us_{id}, score), \ldots\} | it_{id} \in I \wedge us_{id} \in U \wedge score \in \{hi, mid, lo\} \\ \wedge\ us_{id} \text{ gives } it_{id} \text{ a rating of } score\}$$

We refer to the size of a rating database DB_R as $|DB_R|$ which is equal to the number of mapping entries in the database. The set of **common ratings** between two users in DB_R is the number of mapping entries that contain both users. By extension, the set of common ratings between two sets of users U_1 and U_2 in DB_R is the set of mapping entries that contain all users in the two sets, or mathematically:

Table 1. Example Rating Database 1

Item	User rating
i_1	a-hi, b-lo, d-lo
i_2	a-hi, b-lo, d-lo
i_3	a-hi, b-hi, d-hi
i_4	a-hi, b-lo, c-lo
i_5	a-hi, b-lo, c-lo
i_6	a-hi, b-hi, c-lo

$$\{rating = it_{id} \mapsto PAIRSET | rating \in DB_R \wedge \forall_{u \in (U_1 \bigcup U_2)} \cdot \\ \exists_{(us_{id}, score) \in PAIRSET} \cdot\ u = us_{id}\ \}$$

Definition 3.2. (Opposing Community): Let U_i and U_j be two disjoint sets of users. (U_i, U_j) is a pair of user sets we refer to as an opposing community (or simply, o-community).[2]

The number of common ratings between two sets of users U_i and U_j is known as their **support count** and is denoted by $count(U_i, U_j)$. The **support** of the two user sets denoted as $support(U_i, U_j)$ is defined as $\frac{count(U_i,U_j)}{|I|}$ where I represents the set of all items. The extent to which an opposing community is antagonistic is determined by its antagonistic count $antcount(U_i, U_j)$ which is defined as the number of common ratings between U_i and U_j that satisfy the following three conditions:

- Users from U_i share the same rating polarity p_i;
- Users from U_j share the same rating polarity p_j; and
- p_i and p_j are opposite polarities (i.e., one is high and the other is low).

It is obvious that $antcount\ (U_i, U_j) \leq count(U_i, U_j)$. The **antagonistic support** of the two user sets $asupport\ (U_i, U_j)$ is defined as $\frac{antcount(U_i,U_j)}{|I|}$. We also define the **antagonistic confidence** of an opposing community (U_i, U_j) to be $aconf(U_i, U_j) = \frac{antcount(U_i,U_j)}{count(U_i,U_j)}$.

Definition 3.3. (Frequent Opposing Community): An opposing community (U_i, U_j) is called a frequent opposing community (or, frequent o-community for short) if $support(U_i, U_j) \geq \lambda$ and $asupport(U_i, U_j) \geq \lambda \times \sigma$ where λ is the **minimum support threshold** ($\in (0, 1)$), and σ is the **minimum (antagonistic) confidence threshold** ($\in (0, 1)$).

We consider (U_i, U_j) **to subsume** (U'_i, U'_j) if: (a) $U'_i \subset U_i$ and $U'_j \subseteq U_j$; or (b) $U'_i \subseteq U_i$ and $U'_j \subset U_j$. We denote this by $(U'_i, U'_j) \subset (U_i, U_j)$. Frequent o-communities satisfy the important Apriori property as stated below.

Property 3.1. (Apriori Property of Frequent O-community): Every size-$(k-1)$ opposing community (U'_i, U'_j) subsumed by a size-k frequent o-community (U_i, U_j) is a frequent o-community.

Proof. Assume an opposing community g_{k-1} is not a frequent o-community. This

[2] The notion of opposing community is agnostic to the concepts of support and confidence described in the following paragraphs. It is simply a pair of user sets.

Table 2. Example Rating Database 2

Item	User rating
i_1	a-hi, b-lo, c-lo
i_2	a-hi, b-lo, c-lo
i_3	a-hi, b-lo, c-hi
i_4	d-hi, e-lo, f-lo
i_5	d-hi, e-hi

would mean $\frac{count(g_{k-1})}{|I|} < \lambda$ or $\frac{antcount(g_{k-1})}{|I|} < \lambda \times \sigma$. If an user u_k is added to either user set of this opposing community, we call the resulting opposing community $g_{k-1} \cup u_k$. $g_{k-1} \cup u_k$'s count cannot be more than count(g_{k-1}) and its antagonistic count cannot be more than antcount(g_{k-1}). This is because the count is calculated by intersecting the g_{k-1}'s user set's ratings and the u_k's ratings: count($g_{k-1} \cup u_k$) $\leq$ min{count(g_{k-1}),count(u_k)}, and similarly, the antagonistic count is calculated by intersecting g_{k-1}'s user set's ratings and u_k's ratings such that the intersected ratings have opposite polarity: antcount($g_{k-1} \cup u_k$) $\leq$ antcount(g_{k-1}). Therefore, $\frac{count(g_{k-1} \cup u_k)}{|I|} < \lambda$ or $\frac{antcount(g_{k-1}) \cup u_k}{|I|} < \lambda \times \sigma$; that is $g_{k-1} \cup u_k$ is not a frequent o-community neither. By taking its contrapositive, we can prove the property. $\square$

Definition 3.4. (Indirect Antagonistic Community): An opposing community (U_i, U_j) is an indirect antagonistic community (or, a-community for short) if it is a frequent o-community and $aconf(U_i, U_j) \geq \sigma$.

Definition 3.5. (Closed Indirect Antagonistic Community): An a-community (U_i, U_j) is closed if $\neg\exists(U_i', U_j'), (U_i, U_j) \subset (U_i', U_j')$, $count(U_i', U_j') = count(U_i, U_j)$ and $antcount(U_i', U_j') = antcount(U_i, U_j)$.

Example 1. Consider the example rating database in Table 1. Suppose $\lambda = 0.5$ and $\sigma = 0.5$. Both $(\{a\}, \{d\})$ and $(\{a\}, \{b, d\})$ are a-communities. However, since $count(\{a\}, \{d\}) = count(\{a\}, \{b, d\}) = 3$ and $antcount(\{a\}, \{d\}) = antcount(\{a\}, \{b, d\}) = 2$, $(\{a\}, \{d\})$ is not a closed a-community and is subsumed by $(\{a\}, \{b, d\})$. Hence, $(\{a\}, \{d\})$ is considered redundant or a non-closed a-community. On the other hand, both $(\{a\}, \{b\})$ and $(\{a\}, \{b, c\})$ are closed a-communities even though both $(\{a\}, \{b\})$ and $(\{a\}, \{b, c\})$ has the same $aconf$ value which is $\frac{2}{3}$. This is so as $count(\{a\}, \{b\}) \neq count(\{a\}, \{b, c\})$ and $antcount(\{a\}, \{b\}) \neq antcount(\{a\}, \{b, c\})$.

Note that we need to check $count()$ and $antcount()$ separately for closedness property. The example in Table 2 shows that $count(U_i, U_j) = count(U_i', U_j')$ does not imply that $antcount(U_i, U_j) = antcount(U_i', U_j')$ for any $(U_i, U_j) \subset (U_i', U_j')$, and vice versa. In this example, we have $count(\{a\}, \{b\}) = count(\{a\}, \{b, c\}) = 3$ but $(antcount(\{a\}, \{b\}) = 3) > (antcount(\{a\}, \{b, c\}) = 2)$. We also have $antcount(\{d\}, \{e\}) = antcount(\{d\}, \{e, f\}) = 1$ but $(count(\{d\}, \{e\}) = 2) > (count(\{d\}, \{e, f\}) = 1)$.

With the above definitions, we are now ready to define the problem of mining indirect antagonistic communities.

Definition 3.6. (Indirect Antagonistic Community Mining Problem): Given a set of items I rated by a set of users U (the rating database), the a-community mining problem is to find all closed a-communities with the given minimum support threshold λ and minimum (antagonistic) confidence threshold σ.

4. A-Community Mining Algorithm

We develop an algorithm to mine indirect a-communities from a rating database and a divide-and-conquer variant of it. The rating database represents people opinions or views on the rated items. Our algorithm systematically traverses the search space of possible antagonistic communities and uses a search space pruning strategy to effectively remove unfruitful search spaces.

4.1. Overview of Algorithm

Our a-community mining algorithm runs for multiple passes. In the initialization pass, we calculate the *count* and *antcount* of all size-2 a-community candidates and determine which of them are frequent o-communities. In the next pass, we generate new potential frequent o-communities, called *candidate* set, from the set of frequent o-communities found in the previous pass. We then count the actual *count* and *antcount* values for these candidates. At the end of this pass, we determine the frequent o-communities from these candidates. After that, we filter the previous frequent o-community set with the newly generated frequent o-community set to remove non-closed frequent o-communities. Then we move on to the next pass. Frequent o-communities of a pass are used to generate frequent o-community candidates in the next pass. This process continues until no larger frequent o-communities are found. After successful mining of all closed frequent o-communities, we derive the closed a-communities from them.

Algorithm 4.1 shows our a-community mining algorithm called Clagmine. Two basic data structures are maintained namely L_k the intermediary set of frequent o-communities of size k and C_k a candidate set of o-communities of size k. The first two lines of the algorithm derive size-2 candidates from which the frequent size-2 o-communities are obtained. It forms the base for subsequent processing. A subsequent pass, say pass k, consists of three phases. First, at line 5, the o-communities in L_{k-1} found in $k-1$ pass are used to generate the candidate o-community set C_k, using the antCommunityMining-gen method in Algorithm 4.2. Next, the database is scanned and the count and antcount of candidates in C_k is updated (lines 7 to 13). We make use of the hash-tree data structure described in [3] to hold C_k and we then use a subset function to find the candidates that overlap with the raters of an item. After we marked all the overlapped candidates, we update the count and antcount of them. Frequent o-communities can be determined by checking count and antcount against the support threshold and $\lambda \times \sigma$ thresholds respectively. Following that, L_{k-1} is filtered with the newly generated o-communities to remove non-closed o-communities (line 15). After all the passes, the valid o-communities are determined from the frequent o-community set (line 17). The following Section 4.2 to 4.5 zoom into the various components of the mining algorithm in more detail.

Algorithm 4.1 Mining Algorithm – Clagmine(λ,σ,DB_R,U_{Set})

```
Input: min. support thresh. $\lambda$; min. conf. thresh. $\sigma$; rating database $DB_R$; set
    of users $U$; set of items $I$
Output: closed a-communities of all sizes
1: $L_1 = \{\{u_i\} | \frac{count(\{u_i\},\{u_i\})}{|I|} \geq \lambda\}$;
2: $C_2 = \{(\{u_i\},\{u_j\}) | i < j, u_i \in L_1, u_j \in L_1\}$;
3: for $k = 2$;$k \leq |U|$ and $|L_{k-1}| \neq 0$; $k$++ do
4:   if $k > 2$ then
5:     $C_k = antCommunityMining\text{-}gen(L_{k-1})$; // See Algorithm 4.2
6:   end if
7:   root← $buildHashTree(C_k)$; // See Algorithm 4.4
8:   foreach item $t \in DB_R$ do
9:     $C_t = subset(t$,root); // See Algorithm 4.5
10:    foreach candidate $c$ in $C_t$ do
11:      update count and antcount of $c$;
12:    end for
13:  end for
14:  $L_k = \{g_k \in C_k | \frac{count(g_k)}{|I|} \geq \lambda$ and $\frac{antcount(g_k)}{|I|} \geq \lambda \times \sigma\}$;
15:  $L_{k-1} = prune(L_{k-1}, L_k)$; // See Algorithm 4.6
16: end for
17: $G = \{g \in \bigcup_k L_k | \frac{antcount(g)}{count(g)} \geq \sigma\}$;
18: return $G$;
```

4.2. Candidate Generation and Pruning

The antCommunityMining-gen procedure invoked at line 5 of Algorithm 4.1 and described in Algorithm 4.2, takes L_{k-1}, the set of all frequent size-$(k-1)$ o-communities as input. It returns a superset of all frequent size-k o-communities. It works as follows. First, we merge all the elements in L_{k-1} that share the same sub-community of size-(k-2) (line 3). Each of them can be merged into a size-k candidate o-community consisting of the common sub-community and the two differing members, as shown in Algorithm 4.3. We add the candidate o-communities to C_k (line 5). Next, in the pruning stage, we delete $g_k \in C_k$ if some $(k-1)$ subset of g_k is not in L_{k-1} (lines 6-11).

The pruning stage's correctness is guaranteed by Property 3.1. From the property, if g_k is frequent, all its $(k-1)$ subsets must be frequent. In other words, if any one $(k-1)$ subset of an o-community g_k is not frequent, g_k is not frequent too. We thus prune such g_ks. The correctness of antCommunityMining-gen procedure follows from Lemma 1.

Lemma 1. For $k \geq 3$, given a set of all size-$(k-1)$ frequent o-communities, i.e., L_{k-1}, every size-k frequent o-community, i.e., L_k, is in the candidate set, i.e., C_k, output by Algorithm 4.2.

Proof. From Property 3.1, any subset of a frequent o-community must also be frequent. Hence, if we extend each o-community in $L_{k-1}(k \geq 3)$ with all possible users and then delete all those whose $(k-1)$-subsets are not in L_{k-1}, we will be left with a superset of the o-communities in L_k. In the Algorithm 4.2, first we perform a merge process which is equivalent to extending L_{k-1} with all possible

Algorithm 4.2 antCommunityMining-gen(L_{k-1})

```
Input: size-$(k-1)$ o-community set $L_{k-1}$
Output: size-$k$ candidate o-community set
1: $C_k \leftarrow \emptyset$;
2: foreach $p, q \in L_{k-1}$ do
3:     $g_k \leftarrow merge(p, q)$; // See Algorithm 4.3
4:     if $g_k \neq null$ then
5:         add $g_k$ to $C_k$;
6:         foreach $(k-1)$-subsets $s$ of $g_k$ do
7:             if $s \notin L_{k-1}$ then
8:                 delete $g_k$ from $C_k$;
9:                 break;
10:            end if
11:        end for
12:    end if
13: end for
14: return $C_k$;
```

Algorithm 4.3 $merge((\{U_i\},\{U_j\}),(\{U_i'\},\{U_j'\}))$

```
Input: o-community $(\{U_i\},\{U_j\})$; o-community $(\{U_i'\},\{U_j'\})$
Output: merged result of the two input o-communities
1: if $U_i$=$U_i'$ and $diff(U_j,U_j')$=1 then
2:     return $(U_i,U_j \bigcup U_j')$;
3: end if
4: if $U_j$=$U_j'$ and $diff(U_i,U_i')$=1 then
5:     return $(U_i \bigcup U_i',U_j')$;
6: end if
7: if $U_i$=$U_j'$ and $diff(U_j,U_i')$=1 then
8:     return $(U_i,U_i' \bigcup U_j)$;
9: end if
10: if $U_i'$=$U_j$ and $diff(U_i,U_j')$=1 then
11:     return $(U_i \bigcup U_j',U_j)$;
12: end if
13: return $null$;
```

users in the database (line 2) and then at lines 4-8, we delete o-communities whose $(k-1)$-subsets are not in L_{k-1}. Thus after the merge and deletion steps, all frequent o-communities must be a subset of the returned candidate set. $\square$

An example to illustrate the process of candidate generation via merging and deletion is given below.

Example 2. Let L_3 be $((\{u_1\},\{u_2,u_3\}), (\{u_5\},\{u_2,u_3\}), (\{u_1,u_4\},\{u_2\}), (\{u_1,u_5\}, \{u_2\}), (\{u_4,u_5\},\{u_2\}))$. After the merge step performed, C_4 will contain candidate o-community $((\{u_1,u_5\},\{u_2,u_3\}), (\{u_1,u_4,u_5\},\{u_2\}))$. The deletion step, serving as apriori-based pruning, will delete the o-community $(\{u_1,u_5\},\{u_2,u_3\})$ because the o-community $(\{u_1,u_5\},\{u_3\})$ is not in L_3. We will then left with only $((\{u_1,u_4,u_5\},\{u_2\}))$ in C_4.

Algorithm 4.4 buildHashTree(C_k)

```
Input: $C_k$:size-$k$ candidate set
Output: root of the tree
 1: create new node root;
 2: foreach candidate $c_i$ in $C_k$ do
 3:    sort users in $c_i$ by their userID;
 4:    tempNode← root;
 5:    foreach user $u$ in $c_i$ do
 6:       if tempNode has a descendant $d$ labeled $u$ then
 7:          tempNode← $d$;
 8:       else
 9:          create node $d$ with label $u$;
10:          set $d$ as descendant of tempNode;
11:          tempNode← $d$;
12:       end if
13:       if $u$ is the last user in $c_i$ then
14:          set tempNode as a leaf node;
15:          add $c_i$ to tempNode;
16:       end if
17:    end for
18: end for
19: return root;
```

4.3. Subset Function

Candidate o-communities are stored in a hashtree as mentioned at line 7 of Algorithm 4.1. Each node of the hashtree contains either a hashtable (interior node), or a list of candidates (leaf). Each node is labeled with a user identifier representing the user associated with this node. The hashtable at interior nodes contains mappings to nodes at the next level, with each hash key being the corresponding user identifier. The building process of the hashtree is shown in Algorithm 4.4. Every candidate is sorted according to the user identifier, and is then inserted into the hashtree.

The subset function invoked at line 9 of Algorithm 4.1 finds all the candidate o-communities among raters of item t. The raters of item t is first sorted by their user identifiers. The raters are then traversed one by one. A pointer list is kept to maintain a list of nodes which are visited, which initially has only the root of the hashtree. For a rater u, we traverse through all the nodes in the pointer list, if a child node of the current node is found with label u, the child node is further checked to see whether it is an interior or a leaf node. If it is an interior node, we add it to the pointer list (line 9) and if it is a leaf, every o-community stored in the leaf is marked as a subset of raters of t (line 11). A node is removed from the pointer list if all of its child nodes are in the list (i.e., they are visited too) (lines 13-16). The process is repeated through all the raters of item t. At the end, every candidate which is a subset of raters of t will be marked.

Algorithm 4.5 subset(t,root)

```
Input: $t$:item in database; root:root of hashtree
Output: set of candidates contained in the set of raters of $t$
1: $C_t \leftarrow \emptyset$;
2: pointerRef ← empty node vector;
3: pointerRefSuffix ← empty node vector;
4: add root to pointerRef;
5: foreach rater $u$ of $t$ (in ascending order of their userIDs) do
6:   foreach node $node_i$ in pointerRef do
7:     if $node_i$ has descendant $d_i$ with label u then
8:       if $d_i$ is an interior node then
9:         add $d_i$ to pointerRefSuffix;
10:      else
11:        add o-communities stored in $d_i$ to $C_t$; // $d_i$ is a leaf node
12:      end if
13:      $node_i$'s descendant count−−;
14:      if $node_i$'s descendant count==0 then
15:        remove $node_i$ from pointerRef;
16:      end if
17:    end if
18:  end for
19:  append pointerRefSuffix to pointerRef;
20:  pointerRefSuffix← empty node vector;
21: end for
```

4.4. Filtering Non-Closed Antagonistic Communities

As a-communities are derived from frequent o-communities, we ensure the a-communities are closed by filtering out non-closed frequent o-communities. Note that as a closed frequent o-community could potentially subsume a combinatorial number of sub-communities, removal of non-closed frequent o-communities potentially reduces the number of frequent o-communities and a-communities significantly.

The filtering of non-closed frequent o-communities is performed by line 15 of Algorithm 4.1. Its pseudo code is shown in Algorithm 4.6. The procedure works as follows. For each frequent o-community g_k in L_k, we traverse through every frequent o-community g_{k-1} in L_{k-1} (lines 1-2). If g_k subsumes g_{k-1}, and the count and antcount of the two frequent o-communities are equal, g_{k-1} can be filtered (lines 3-4). This step ensures all the frequent o-communities in L_{k-1} are closed. By iterating through k, we can have all the non-closed frequent o-communities of any size filtered. Only closed frequent o-communities will remain.

The correctness of the algorithm is guaranteed by Theorems 1 & 2 stated below. The theorems guarantee that everything reported are correct and a complete set of closed indirect antagonistic communities are reported.

Theorem 1. Mined a-community set G contains all the closed a-communities.

Proof. Consider an arbitrary closed a-community g. Since g is an a-community, by definition, $\frac{count(g)}{|I|} \geq \lambda$ and $\frac{antcount(g)}{count(g)} \geq \sigma$. By multiplying the two, g also fulfills $\frac{antcount(g)}{|I|} \geq \lambda \times \sigma$. By Definition 3.3, g is a frequent o-community. Ac-

Algorithm 4.6 $prune(L_{k-1}, L_k)$

```
Input: frequent o-community set $L_{k-1}$; frequent o-community set $L_k$
Output: closed frequent o-community set of size k
1: foreach $g_k \in L_k$ do
2:   foreach $g_{k-1} \in L_{k-1}$ do
3:     if $g_{k-1} \subseteq g_k$ and count($g_{k-1}$)=count($g_k$) and antcount($g_{k-1}$)=
       antcount($g_k$) then
4:       remove $g_{k-1}$ from $L_{k-1}$;
5:     end if
6:   end for
7: end for
8: return $L_{k-1}$;
```

cording to Lemma 1, g will be in $C_{|g|}$. The a-community g can be captured by line 5 of Algorithm 4.1. As g fulfills both $\frac{count(g)}{|I|} \geq \lambda$ and $\frac{antcount(g)}{|I|} \geq \lambda \times \sigma$, the a-community g will be captured by line 14 of algorithm 4.1. Since g is closed, g will remain in $L_{|g|}$ after line 15 of algorithm 4.1. And finally, since $\frac{antcount(g)}{count(g)} \geq \sigma$, g will be added to G by line 17 of algorithm 4.1. Hence, every closed a-community will be contained in G. □

Theorem 2. Mined a-community set G contains only the closed a-communities.

Proof. Suppose an opposing community $g \in G$ is not antagonistic, that is, $\frac{count(g)}{|I|} < \lambda$ or $\frac{antcount(g)}{count(g)} < \sigma$. From line 17 of Algorithm 4.1, we can know $g \in \bigcup_k L_k$, and $\frac{antcount(g)}{count(g)} \geq \sigma$. However, every a-community g_k in $\bigcup_k L_k$ has $\frac{count(g_k)}{|I|} \geq \lambda$. Thus $\frac{count(g)}{|I|} \geq \lambda$. It contradicts $\frac{count(g)}{|I|} < \lambda$ or $\frac{antcount(g)}{count(g)} < \sigma$. Thus, g must be an a-community. Hence, G contains only a-communities. The closure property of G can be guaranteed by line 15 of Algorithm 4.1. Every valid a-community in G will be checked to filter out the non-closed ones. The filtering method will not leave any non-closed a-community in G. Hence G contains only closed a-communities. □

4.5. Scalability Variant: Divide and Conquer Strategy

At times, the main memory required to generate all the candidates could be prohibitive. If there are too many L_2 patterns, storing all of them in the memory would not be feasible. To address this issue, we perform a divide and conquer strategy by partitioning the database, mining each partition, and merging the partial results. We first state some new definitions and describe a property.

Definition 4.1. User Containment: Consider a member $m = it_{id} \mapsto PairSet$ in a rating database DB_R. We say that a user u_i is contained in the entry, denoted by $u_i \in m$, iff $\exists$ $(u_i, score)$ where $score \in \{hi, lo, mid\}$ and $(u_i, score)$ is in $PairSet$. We also say that a user u_i is in an o-community $a = (S_1, S_2)$ iff $(u_i \in S1 \vee u_i \in S2)$

Example 3. To illustrate, consider the first entry etr in the example rating

Table 3. Projected Rating Database 1 on User d

Item	User rating
i_1	a-hi, b-lo, d-lo
i_2	a-hi, b-lo, d-lo
i_3	a-hi, b-hi, d-hi

database shown in Table 1(left). The first entry *etr* contains users a, b and d; in other words, $a \in etr$, $b \in etr$, and $d \in etr$.

Definition 4.2. Database Partition: Consider a user u_i and a database of ratings DB_R. The partition of the database with respect to user u_i, denoted as $DB_R[u_i]$, is defined as: $\{etr|u_i \in etr \wedge etr \in DB_R\}$

Example 4. To illustrate, projection of the database shown in Table 1 with respect to user d is the database shown in Table 3.

Using the two definitions above, Lemma 2 describes our divide and conquer strategy.

Lemma 2. Divide and Conquer: Consider a database of ratings DB_R, minimum support threshold λ, and minimum confidence threshold σ. Let U and I be the set of users and items in DB_R. Also, let Cm be the shorthand of the Clagmine procedure described in Algorithm 4.1. The following is guaranteed:

$$Cm(\lambda, \sigma, DB_R, U, I) = \bigcup_{u_i \in U} \{g|u_i \in g \wedge g \in Cm(\lambda, \sigma, DB_R[u_i], U, I)\}$$

Proof. An entry in the database could only be counted as an additional support to an a-community containing user u_i iff the entry contains an item u_i. Hence, partitioning the database with respect to a user u_i would return the relevant portion of the database that is relevant to u_i. The support *count* and *antcount* of an arbitrary a-community containing u_i in the partitioned database $DB_R[u_i]$ would be the same as that in the original database DB_R. All a-communities reported in $Cm(\lambda, \sigma, DB_R[u_i], U, I)$ that contains u_i would have the correct support. All a-communities containing u_i should be output by $Cm(\lambda, \sigma, DB_R[u_i], U, I)$. However, nothing is guaranteed for a-communities that do not contain u_i in the set returned by $Cm(\lambda, \sigma, DB_R[u_i], U, I)$ – they could have a wrong support. They should be dropped.

Hence, it could be easily seen that the union of the mining results over the partitions with various u_is, with removal of results that does not contain u_i would be equal to the results returned by the mining operation on the entire dataset. □

Based on Lemma 2, our algorithm to perform divide and conquer is shown in Algorithm 4.7. The algorithm partitions the database one item at a time and subsequently calls the original closed antagonistic community mining algorithm defined in Algorithm 4.1. Theorem 3 guarantees that the mined result is correct and a complete set of a-communities are mined by Algorithm 4.7.

Theorem 3. Algorithm 4.7 would return a complete set of closed a-communities and all returned a-community would be closed.

Proof. From Theorems 1 & 2 and Lemma 2 it is easy to see that the above theorem holds. □

Algorithm 4.7 *Clagmine-partitional*(λ,σ,DB_R,U,I)

```
Input: min. support thresh. $\lambda$; min. conf. thresh. $\sigma$; rating database $DB_R$; set
    of users $U$; set of items $I$
Output: closed a-communities of all sizes
1: $G = \{\}$;
2: foreach $u_i \in U$ do
3:     $G = G \cup \{g | u_i \in g \wedge g \in Clagmine(\lambda,\sigma,DB_R[u_i],U,I)\}$;
4: end for
5: return $G$;
```

Note that the divide and conquer algorithm reduces memory costs however it could potentially increase the runtime cost since the database would now need to be scanned more number of times. In Section 5, we show the results of running the two algorithms over a number of datasets.

5. Performance Study

In this section, we describe our performance study on synthetic datasets generated by our synthetic data generator, and real datasets from Amazon, Epinions, and Slashdot. In addition to the performance study results, we also highlight some interesting findings by analyzing the a-communities mined from these datasets. All experiments are conducted on a desktop PC with 3.17GHz CPU and 3GB RAM.

5.1. Experiments on Synthetic Datasets

We evaluate the scalability of our algorithm on synthetic datasets generated by a synthetic data generator on various parameter settings. Our synthetic data generator accepts five input parameters: number of users $|U|$ (in '000), number of items $|I|$ (in '000), the expected number of users rating an item P, average size of maximal potential large a-community N_G, and number of maximal potential large a-community N_L (in '000). We generate four datasets using the following parameter settings.

Dataset	Parameters
DS_1	$|U|$=10, $|I|$=100, P=20, N_G=6, N_L=2
DS_2	$|U|$=50, $|I|$=100, P=20, N_G=6, N_L=2
DS_3	$|U|$=10, $|I|$=100, P=30, N_G=6, N_L=2
DS_4	$|U|$=50, $|I|$=10, P=20, N_G=6, N_L=2

For all the experiments, we have set the minimum confidence threshold σ at 0.7. We measure the runtime for different support thresholds. The results for dataset DS_1 for support thresholds from 0.002 to 0.006 are shown in Figure 1. Figure 1(a) shows the runtime needed to execute the algorithm at various support thresholds. "Non-Split" and "Split" correspond to *Clagmine* algorithm and *Clagmine-partitional* algorithm respectively. We only include 3 data points for "Non-Split", as mining at lower thresholds took too long to complete. Figure

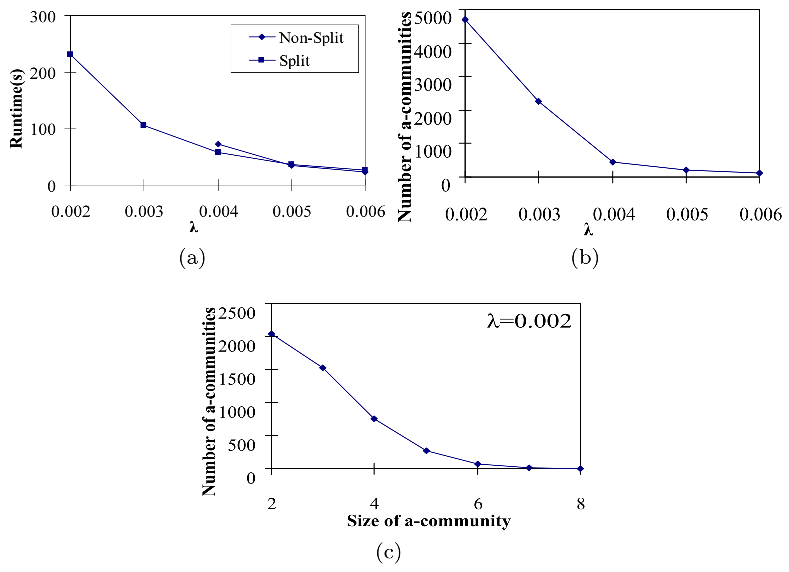

Fig. 1. Runtime and Patterns: DS_1 at various minimum support thresholds (i.e., λ) with σ=0.7.

1(b) shows the number of a-communities found at various support thresholds. Finally in Figure 1(c), we plot a graph showing the number of a-communities of different sizes when we mine with the minimum support threshold set at 0.002.

The result shows that the number of a-community as well as the runtime decreases with increasing support threshold. Figure 1(c) also shows that a-communities mined have small sizes.

For the second dataset, we consider a larger set of users. The results for various support thresholds with σ=0.7 are shown in Figure 2.

For the third dataset, we use a smaller set of users and larger expected set of users rating an item. The results for various minimum support thresholds with σ=0.7 are shown in Figure 3.

For the fourth dataset, we consider a smaller set of items and larger set of users. The results for various minimum support thresholds with σ=0.7 are shown in Figure 4.

The performance study shows that the algorithm is able to run well on various settings. The lower the support threshold is, the larger are the runtime and the number of a-communities. DS_1 and DS_2 have the same parameter settings except that the number of users of DS_2 is larger than that of DS_1. By comparing the runtime of the two, we conclude that the larger the number of users is, the more time consuming it is to mine a dataset. Similarly, by comparing DS_2 and DS_4, we conclude that the larger the number of items is, the more time consuming it is to mine a dataset. Comparing DS_3 and DS_1, we conclude that the larger the expected number of users rating an item is, the more time consuming it is to mine a dataset.

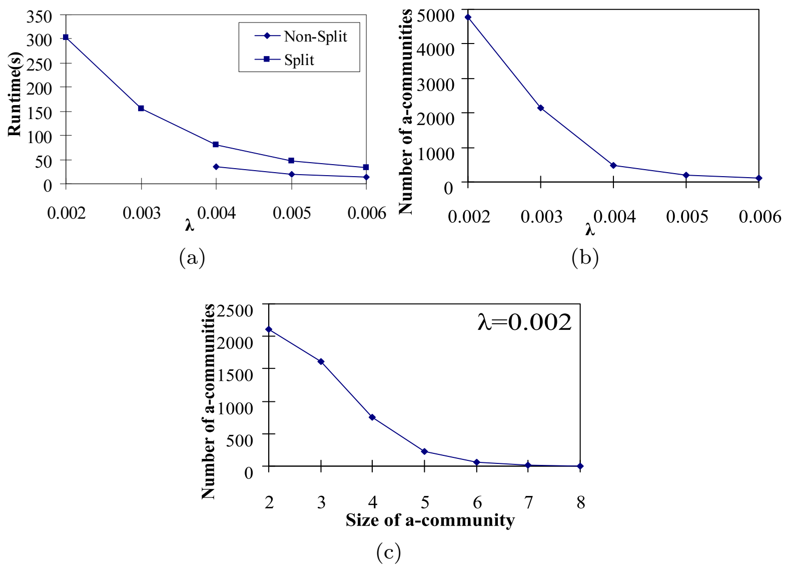

Fig. 2. Runtime and Patterns: DS_2 at various minimum support thresholds (i.e., λ) with σ=0.7.

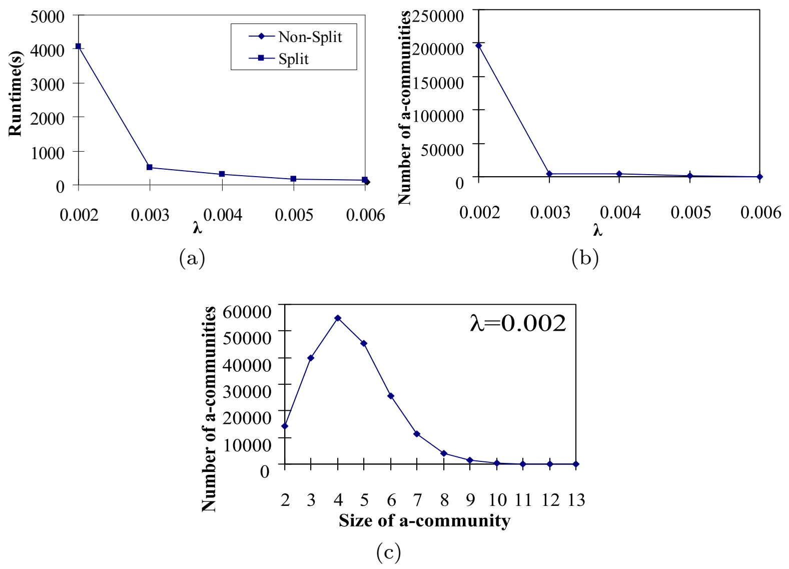

Fig. 3. Runtime and Patterns: DS_3 at various minimum support thresholds (i.e., λ) with σ=0.7.

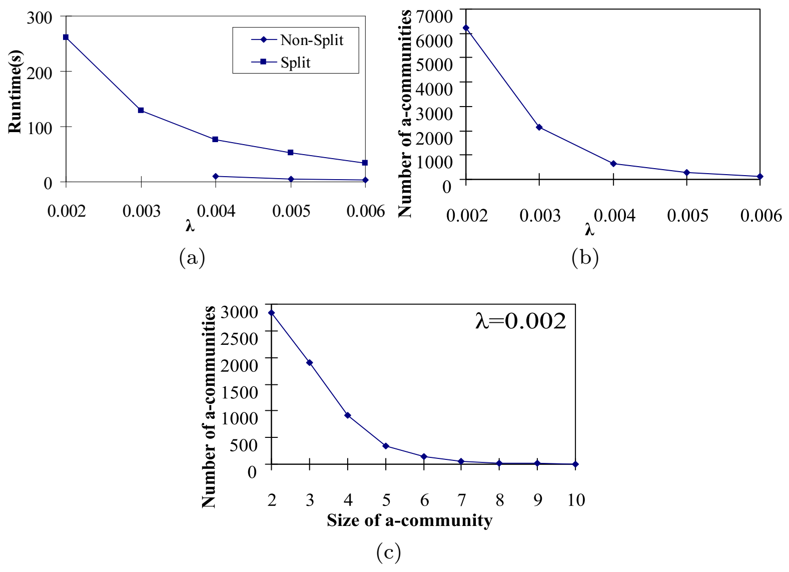

Fig. 4. Runtime and Patterns: DS_4 at various minimum support thresholds (i.e., λ) with σ=0.7.

5.2. Experiments on Amazon Dataset

We describe the Amazon dataset, the result of the performance study, and the findings of the efficacy analysis on the resultant mined a-communities.

5.2.1. Dataset Description

We obtain the Amazon dataset from Bing Liu's group in the University of Illinois at Chicago. In this dataset, there are a total of 99,255 users rating 108,142 books in 935,051 reviews. Each review is associated with a rating issued to the item by the user. The rating ranges from 1 to 5. We map ratings of 4-5 to high polarity (hi), ratings of 1-2 to low polarity (lo), and the rest are mapped to medium polarity (mid). Among the 935,051 ratings, 699,925 (74.9%) are high, 108,013 (11.6%) are low, and 104,373 (11.2%) are medium. The distribution of number of users rating an item versus number of items (i.e., indegree distribution), and the distribution of the number of items a user rates versus the number of users (i.e., outdegree distribution) are shown in Figure 5[3]. They follow power law distribution. This agrees with the "power law degree distribution" of large networks [32, 4], though there are some outliers. In Figure 5(a), when indegree equals 1 or 2, the number of nodes is much fewer than the expected values following power law. Similar cases exist in Figure 5(b). This suggests that Amazon book rating dataset has a small number of nodes (items or users) with extremely low indegrees or outdegrees.

[3] We represent a rating to an item as an inlink to the item. A rating issued by a user is represented as an outlink from the user.

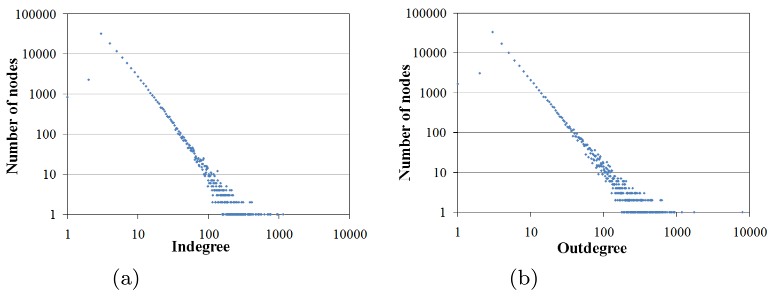

Fig. 5. Amazon Book Ratings Dataset: Indegree and Outdegree Distribution.

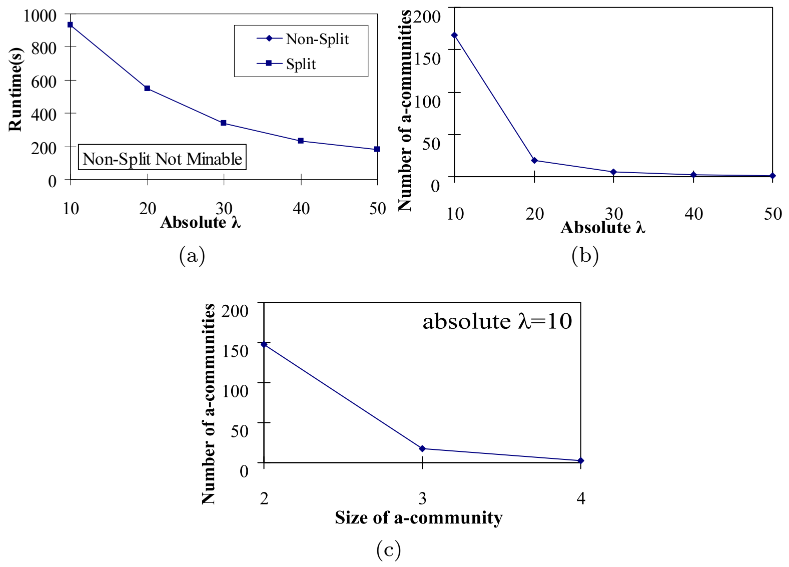

Fig. 6. Amazon Dataset: Runtime and Distribution of A-Communities.

5.2.2. Performance Study

The performance study is conducted with σ=0.5 and different λ values. The results are shown in Figure 6. Figure 6(c) is obtained with absolute λ=10 (we call $\lambda \times |I|$, where $|I|$ is the number of different items, as absolute λ).

Figure 6(b) shows that the number of a-communities mined is small even with low minimum support thresholds. Most of the a-communities are of size 2. This result shows that Amazon dataset does not contain large groups of people with opposite opinions. A possible explanation is because Amazon makes use recommendation system extensively so that most items are shown to users with the same preferences/opinions (thus, reducing the opposing ratings).

Table 4. Z-test of Item Metrics for Amazon Dataset

		Item HRR	Item BR
A-community rated items	Size	1379	1379
	Mean	0.767	39.381
	Std Dev.	0.189	75.148
General items	Size	106582	106763
	Mean	0.875	7.059
	Std Dev.	0.206	12.105
z value		-21.137	15.969

5.2.3. Efficacy Analysis

In our efficacy analysis, we investigate if items rated by an a-community (called *a-community rated items*) differ from other items (called *general items*). We also investigate if users participating in an a-community (called *a-community users*) differ from other users (called *general users*). We analyze the a-communities mined with absolute λ=10 and σ=0.5.

A-Community Rated Items vs. General Items. We use the following *item metrics* to compare a-community rated items and general items:

1. **Item High Rating Ratio (Item HRR)**$=\frac{\#high\_rating}{\#high\_rating+\#low\_rating}$. This metric reflects the controversial level of an item. The closer the metric to 0.5, the more controversial the item is.
2. **Item Biased Rating (Item BR)**$=\#high\_rating+\#low\_rating$. This metric reflects how many biased ratings (high and low ratings) an item attracts.

For a-community rated items and general items, we obtain the mean and standard deviation for each of the above metrics. We then perform a z-test on each metric to tell if the set of a-community rated items is different from that of general items. Our z-test results are shown in Table 4. For the two sets to be similar with 99% confidence, we need the z-value to be within [-2.57,2.57] range. Table 4 shows that the z-values of the two metrics are all outside the range. Hence, we can say with 99% confidence, the two populations are different with respect to the two metrics. Furthermore, we observe that:

1. In terms of item HRR, both a-community rated items and general items receive more high ratings than low ones. The high and low ratings of the a-community rated items are more balanced than that of general items. Hence, the a-community rated items attract significantly more opposing ratings than general items.
2. In terms of item BR, a-community rated items attract significantly more biased ratings than general items.

A-Community Users vs. General Users. We use the following *user metrics* to compare the two user sets:

1. **User High Rating Ratio (User HRR)**$=\frac{\#high\_rating}{\#high\_rating+\#low\_rating}$. This metric reflects whether a user's opinions are biased towards high or low ratings.
2. **User Biased Rating (User BR)**$=\#high\_rating+\#low\_rating$. This metric reflects how many biased ratings (high and low ratings) a user gives..

Table 5. Z-test of User Metrics for Amazon Dataset

		User HRR	User BR
A-community users	Size	166	166
	Mean	0.693	208.572
	Std Dev.	0.264	634.060
General users	Size	39517	39517
	Mean	0.846	15.108
	Std Dev.	0.196	29.820
z value		-7.468	3.931

To compare the two user sets, we only consider users with User BR $\geq$ (absolute λ) $\times$ σ. This requires users to rate at least (absolute λ) $\times$ σ items as high or low. Thus, their user BRs are at least (absolute λ) $\times$ σ. We apply z-test to investigate if the distributions of the metrics for the two user sets are different. The z-test results are shown in Table 5. The z-values of the two metrics are all outside the 99% confidence interval (i.e., [-2.57,2.57]). Hence, we can say with 99% confidence that the two user sets are different with respect to the two metrics. From the table, we observe that:

1. In terms of user HRR, both a-community users and general users give more high ratings than low ones. A-community users give more balanced high and low ratings than general users.
2. In terms of user BR, a-community users give significantly more biased ratings than general users.

5.3. Experiments on Epinions Dataset

We describe the Epinions dataset, the result of the performance study, and the findings of the efficacy analysis on the resultant mined a-communities.

5.3.1. Dataset Description

We analyze the Epinions dataset downloaded from [1] which contains 49,290 users who rated 139,738 different items in 664,823 reviews with ratings scale from 1 to 5. Again, we map ratings of 4-5 to high polarity (hi) and ratings of 1-2 to low polarity (lo). The rest are mapped to medium polarity (mid). Among the 664,823 ratings, 495,392 (74.5%) are high, 93,906 (14.1%) are low, and 75,525 (11.4%) are medium. The indegree distribution of items and the outdegree distribution of the users are shown in Figure 7. Both the indegree and outdegree distributions follow power law.

5.3.2. Performance Study

The performance study is conducted with σ=0.5 and different λ values. The results are shown in Figure 8. Figure 8(c) is obtained with absolute λ=10.

The results of Epinions dataset are similar to that of Amazon dataset. The number of a-communities is small even with very low support threshold. Most of the a-communities are of size 2. The antagonistic behavior is not so much apparent in this dataset.

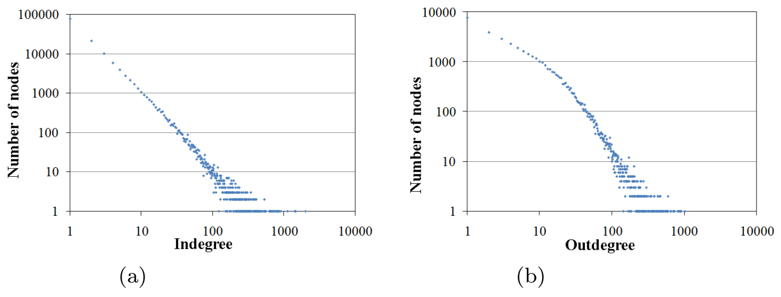

Fig. 7. Epinions Dataset: Indegree and Outdegree Distribution.

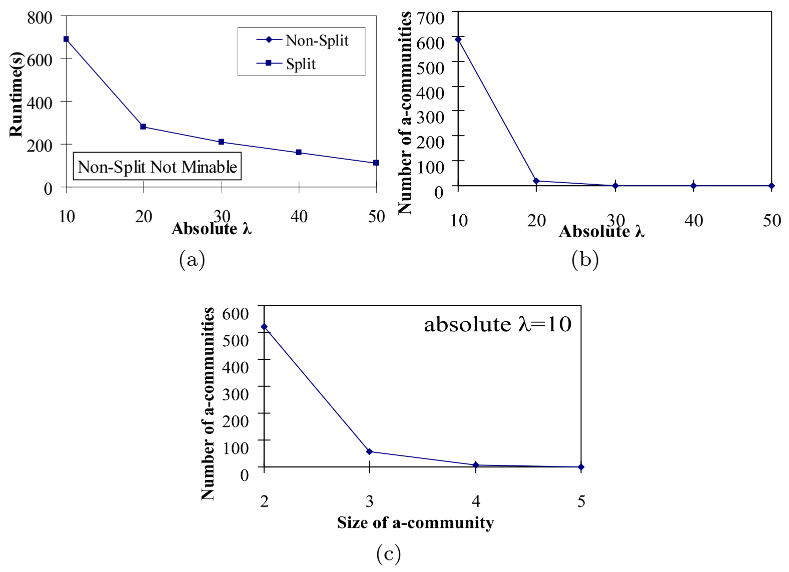

Fig. 8. Epinions Dataset: Runtime and Distribution of A-Communities.

5.3.3. Efficacy Analysis

We analyze a-community rated items and users for a-communities mined with absolute λ=10 and σ=0.5.

A-Community Rated Items vs. General Items. We compare the two sets using the same item metrics and z-test used in analyzing the Amazon dataset. Our z-test results are shown in Table 6. The z-values of the two metrics are all outside the 99% confidence interval (i.e., [-2.57,2.57]). Hence, we can say with 99% confidence that the two sets are different with respect to the two metrics. We also observe that:

1. In terms of item HRR, similar to the Amazon dataset, both of the two sets receive more high ratings than low ones. The high and low ratings received by a-community rated items are more balanced than that of general items.

Table 6. Z-test of Item Metrics for Epinions Dataset

		Item HRR	Item BR
A-community rated items	Size	1503	1503
	Mean	0.732	93.582
	Std Dev.	0.236	124.059
General items	Size	128015	138235
	Mean	0.874	3.246
	Std Dev.	0.293	7.872
z value		-23.145	28.229

Table 7. Z-test of User Metrics for Epinions Dataset

		User HRR	User BR
A-community users	Size	434	434
	Mean	0.772	146.150
	Std Dev.	0.136	128.980
General users	Size	21873	21873
	Mean	0.845	22.461
	Std Dev.	0.132	32.039
z value		-11.108	19.966

2. In terms of item BR, a-community rated items receive significantly more biased ratings than general items.

A-Community Users vs. General Users. We compare the two sets using the same user metrics and z-test used in analyzing the Amazon dataset. Our z-test results are shown in Table 7. The z-values of the two metrics are all outside the 99% confidence interval (i.e., [-2.57,2.57]). Hence, we can say with 99% confidence that the two populations are different with respect to the two metrics. From the table, we can also observe that:

1. In terms of user HRR, similar to the Amazon dataset, both a-community users and general users give more high ratings than low ones. A-community users give more balanced high and low ratings than general users.
2. In terms of user BR, a-community users tend to give significantly more biased ratings than general users.

5.4. Experiments on Slashdot Dataset

We describe the Slashdot dataset, the result of the performance study, and the findings of the efficacy analysis on the resultant mined a-communities.

5.4.1. Dataset Description

Different from our previous two datasets, Slashdot dataset does not contain ratings people give to products. Rather, it contains ratings people give to other people. In this dataset, a person rates another as a "friend" (a high polarity rating) or "enemy"(a low polarity rating). There is no medium polarity rating in this dataset. Items here refer to the people receiving at least one rating and users refer to the people giving at least one rating.

We analyze the Slashdot dataset downloaded from [2] which contains information on 82,144 individuals. Forty four thousands forty four (53.6%) of them

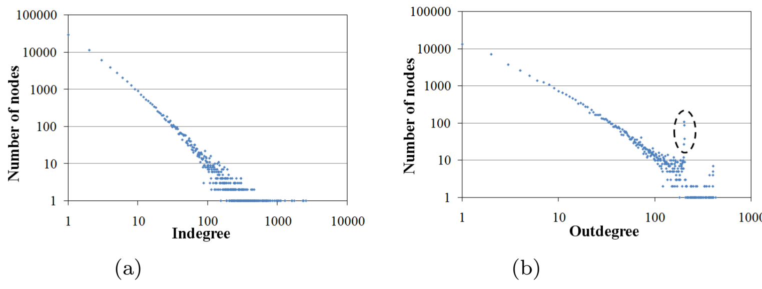

Fig. 9. Slashdot Dataset: Indegree and Outdegree Distribution.

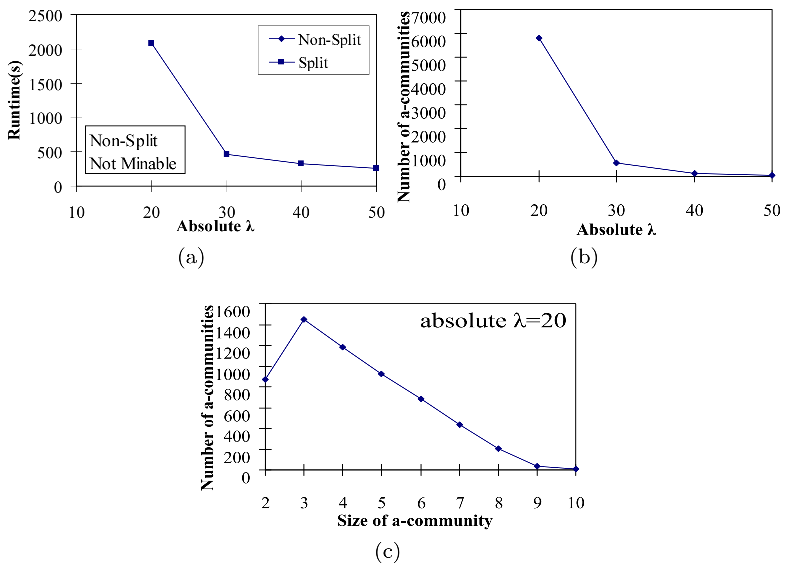

Fig. 10. Slashdot Dataset: Runtime and Distribution of A-Communities.

give at least one rating and 70,284 (85.6%) have at least one rating. The difference between the number of people that have at least one rating and the number of people that give at least one rating is 26,240. This indicates that some people give many ratings. There are 549,202 links, with 425,072 (77.4%) of them having high rating, and 124,130 (22.6%) of them having low rating. This suggests that users in Slashdot dataset give much more high ratings than low ratings. The indegree and outdegree distributions are shown in Figure 9. As shown in Figure 9(a), the indegree is strictly power law distributed. Figure 9(b) shows that the outdegree follows power law too except four nodes in the dashed circle.

5.4.2. Performance Study

The performance study is conducted with σ=0.7 and different λ values. The result is shown in Figure 10. Figure 10(c) is obtained with absolute λ=20.

Table 8. Z-test of Item Metrics for Slashdot Dataset

		Item HRR	Item BR
A-community rated items	Size	2556	2556
	Mean	0.647	81.172
	Std Dev.	0.220	147.063
General items	Size	67728	67728
	Mean	0.779	5.046
	Std Dev.	0.347	12.632
z value		-28.986	26.167

Figure 10(a) shows that the runtime decreases with larger λ. This is due to smaller number of a-communities as shown in Figure 10(b). Unlike the Epinions and Amazon datasets, most of the a-communities in this dataset are of size 3 and some large size a-communities are mined. For example, we have around 200 a-communities of size 8. As the size of a-community increases, the number of a-communities decreases.

5.4.3. Efficacy Analysis

We analyze a-community rated items and users for a-communities mined with absolute λ=20 and σ=0.7.

A-Community Rated Items vs. General Items. We compare the two sets using the same item metrics and z-test used in analyzing the previous two datasets. Our z-test results are shown in Table 8. The z-values of the two metrics are all outside the 99% confidence interval (i.e., [-2.57,2.57]). Hence, we can say with 99% confidence that the two sets are different with respect to the two item metrics. We also observe that:

1. In terms of item HRR, similar to our previous two datasets, both sets receive more high ratings than low ones. A-community rated items receive more balanced high and low ratings than general items.
2. In terms of item BR, a-community rated items receive significantly more biased ratings than general items.

A-Community Users vs. General Users. We compare the two sets using the same user metrics and z-test used in analyzing the previous two datasets. Our z-test results are shown in Table 9. The z-values of the two metrics are all outside the 99% confidence interval (i.e., [-2.57,2.57]). The z-values indicate that with 99% confidence we can say the two sets are different with respect to the two metrics. We also observe that:

1. In terms of user HRR, similar to our previous two datasets, both a-community users and general users give more high ratings than low ones. A-community users give more balanced high and low ratings than general users.
2. In terms of user BR, a-community users give significantly more biased ratings than general users.

5.4.4. Examples of Mined A-Communities

We show examples of some interesting a-communities are discovered from the Amazon dataset. The Amazon dataset is particularly rich since it contains ratings

Table 9. Z-test of User Metrics for Slashdot Dataset

		User HRR	User BR
A-community users	Size	399	399
	Mean	0.682	182.727
	Std Dev.	0.344	91.403
General users	Size	7704	7704
	Mean	0.790	46.106
	Std Dev.	0.245	44.925
z value		-6.199	29.672

Table 10. Interesting Examples from Amazon Book Rating Dataset

ID	Antagonistic Groups	Num. of Common Ratings	Num. of Ratings by User 1 (% of Common Ratings)	Num. of Ratings by User 2 (% of Common Ratings)	Num. of Ratings by User 3 (% of Common Ratings)
1	({Johnston},{Weissgarber})	12	56 (21%)	13 (92%)	-
2	({Johnston, Jump},{Weissgarber})	10	56 (17.8%)	61 (16.4%)	13 (76.9%)
3	({Johnston, Hill},{Weissgarber})	10	56 (17.8%)	106 (9.4%)	13 (76.9%)
4	({Leeper},{Weissgarber})	10	137 (7.3%)	13 (76.9%)	-
5	({Kern},{Sklarski})	14	452 (3.1%)	22 (63.6%)	-

of books of various types and genres and each rating potentially comes with comments that tell us why a particular user like or dislike a particular item.

We run the mining algorithm on Amazon dataset with absolute λ=10 and σ=0.5 and analyze the mined a-communities. The program runs for 930 seconds with 167 a-communities generated. A hundred and forty seven of the a-communities are of size 2, 18 of them are of size 3 and 2 of them are of size 4. We post-process the a-communities with the following criterion:

– $\frac{number\_of\_commonly\_rated\_items}{number\_of\_rated\_items}$: Retain a-communities if at least one user in the a-community has $\frac{number\_of\_commonly\_rated\_items}{number\_of\_rated\_items} > 0.6$. This criteria is to ensure that at least one user behaves highly antagonistically against others.

After post-processing, we note five of the most interesting a-communities. We select those having highest antagonistic confidence and average $\frac{number\_of\_commonly\_rated\_items}{number\_of\_rated\_items}$ scores over all constituent users. They are shown in Table 10. We select the first a-community and observe the following:

– *High antagonistic level*: We observe that the two users in the first a-community rated with a high level of antagonism. Among Jason Johnston's 56 rated books, 12 have ratings opposite to the ratings by Luke Weissgarber. Similarly for Weissgarber, 12 of all his 13 rated books have ratings opposite to those by Johnston, which means more than 92% of Weissgarber's ratings are opposite to Johnston's. It is a significantly high figure.
– *Antagonistically rated books*: We found that books given opposite ratings by Weissgarber and Johnston are some novels with a similar story background. These books are clearly liked by Johnston but not by Weissgarber. Figure 11 lists the books rated oppositely by Johnston and Weissgarber. When we look into the comments of the ratings made by Johnston and Weissgarber for the

Table 11. Johnston and Weissgarber's Ratings on Their Commonly Rated Books

ID	Book title	Johnston's rating	Weissgarber's rating
1	Armageddon	4	1
2	The Remnant: On the Brink of Armageddon	4	1
3	Desecration: Antichrist Takes the Throne	4	1
4	The Mark: The Beast Rules the World	4	1
5	The Indwelling: The Beast Takes Possession	4	1
6	Assassins	4	1
7	Apollyon: The Destroyer Is Unleashed	4	1
8	Soul Harvest: The World Takes Sides	4	1
9	Nicolae: The Rise of Antichrist	4	1
10	Tribulation Force: The Continuing Drama of Those Left Behind	4	1
11	Left Behind: A Novel of the Earth's Last Days	4	1
12	Glorious Appearing: The End of Days	4	1

books in Figure 11, we find out that the books are deemed as heretical by Weissgarber, but liked by Johnston.

– *Antagonistically behaved users*: It is interesting that Weissgarber appears in four out of five a-communities. His ratings are opposite to other 4 users for at least 10 books. He tends to rate books against the ratings of others.

6. Discussion

In this section we discuss several interesting points on the number of non-closed patterns, the amount of useless patterns mined by the partitioning approach, and the relationship between the power law nature of the indegree and outdegree distributions with the size of mined patterns.

Number of non-closed patterns pruned. We perform late pruning (i.e., filtering) of non-closed patterns. This is done at line 15 of Algorithm 4.1. We would like to investigate the number of non-closed patterns that are filtered. Tables 12 & 13 show the number of non-closed patterns that are filtered for the synthetic and real datasets respectively. We notice that the number of non-closed patterns filtered for Amazon and Epinions datasets are not that many. Thus, employing early pruning of non-closed patterns are not useful for these datasets (at least for the support thresholds considered). Indeed, early pruning of non-closed patterns typically include additional checks with incurs computational cost, e.g., [39]. Thus unless there are many non-closed patterns, employing early pruning of non-closed patterns would not improve and might even reduce the efficiency of the mining process. On the other hand the number of non-closed patterns filtered for Slashdot and the synthetic datasets are many. For these cases, employing early pruning of non-closed patterns would be very useful in improving efficiency. We leave this as a future work.

Number of useless patterns generated. In the partitioning approach (i.e., divide and conquer strategy described in Section 4.5), we might generate useless patterns which are later filtered, i.e., the patterns do not include the user used to create the partition. We would like to analyze the number of such useless patterns. Tables 14 & 15 show the number of useless patterns that are filtered for

Table 12. Number of Non-closed Patterns in Synthetic Datasets

Dataset	λ	#Non-closed	#Closed
DS_1	0.002	8,847	8,083
	0.003	898	3,411
	0.004	177	1,806
	0.005	26	1,031
	0.006	1	721
DS_2	0.002	10,534	8,129
	0.003	1,228	3,648
	0.004	89	1,699
	0.005	11	1,003
	0.006	0	705
DS_3	0.002	810,881	149,675
	0.003	20,425	7,816
	0.004	1,674	6,963
	0.005	463	2,215
	0.006	292	1,736
DS_4	0.002	315,092	7,299
	0.003	26,721	3,592
	0.004	5,450	1,875
	0.005	981	1,156
	0.006	224	717

the synthetic and real datasets respectively. We notice that the number of useless patterns ranges from 0 to 77% of the number of closed patterns. This shows a tradeoff between speed and memory consumption. Without the partitioning strategy, there is no such useless patterns however the memory consumption needed is too large for many cases such that the algorithm crashes due to out-of-memory exception.

Power law and pattern size. We notice that the indegree and outdegree distributions for the three real datasets follow the power law (see Figures 5, 7 & 9). Thus only a few nodes have large indegree and outdegree values. This observation matches with the mining result: we find that in general most patterns that we mine are of small sizes and large patterns are less in number than small patterns (see Figures 6c, 8c & 10c).

7. Conclusion

In this study, we propose a new pattern mining algorithm to mine indirect antagonistic communities from social interactions. Our algorithm traverses the search space of possible antagonistic communities and uses a pruning strategy to remove unfruitful search spaces that do not contain any antagonistic community. We also propose a variant of the algorithm that utilizes a divide and conquer strategy which enables us to mine larger datasets. Performance studies have been conducted on various synthetic datasets to show the scalability of our approach on various dataset sizes and parameter values. We also mine antagonistic communities from Amazon, Epinions, and Slashdot datasets. The results show that the algorithm is effective in finding indirect antagonistic communities from these

Table 13. Number of Non-closed Patterns in Real Datasets

Dataset	Absolute λ	#Non-closed	#Closed
Amazon	10	66	167
	20	2	19
	30	0	6
	40	0	2
	50	0	1
Epinions	10	873	586
	20	2	19
	30	0	1
	40	0	0
	50	0	0
Slashdot	20	243,236	5,802
	30	7,926	551
	40	1,279	126
	50	494	51

Table 14. Number of Useless Patterns in Synthetic Datasets

Dataset	λ	#Useless	#Closed	%Useless
DS_1	0.002	17,549	8,083	68.47%
	0.003	5,052	3,411	59.70%
	0.004	2,415	1,806	57.21%
	0.005	1,265	1,031	55.10%
	0.006	826	721	53.39%
DS_2	0.002	17,676	8,129	68.50%
	0.003	5,799	3,648	61.38%
	0.004	2,284	1,699	57.34%
	0.005	1,263	1,003	55.74%
	0.006	837	705	54.28%
DS_3	0.002	529,239	149,675	77.95%
	0.003	17,092	7,816	68.62%
	0.004	14,039	6,963	66.85%
	0.005	3,100	2,215	58.33%
	0.006	2,444	1,736	58.47%
DS_4	0.002	14,606	7,299	66.68%
	0.003	5,829	3,592	61.87%
	0.004	2,649	1,875	58.55%
	0.005	1,487	1,156	56.26%
	0.006	875	717	54.96%

datasets. Furthermore, we found that items rated by, and users participating in, an antagonistic community are significantly different from general items and users respectively. In the future, we plan to further speed up the mining algorithm and investigate antagonistic communities in other social network settings.

Acknowledgements. This work is supported by National Research Foundation of Singapore under project NRF2008IDM-IDM004-036. We would like to thank Paolo Massa for sharing his Epinions dataset. We would also like to thank Bing Liu for sharing the book ratings dataset.

Table 15. Number of Useless Patterns in Real Datasets

Dataset	Absolute λ	#Useless	#Closed	%Useless
Amazon	10	189	167	53.09%
	20	21	19	52.5%
	30	6	6	50%
	40	2	2	50%
	50	1	1	50%
Epinions	10	662	586	53.04%
	20	19	19	50%
	30	1	1	50%
	40	0	0	0%
	50	0	0	0%
Slashdot	20	18,863	5,802	76.48%
	30	1,236	551	69.17%
	40	255	126	69.93%
	50	104	51	67.10%

References

[1] Downloaded Epinions Dataset - Trustlet. http://www.trustlet.org/wiki/Downloaded_Epinions_dataset/ratings_data.txt.bz2.

[2] Stanford large network dataset collection. http://snap.stanford.edu/data/.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of International Conference on Very Large Data Bases*, 1994.

[4] R. Albert, H. Jeong, and A. L. Barabasi. The diameter of the world wide web. *Nature*, 401:130–131, 1999.

[5] P. Bonacich and P. Lloyd. Calculating status with negative relationships. *Social Networks*, pages 331–338, 2004.

[6] D. Cai, Z. Shao, X. He, X. Yan, and J. Han. Community mining from multi-relational networks. In *Proc. of European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2005.

[7] D. Cartwright and F. Harary. Structural balance: a generalization of heider's theory. *Psychological Review*, 63(5):277–93, 1956.

[8] I. Dasgupta. 'living' wage, class conflict and ethnic strife. *Journal of Economic Behavior & Organization*, 2009. in press.

[9] I. Dasgupta and R. Kanbur. Community and class antagonism. *Journal of Public Economics*, 91(9):1816–1842, Sep 2007.

[10] J. Denrell. Why most people disapprove of me: Experience sampling in impression formation. *Psychological Review*, 112(4):951–978, 2005.

[11] B. Ding, D. Lo, J. Han, and S.-C. Khoo. Efficient mining of closed repetitive gapped subsequences from a sequence database. In *Proc. of the IEEE International Conference on Data Engineering*, 2009.

[12] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World.* Cambridge Unversity Press, 2010.

[13] G. Flake, S. Lawrence, C. Giles, and F. Coetzee. Self-organization and identification of web communities. *Computer*, 35(3):66–71, 2002.

[14] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Proc. of ACM conference on Hypertext and hypermedia*, 1998.

[15] M. Giles and A. Evans. The power approach to intergroup hostility. *The Journal of Conflict Resolution*, 30(3):469–486, Sep 1986.

[16] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, June 2002.

[17] A. Y. R. González, J. F. M. Trinidad, J. A. Carrasco-Ochoa, and J. Ruiz-Shulcloper. Rp-miner: a relaxed prune algorithm for frequent similar pattern mining. *Knowl. Inf. Syst.*, 27(3):451–471, 2011.

[18]F. Harary. On the notion of balance of a signed graph. *Michigan Math. Journal*, 2(2):143–146, 1953.
[19]Y. Jia, J. Zhang, and J. Huan. An efficient graph-mining method for complicated and noisy data with real-world applications. *Knowl. Inf. Syst.*, 28(2):423–447, 2011.
[20]S. Labovitz and R. Hagedorn. A structural-behavioral theory of intergroup antagonism. *Social Forces*, 53(3):444–448, Mar 1975.
[21]E. A. Leicht and M. E. J. Newman. Community structure in directed networks. *Physics Review Letter*, 100(11):118703, Mar 2008.
[22]J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *Proc. of International Conference on World Wide Web*, pages 641–650, 2010.
[23]H.-F. Li, H.-Y. Huang, and S.-Y. Lee. Fast and memory efficient mining of high-utility itemsets from data streams: with and without negative item profits. *Knowl. Inf. Syst.*, 28(3):495–522, 2011.
[24]H. Liu, Y. Lin, and J. Han. Methods for mining frequent items in data streams: an overview. *Knowl. Inf. Syst.*, 26(1):1–30, 2011.
[25]D. Lo, S.-C. Khoo, and J. Li. Mining and ranking generators of sequential patterns. In *Proc. of SIAM International Conference on Data Mining*, 2008.
[26]D. Lo, S.-C. Khoo, and C. Liu. Efficient mining of recurrent rules from a sequence database. In *DASFAA*, pages 67–83, 2008.
[27]D. Lo, D. Surian, K. Zhang, and E.-P. Lim. Mining direct antagonistic communities in explicit trust networks. In *Proc. of ACM International Conference on Information and Knowledge Management*, 2011.
[28]M. McPerson, L. Smith-Lovin, and J. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(3):415–444, 2001.
[29]F. Moerchen, M. Thies, and A. Ultsch. Efficient mining of all margin-closed itemsets with applications in temporal knowledge discovery and classification by compression. *Knowl. Inf. Syst.*, 29(1):55–80, 2011.
[30]M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physics Review*, E(69), 2004.
[31]N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. of International Conference on Database Theory*, pages 398–416, 1999.
[32]S. H. Strogatz. Exploring complex networks. *Nature*, 410(6825):268–276, March 2001.
[33]Tolsma, Jochem, N. D. Graaf, and L. Quillian. Does intergenerational social mobility affect antagonistic attitudes toward ethnic minorities? *British Journal of Sociology*, 60(2):257–277, June 2009.
[34]V. A. Traag and J. Bruggeman. Community detection in networks with positive and negative links. *Physical Review E*, 80(3):036115, Sep 2009.
[35]J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In *Proc. of the IEEE International Conference on Data Engineering*, 2004.
[36]D. R. White, F. Harary, M. Sobel, and M. Becker. The cohesiveness of blocks in social networks: Node connectivity and conditional density. *Sociological Methodology*, 2001.
[37]X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc. of IEEE International Conference on Data Mining*, 2002.
[38]B. Yang, W. Cheung, and J. Liu. Community mining from signed social networks. *IEEE Trans. on Knowledge and Data Engineering*, 19(10):1333–1348, 2007.
[39]M. J. Zaki and C.-J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SDM*, 2002.
[40]K. Zhang, D. Lo, and E.-P. Lim. Mining antagonistic communities from social networks. In *Proc. of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2010.

Author Biographies

Kuan Zhang received Master degree from Singapore Management University. He is interested in mining antagonistic sub-communities, assocation rule mining, data clustering, classification, social network trust predicting, collaborative filtering, text classification and natural language processing.

David Lo is an assistant professor in the School of Information Systems at Singapore Management University. His research interests include dynamic program analysis, specification mining, frequent pattern mining, discriminative pattern mining, and social network mining. He received a PhD in computer science from the National University of Singapore. He is a member of the IEEE and the ACM.

Ee-Peng Lim is currently a professor at the School of Information Systems of the Singapore Management University (SMU). He received Ph.D. from the University of Minnesota, Minneapolis. His research interests include social network/web mining, information integration, and digital libraries. He is currently an Associate Editor of the ACM Transactions on Information Systems (TOIS), Journal of Web Engineering (JWE), IEEE Intelligent Systems, International Journal of Digital Libraries (IJDL) and International Journal of Data Warehousing and Mining (IJDWM). He is a member of the ACM Publications Board, and is a member of the Steering Committees of the Pacific Asia Conference on Knowledge Discovery and Data Mining (PAKDD) and International Conference on Asian Digital Libraries (ICADL).

Philips Kokoh Prasetyo is a research engineer at Living Analytics Research Centre in the School of Information Systems, Singapore Management University. He received Master degree in computer science from National Cheng Kung University, Taiwan. His research interests include data mining, machine learning, natural language processing, information integration, and social network analysis.