2010

Mining Antagonistic Communities From Social Networks

Kuan ZHANG
Singapore Management University, kuan.zhang.2008@smu.edu.sg

Mining Antagonistic Communities from Social Networks

By

Zhang Kuan

Submitted to School of Information Systems in
Partial Fulfillment of The Requirements for the Degree
of Master of Science in Information Systems

Supervisor: Professor **Lim Ee-Peng**

Singapore Management University

2010

Mining Antagonistic Communities from Social Networks

Zhang Kuan

Abstract

In this thesis, we examine the problem of mining antagonistic communities from social networks. In social networks, people with opposite opinions normally behave differently and form sub-communities each of which containing people sharing some common behaviors. In one scenario, people with opposite opinions show differences in their views on a set of items. Another scenario is people explicitly expressing whom they agree with, like or trust as well as whom they disagree with, dislike or distrust. We defined the indirect and direct antagonistic groups based on the two scenarios. We have developed algorithms to mine the two types of antagonistic groups. For indirect antagonistic group mining, our algorithm explores the search space of all the possible antagonistic groups starting from antagonistic groups of size two, followed by searching antagonistic groups of larger sizes. We have also developed a divide and conquer strategy to ensure our algorithm runs on large databases. We have conducted experiments on both synthetic datasets and real datasets. The results show that our approach can efficiently compute indirect antagonistic groups. Our experiments on the four real datasets show the utility of our work in extracting interesting information from real datasets. For direct antagonistic group mining, we have combined several existing algorithm blocks to mine the patterns. We found significant differences in behaviors of users showing antagonistic relationships and those showing friendship relationships.

Contents

Acknowledgments

I wish to express my great gratitude to my supervisor, Professor Lim Ee-Peng. He constantly generated new ideas for my research project. He guided me to think and solve problems. He usually answered my doubts patiently, even in his very busy schedule. During the thesis writing, he has helped me to improve the thesis content organization and writing. Without his help, I could not have completed the thesis of this quality.

I also want to express my great gratitude to Professor David Lo. I would like to thank him for collaborating and advising me throughout the project. Without that, I would not have gone so far into investigating direct antagonistic group mining. His kindness, patience, diligence and intelligence greatly inspire me.

Finally, I would thank my family members for their support, both financially and mentally. They have been showing me with much care and encouragement all the way. Without their selfless support, I would not have come this far.

Chapter 1

Introduction

1.1 Motivation and Background

People form opinions based on their experience and perception. With different experience and perception, people are likely to hold different or even opposite opinions on the same issues. We call two groups of people holding opposite opinions on some issues an indirect antagonistic group. Indirect antagonistic groups (indirect a-group for short) are common, especially on issues like politics, religions and military.

Other than expressing opinions, people form positive (trust, agree with, etc.) and negative (distrust, disagree with, etc.) relationships with one another. Such relationships divide people into groups of two sub-communities each. Within a sub-community, people are positive to each other, and across sub-communities, people are negative to each other. We call the group formed by two such sub-communities direct antagonistic group (direct a-group for short).

Both direct and indirect a-groups (we call these two types of antagonistic groups in general as antagonistic communities) and their characteristics have been studied in sociology [40, 16, 15, 28, 20, 18, 27, 32]. Research conducted on such communities ranges from how the antagonistic groups are formed to how the conflicts in antagonistic groups affect members' performance.

Both the two types of antagonistic groups may cause conflicts. If not dealt properly, these conflicts may become more severe, or even out of control. It is therefore important to detect such groups and study them. If we can detect antagonistic communities early, we can take measures to avert the tension so as to avoid possible conflict situations. If we know why the antagonistic communities are formed, we can try to change the causal factors such that the antagonistic communities will not be formed. We can also study how the antagonistic communities grow over time so as to control their growth, or study when the antagonistic communities stop being antagonistic, etc.

Other than the social aspect, studies on antagonistic communities bring benefits to other domains as well. In business, information on antagonistic communities can be used for designing better marketing/product survey strategies to have a deeper understanding of people's preference. At product review sites, we can also use antagonistic community information to recommend or to find preferred reviewers assuming that reviews from opposing group members can be ignored.

With the advent of Web 2.0, people's opinion are readily expressed on the web. Many forums and blogs allow people to post their opinions on issues and to express relationships with other people. Some online systems like Amazon [1] and

Epinions [3] also provide system features for people to vote items, or to express their relationships (trust or distrust) with other users. These provide a wealth of data ready to be analyzed.

1.2 Objectives and Contributions

In our work, we aim to identify the antagonistic communities from networks of people and study the properties of antagonistic communities. Our work covers both direct and indirect a-groups.

The main objectives and contributions are listed below:

1. Definition of the indirect antagonistic group and its properties.

   We consider a database of users voting on items. We formally define indirect a-group based on number of commonly voted items and number of oppositely voted items (antagonism level) among two user sub-groups. We have introduced a formal definition of such an indirect a-group and studied its property. More specifically, we formulated and proved the Apriori property of indirect a-groups.

2. Indirect antagonistic group mining algorithm design.

   The mining of indirect a-groups is next on our research objective. In this aspect, we have designed an Apriori-like algorithm *Clagmine* (Algorithm 2) to mine indirect a-groups from a database which contains users' votes on items. Our algorithm explores the search space in a breadth first way and it employs the Apriori property to prune the search space. We only report

the closed patterns (closed pattern refers to maximum pattern, details are in Chapter 3). To ensure that our algorithm can cope with large databases and low support threshold, a extended version of our algorithm is introduced.

3. Evaluation of indirect antagonistic group mining algorithm.

   To evaluate the efficiency and effectiveness of our algorithm, we conducted experiments on both synthetic datasets and real datasets. We developed a synthetic data generator similar to the IBM data generator in [9]. Our data generator generates data of different scales using different parameter settings. The results show that our algorithm is able to run well on various data settings. In our experiments on four real datasets, our algorithm can effectively mine all the indirect a-groups. We have also made comparison between items voted by indirect a-groups and those not voted by indirect a-groups. The results show the two item sets differ significantly. Further investigation on the reasons of such differences has been carried out.

4. Definition of direct antagonistic group and its properties.

   In this thesis work, we also study direct a-group and their properties. We have introduced a direct a-group definition based on the well known concepts of strongly connected component and bi-partite graph. The membership property of the direct a-group is proved.

5. Direct antagonistic group mining algorithm design.

   For our proposed direct a-group definition, we have developed a mining algorithm, *Dagmine* (Algorithm 10), to mine the direct a-groups in a positive-negative networks. The algorithm adopts some existing algorithm compo-

nents including Tarjan's algorithm [39] and a bi-clique mining algorithm [31].

6. Evaluation of direct antagonistic group mining algorithm.

   We have conducted experiments to evaluate the proposed direct a-group mining algorithm on both synthetic datasets and real datasets. Detailed experimental results and analysis are available in [5]. In this thesis, we show the experiments on Epinions dataset. We extracted more than 500 direct a-groups from this dataset within a short running time. We have also compared users within one sub-communities (allied pairs) and across sub-communities (opposing pairs). We found members of the two types of pairs behave differently towards one another.

1.3 Report Outlines

This report contains six chapters. Chapter 2 presents the related work. Readers will find the priori work related to the project in this chapter. Chapter 3 presents our work on mining indirect a-groups from social networks. The definition and property are given first. It then explains our algorithm in detail. It also gives some analysis and validation of the algorithm. Next, a variation of the algorithm is introduced. Chapter 4 presents the experimental results and analysis. It mainly shows our performance study and effectiveness study on the synthetic datasets and the real datasets. Chapter 5 introduces direct a-group mining. Definition and property are given first. The algorithm is then introduced and the experiment results are shown. Chapter 6 concludes the report and discusses the future work.

Chapter 2

Related Work

In a network, communities refer to clusters where intra-cluster nodes are strongly connected and inter-cluster nodes are weakly connected. Community finding is a key problem in social network analysis and it has been extensively studied [21, 45, 26, 34, 29, 41, 47, 13]. This chapter will therefore review the previous community finding research for different types of networks in Section 2.1. Unlike the previous work, this thesis focuses on mining antagonistic communities from networks with positive and negative links. Community finding for networks with both positive and negative links is a relatively less explored research topic. We shall review works on such networks in Section 2.2. In Section 2.3, an overview of frequent itemset mining is given as the mining technique is relevant to our proposed antagonistic community mining algorithms. Our proposed algorithm is based on the classical frequent itemset mining algorithm known as Apriori [9].

2.1 Community Finding in Social Network

Communities are common in networks. Girvan *et al.* pointed out that besides the three properties of network ("small world property"[17, 25], "power law distribution of degree property"[38, 10] and "network transitivity property"[44, 33]), the community structure property (nodes in a network tend to form communities) is also an important property for many networks [21]. In social networks, this property is even more salient. Network of trust and distrust can form sub-communities where people within a sub-community trust each other but people from different sub-communities distrust each other. Trophic network can form communities where species in the same community dwell in similar environment and form a small ecological sub-system but species in different communities have little trophic relationship. Similarly, most of the time, communities in co-authorship network reflect grouping of people of different working fields.

Below, we will examine some works in community finding in networks. We will give some brief summaries of the works and show their contributions. They are related or partially related to our work. For some of them, we will give some comparison of their works to ours and discuss what is valuable of their works to our work.

2.1.1 Community Finding in Unsigned Undirected Network

The traditional way of detecting community structure in a network is hierarchical [25, 36, 37]. A weight is associated with each pair of nodes in the network, which represents how closely the two nodes are connected. One begins by adding a link between the pair of nodes with the largest weight. We add links in decreasing order of the weights. Nodes can then be connected in this way. A minimum threshold on the weights can be defined to restrict the links to be added. By varying the threshold, we can get communities at any level. Various link weighting schemes have been proposed. In [45], the number of node-independent paths between two nodes was proposed to be used as link weight. In [26], sum over weighted path length of all path (not only the node-independent paths) was used. Both these weighting schemes give good results when the networks have no isolated nodes.

The traditional methods suffer from the isolated nodes problem. When there is a single node connected to a community with a small link weight, this single node should be part of the community. Using the traditional method, the link weight between this nodes and the other nodes in the community may however be low so that the node is marked as isolated and does not belong to any community. To address this problem, Girvan *et al.*[21] introduced a new algorithm to mine communities from networks. Different from the traditional methods, the algorithm starts from the boundaries of communities. The basic principle is that for two different communities, the links connecting them must be few and the shortest paths between any two nodes from the two different communities must

pass through such links. These links carry high "betweenness". The algorithm therefore removes links with highest betweenness iteratively and re-calculates the betweenness of every remaining link after each iteration. This algorithm greatly increases the precision of community finding. However, the time complexity of the algorithm is in the order of $O(n^3)$ on sparse networks and $O(m^2n)$ in worst-cast time where m and n are the number of links and nodes respectively. Such high time complexity makes it impractical to mine large graphs.

To improve the time complexity, Newman [34] proposed a modularity based algorithm. He defined modularity as an objective function Q to evaluate the quality of a community partition. He then introduced a greedy based approximation algorithm to find the partition that gives the maximum Q value. The algorithm runs in $O((m+n)n)$ or $O(n^2)$ time for sparse graphs.

2.1.2 Community Finding in Unsigned Directed Network

The community finding methods mentioned above work for unsigned undirected networks. They however fail in directed networks. For example, if communities are "star" structures (star here refers to a directed sub-graph where one center nodes pointing out to a set of outer nodes), using traditional method, two outer nodes of the structure may have very low weight and they will not be identified as in one community (shown in Figure 2.1). Girvan's method also fails to identify the star community, because the betweenness of the links which are borders between the star community and other communities would be very low due to the reason that the outer nodes do not contribute to the betweenness of border links (shown

in Figure 2.2).

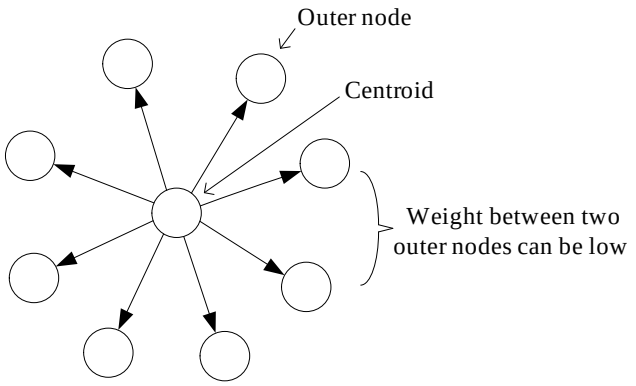


Figure 2.1: Star Community.

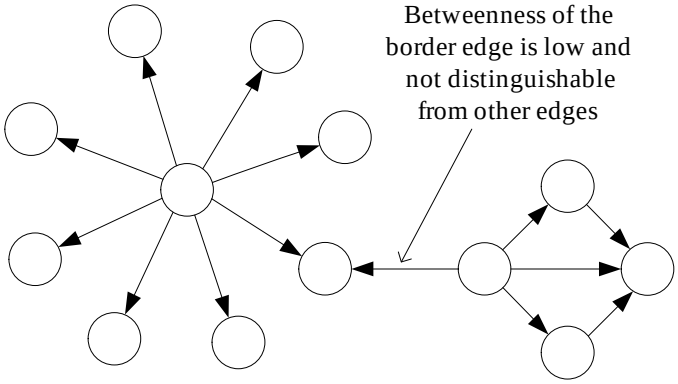


Figure 2.2: Betweenness of Border Link.

How to mine communities from directed networks? Ignoring the direction of the graph and apply the algorithm of undirected graph is one solution, but in this case, we may loose the information provided by the link direction. Leicht [29] proposed a method to find community in directed graph using the link direction information. Based on Newman's work [34], an objective function Q is defined as the fraction of links within communities subtracted by the expected fraction of such links. The directions of the links are considered when computing the expected fraction of links within a community. To optimize Q, he developed an algorithm based on the eigenvectors of the corresponding modularity matrix [29]. The author found his algorithm finding more reasonable communities than previous works due to the consideration of the link direction information.

2.1.3 Community Finding in Signed Network

Signed networks refer to networks with both positive and negative links. Mining communities from signed networks is applicable to friend and foe networks, trust and distrust networks and others.

Previous works on unsigned networks only consider one issue when dividing the networks into communities, namely density. The link density within community should be as dense as possible and the link density between community should be as sparse as possible. For signed networks, we need to take signs of links into consideration as well. The basic criteria is that for positive links, the requirement on density is the same as that for unsigned network, but for negative links, the requirement on density is the reverse of that for positive links.

Traag [41] proposed his solution on mining communities from signed network based on Newman's work [34]. He proposed his "Hamiltonian"(H function) similar as Q function above. H function is basically a function measure rewarding internal positive links, penalizing absent internal positive links, penalizing internal negative links and rewarding absent negative internal links. The author used some approximation algorithm to optimize the H function. Their experiment showed their mined results agree with analyzed results on some dataset. The problem with this work is that they only consider the signs of the links when doing partition. The density of the links is neglected. This may result in improper communities with low density.

Yang *et al.*'s work [47] considers both density and signs when partitioning the

network. 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. The algorithm first calculates the final transition probability of each node to a sink node after l steps. In this step, they treat the negative links as no link. In the second step, they consider the density of both positive links and negative links in the adjacency matrix and perform a cut to get two communities. They recursively apply the algorithm on the sub-matrix not containing the sink node to get other communities. Their algorithm runs in $O(l(m+n))$, where l is number of iteration. Compared with algorithms mentioned previously, this algorithm is very efficient. They have also proved their effectiveness through experiments on real datasets. However, when cutting the adjacency matrix, their algorithm is more biased on sign of links, less on the density. In addition, their algorithm may not work well on directed graphs. For example, if started from an outer node, the algorithm would fail to identify star communities.

2.1.4 Community Finding in Other Types of Network

Cai *et al.* [13] focused on mining communities from multi-relational social networks (multiple networks on the same set of nodes). They first formed the target relationship network (partial information of the hidden relationship network which is inferred from the multi-relational networks) from the labeled nodes (nodes in the same community have the same label). Next, they combined the original heterogeneous networks to approximate the target relationship networks. They

defined an objective function based the L_2 norm of the difference matrix between the target relationship matrix and weighted combination of the heterogeneous networks. The problem of approximating the target network is then equivalent to calculating the weights which minimize the objective function. After the weights were obtained, they took the weighted sum of the multi-relational networks to get the hidden relationship network. They mined communities from this hidden relationship network using the community mining algorithm *threshold cut*. The innovative part of this paper is that they mine communities from the hidden relationship network instead of different heterogenous networks. The mined result would be less biased to a particular relationship network.

Different from their work, our work mainly deals with homogeneous relationship networks. In our indirect a-group mining problem, we consider mainly with voting relationships. In our direct a-group mining problem, we consider mainly with trust/friend and distrust/foe relationships. It can be seen as different weights (e.g. trust is 1, distrust is -1) on a same type of link, we may introduce techniques in [13] in our future work to combine a set of similar natured but heterogenous networks to ensure results less biased to a particular network.

2.2 Positive and Negative Relationships in Social Network

Social scientists recognize that both positive and negative relationships can be part of social networks, and call them signed networks. For many years, they have

studied this kind of mixed relationship networks and developed theories around such signed networks. We review some of them in this section.

2.2.1 Balance Theorem and Its Extensions

In [19], 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 [14, 23] 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 (refer to Figure 2.3).

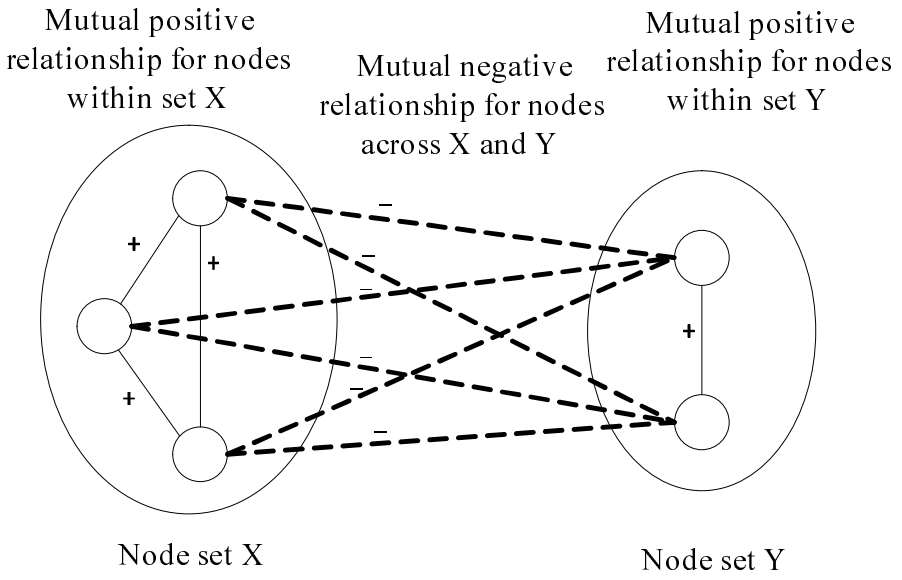


Figure 2.3: Network with Structure Balance.

Two extensions of balance theorem have been studied in [19]. The first extension addresses balance structure of non-complete networks. Easley and Kleinberg

defined a signed non-complete network to be balanced if it is possible to divide the nodes into to two camps where within each camp, the links are positive and across the two camp, the links are negative. A breadth-first based algorithm was introduced to judge whether an non-complete network is balanced. The second extension is about approximately balanced networks with only most of its triangles balanced [19]. It relates the percentage of balanced triangles, percentage of positive links within community and percentage of negative links across communities as follows.

Let ε be any number such that $0 \leq \varepsilon < \frac{1}{8}$, and define $\delta = \sqrt[3]{\varepsilon}$. If at least $1 - \varepsilon$ of all triangles in a labeled complete graph are balanced, then either

(a) there is a set consisting of at least $1 - \delta$ of the nodes in which at least $1 - \delta$ of all pairs are friends, or

(b) the nodes can be divided into two groups, X and Y, such that

   (i) at least $1 - \delta$ of the pairs in X like each other,

   (ii) at least $1 - \delta$ of the pairs in Y like each other, and

   (iii) at least $1 - \delta$ of the pairs with one end in X and the other end in Y are enemies [19].

Our work deals mainly on mining indirect and direct a-groups. Balance theorem and its extensions explore the possibility of partitioning the networks into two antagonistic sub-communities. There are both some similarities and differences between our work and works in balance theorem. The differences between our indirect a-group mining and works on balance theorem are:

- Our indirect a-group mining mainly deals with networks with two kinds of nodes, the user nodes and item nodes. The user nodes have only outlinks, while the item nodes only have inlinks. In the balance theorem, they deal with networks with the same types of nodes. Each nodes can have both inlinks and outlinks.

- In our work, different indirect a-groups can have overlapping user nodes, while works on balance theorem divide nodes into non-overlapping sub-communities.

Despite the differences, our networks contain similar information as the ones dealt by balance theorem. Considering our indirect a-group, for users voting a set of items with similar votes (users within a sub-community), we can consider they have mutual undirected positive links. For users voting a set of items with opposite votes (users across sub-communities), we can consider they have mutual undirected negative links. Hence, the indirect a-groups mined by us and the antagonistic groups in balance theorem are similar because they both have mainly positive links within sub-communities and negative links cross sub-communities.

In the case of direct a-group mining, we deal with directed networks. Each sub-community is expected to form strongly connected component. Balance theorem however deals with undirected networks, nodes forming groups of positively connected components.

Though our direct a-groups and the groups found by balance theorem seem different, they can be defined uniformly. For these two types of groups, they

need to have the following two conditions fulfilled. Firstly, the links across sub-communities are mutual negative. Secondly, for any two nodes within the same sub-community, there is a path with only positive links connecting them. In our direct a-group, the path is directed, while in balance theorem's group, the path is undirected. Hence, we can say our direct a-group is a stronger definition than that of the groups found by balance theorem and its extensions.

2.2.2 Recent Works on Positive and Negative Relationships in Social Networks

Leskovec *et al.* [30] worked on predicting the polarity of a known link. He proposed a logistic regression classifier using two classes of features. The first class of features is based on indegree, outdegree and their combinations (taking the sign of the links into consideration). The second class of features is based on "triad", involving the target link (local features). The authors compared the classification results with the results from two social-psychological theories: balance theory and status theory. They found that the two results agree with each other to a large extent. They have also investigated the global structure of signed networks. They found that the three experimented datasets have structures matching well with the structure suggested by status theory. Finally, they showed that the importance of negative links in predicting positive links. Their work can predict the link sign in a high accuracy. It can be utilized to estimate the sentiment between nodes. Combined with link prediction methods, this work can be used to make up missing links for signed networks.

Bonachich and Lloyd proposed [11] to use eigenvector to measure the centrality or status of each node. 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.

If there are two communities in the network, all the eigenvector values of the first community will be positive and the rest negative. The shortcomings of this method are that it requires the network to be divisible into two communities and it cannot handle networks with more than two communities. Although the authors have tested on a real dataset and got experiment result that matches the real situation, the real dataset has only of 18 nodes. The scalability of their method is unknown.

2.3 Frequent Itemset Mining

Our algorithm of mining antagonistic communities from social networks has similar nature of frequent itemset mining algorithms. First, let us introduce a concept closely related to frequent itemset mining: association rule mining. Since its first introduction in 1993 [8], association rule mining has attracted much attention. The basic problem of association rule mining is: let $\mathcal{D}$ be a set of transactions. A transaction is a collection of items. Let X be an itemset and Y be another itemset. We say a transaction T contains X if all the items in X appear in T,

denoted by $X \subseteq T$. We say rule $X \Rightarrow Y$ holds if (a) at least $s\%$ of $\mathcal{D}$ contain $X \cup Y$, and (b) at least $c\%$ of transactions containing X also contains Y. The association rule mining problem is to find all such rules.

For this problem, the common solution is to use the first criteria to find all $X \cup Y$ appearing at least $s\%$ times in $\mathcal{D}$ (frequent itemset mining), and then do a further checking on the second criteria. Much frequent itemset mining research has been conducted [22, 9, 8, 24, 48, 43, 35, 42]. They can be broadly divided into two categories: breadth first (BF) and depth first (DF) based. The earlier works are all BF based [9, 8, 24]. The classical and well cited "Apriori" algorithm belongs to this category. A representative DF based method is Han's FP-growth algorithm [22]. We will examine them in subsequent sections.

2.3.1 Breadth First Based Approach

The general framework of BF based method is to first scan the database and get the frequent size-1 itemsets. This serves as a basis for subsequent processing. After getting the size-k frequent itemsets, we extend the size-k itemsets to size-$(k+1)$ itemsets and scan the database again to get the frequent size-$(k+1)$ itemsets. We continue this process until no more frequent itemsets can be generated. The search space of the frequent itemsets forms a lattice [49]. This method is like exploring the lattice in a breadth first way. It generates frequent itemsets level by level. Some of the earliest works in this category are AIS and SETM algorithm [8, 24].

The AIS and SETM Algorithm

AIS [8] and SETM [24] generate candidates as the database is scanned, which is so called "candidates are generated on the fly". This makes the number of candidate generating operation scales up with the database size. In addition, due to their low efficiency in candidate generation, they generate many candidates with small support, which turn out to be invalid frequent itemsets. This wastes the time to process them and wastes space to store them. Apriori algorithm improves on these two aspects by adopting the "Apriori property".

The Apriori Algorithm

Apriori Property Before introducing the framework of the Apriori algorithm [9], let us first look into an important Apriori property of frequent itemset: All subsets of a frequent itemset are frequent. There are two implications of this property. Firstly, we can improve the candidate generation process by generating candidate frequent set of size-k from that of size-$(k-1)$. Secondly, if any subset of an itemset is not frequent, this itemset is not frequent. This can be used in pruning the candidates. Adopting this property greatly improves the efficiency of the algorithm.

Algorithm Framework The Apriori algorithm is shown in Algorithm 1. It follows the framework of BF method we mentioned earlier. In the first pass, it generates all the frequent size-1 itemsets (line 1). Now, let us consider we have get all the size-$(k-1)$ frequent itemsets. In the kth pass, we first generate the

size-k candidate frequent itemsets from size-$(k-1)$ frequent itemsets with the *apriori-gen* method (line 3). Next, we scan the database once to update the count of each candidate (line 4-9). Following that, we decide which candidates are frequent (line 10). We repeat the process until no more frequent itemsets can be generated. Finally, we output frequent itemsets of all sizes (line 12).

Input: minimum support:*minsup*; rating database
Output: frequent itemset of all size
1 L_1 = {large 1-itemsets};
2 **for** $k = 2; L_{k-1} \neq \emptyset$; k++ **do**
3 C_k=apriori-gen(L_{k-1});
4 **forall** *transaction* $t \in \mathcal{D}$ **do**
5 C_t=subset(C_k,t);
6 **forall** *candidates* c *in* C_t **do**
7 c.count++;
8 **end**
9 **end**
10 L_k=$\{c \in C_k | c.\text{count} \geq \text{minsup}\}$;
11 **end**
12 Answer=$\bigcup_k L_k$

Algorithm 1: Apriori Algorithm

Apriori Candidate Generation The *apriori-gen* function takes L_{k-1} as input. It first joins the elements in L_{k-1}. The principle of joining two frequent itemsets p and q of size-$(k-1)$ is as follows: p and q are joined if $p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$, and the joined result is itemset { $p.item_1$, $p.item_2$, ..., $p.item_{k-1}$, $q.item_{k-1}$ }. After we get all the joined result, a *prune* step is performed. A candidate c in C_k is deleted if any of its $(k-1)$-subset is not in L_{k-1}.

The improvement of Apriori algorithm arises from the following points. Firstly, the candidates are not generated on the fly. For each database scan, it requires only one generation of the candidates. This saves time greatly. Secondly, Apriori

algorithm adopts the Apriori property to prune the candidates in candidate generation process. This saves storage and time. Thirdly, as the candidates are built into hashtree structure, the subset testing time is shortened to the traversing time of the hashtree, which is short as the tree depth is only of the candidate size (in 10s).

The AprioriTid and AprioriHybrid Algorithm

However, Apriori algorithm has its shortcomings, which is also the shortcomings of all the BF based approach. First, the candidate generation process can be intolerably long when the support is low, especially for the initial several iteration. This can be counted as a bottleneck of BF based approach. Second, it requires database scan in each iteration, which resulted in a lot of disk I/Os. This is another factor which slows the algorithm down. To address the second shortcoming, Agrawal proposed AprioriTid, a modified version of Apriori [9].

AprioriTid keeps another set $\overline{C_k}$ to keep track of the Id of the transaction containing the frequent itemsets, as well as those frequent itemsets. After the first database scan, $\overline{C_k}$ will be built and all subsequent database scan in the original Apriori algorithm is substituted by scan of $\overline{C_k}$. This method gets rid of the problem of multiple database scan, but it requires more storage space. When the $\overline{C_k}$ can fit into the memory, the AprioriTid will be faster than Apriori. However, this method suffers from the problem of "slow start". In the initial passes, this method is slower than Apriori due to $\overline{C_k}$ building process and its own containment checking scheme (checking which candidates are contained in the

transaction). However, in the later passes, AprioriTid outperforms Apriori due to small number of candidates and no database scan.

AprioriHybrid algorithm is then introduced as a hybrid of Apriori and AprioriTid algorithms. In initial passes, AprioriHybrid uses Apriori, in later passes, the algorithm switches to AprioriTid. The switching criteria is based on some heuristics.

2.3.2 Depth First Based Approach

Although the database scan can be avoided by using some data structure, the candidate generation is fundamental in BF based approach and is the bottleneck when the database is huge and support is low. Later, some works tried to address this issue by adopting the depth first based approach. One representative is the FP-growth algorithm [22].

FP-growth algorithm is based on a data structure called Frequent Pattern tree (FP-tree). The FP-tree is used to compress the database. The compression ratio can range from 20-100 depending on the overlapping ratio of the transactions [22]. The FP-tree can achieve high compression ratio when there are a lot of overlaps in transactions. This is because FP-tree adopts a tree structure similar to prefix tree. The more overlapping, the more shared nodes, hence, the larger the compression ratio. The FP-growth algorithm mines frequent patterns recursively on FP-tree. It is about an order of magnitude faster than the Apriori algorithm. For long pattern, say pattern with 100 items, instead of generating $2^{100} \approx 10^{30}$

candidates and test them one by one in the case of Apriori algorithm, FP-growth will generate all the candidates in one go and no test is required [22]. The main shortcoming of FP-growth is that it requires building FP-tree recursively, which results in large memory consumption.

Some similar works of depth first based approach are also proposed [6, 7]. In Agarwal's work [7], a tree projection based algorithm is developed. The algorithm adopts a lexicographic tree to keep the possible frequent itemsets. The transaction set is projected to reduced transaction set based on frequent itemsets mined. Their algorithm can also outperform the classical algorithms by an order of magnitude. Still, the shortcoming of this method is the high requirement of memory space to hold the projected transaction sets.

Chapter 3

Indirect Antagonistic Group Mining

In this chapter, the formal definition of indirect a-group and its properties are introduced first. Following that, our algorithms of mining indirect a-groups are presented.

3.1 Preliminaries

Indirect a-group is applicable to a network of users interacting with one another through voting on objects. The votes from the users can be binary, numeric (e.g., ratings), or textual (e.g., opinion text). To keep the indirect a-group concept general, we map the different votes to only three types of votes, positive, negative and neutral. Hence there is a bipartite graph between users and objects where the arrows are labeled with vote type. For example in Epinions and Amazon where there is a 5-point scale for ratings assigned to items by users. As $1-2$ are con-

sidered poor rating scores and they reflect the rater has negative opinions on the item, we map rating scores of $1-2$ to negative votes. $4-5$ reflect positive opinions on the items. We map them to positive votes. 3 reflects a neutral opinion. We map it to neutral. In Slashdot, each person can vote another person as friend (a positive relationship, mapping to a positive vote), or enemy (a negative relationship, mapping to a negative vote), or neutral (neutral relationship, mapping to neutral vote). Hence, votes (numeric, textual, etc.) under different schemes can be mapped into our three types of votes.

We formalize our input as a database of votes as defined in Definition 3.1.1.

Definition 3.1.1. *Consider a set of users U and a set of items I. A database of votes consists of a set of mappings of item identifiers to a set of (user,vote) pairs. There are three types of vote polarity considered: positive, negative, and neutral. The database of vote could be formally represented as:*

$$DB_R = \{it_{id} \mapsto \{(us_{id}, v)\ldots\} | it_{id} \in I \wedge us_{id} \in U \wedge v \in \{positive, negative, neutral\} \wedge us_{id} \text{ gives } it_{id} \text{ a vote of } v\}$$

We refer to the size of a vote database DB_R as $|DB_R|$ which is equal to the number of mapping entries in the database. Two votes are said to be common between two users if the votes are assigned by the two users on the same item. A set of votes is said to be common among a set of users if these votes are on a common set of items voted by the set of users.

Definition 3.1.2. ***(Opposing Group):*** *Let U_i and U_j be two disjoint sets of users. (U_i, U_j) is called an opposing group (or, o-group for short).*

The number of common votes between two sets of users U_i and U_j is known as their **count** and is denoted by $count(U_i, U_j)$. The **support** of the two user sets $support(U_i, U_j)$ is defined as $\frac{count(U_i,U_j)}{|I|}$ where I represents the set of all items.

The number of common votes between U_i and U_j that satisfy the following three conditions:

- Users from U_i share the same vote polarity p_i;
- Users from U_j share the same vote polarity p_j; and
- p_i and p_j are opposite polarities.

is called the **antagonistic count**, denoted by $antcount(U_i, U_j)$. Obviously, $antcount(U_i, U_j) \leq count(U_i, U_j)$. 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 group (U_i, U_j) to be $aconf(U_i, U_j) = \frac{antcount(U_i,U_j)}{count(U_i,U_j)}$.

Definition 3.1.3. ***(Frequent Opposing Group):*** *An opposing group (U_i, U_j) is called a frequent opposing group (or, frequent o-group for short) if $support(U_i, U_j) \geq \lambda$ and $asupport(U_i, U_j) \geq \lambda \times \sigma$ where λ is the **support threshold** ($\in (0, 1)$), and σ is the **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-groups satisfy the important property as stated below:

Property 3.1.1. ***(Apriori Property of Freq. O-group):*** *Every size $(k-1)$ opposing group (U'_i, U'_j) subsumed by a size-k frequent o-group (U_i, U_j) is a frequent o-group.*

Proof. Assume an opposing group g_{k-1} is not a frequent o-group. This 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 group, we call the resulting opposing group $g_{k-1} \cup u_k$. $g_{k-1} \cup u_k$'s count can not be more than count(g_{k-1}) and its antagonistic count can not be more than antcount(g_{k-1}). This is because the count is calculated by intersecting the g_{k-1}'s user set's vote and the u_k's vote: 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 vote and u_k's vote such that intersected vote 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-group neither. By contrapositive, we can get the property. □

Definition 3.1.4. ***(Indirect Antagonistic Group):*** *An opposing group (U_i, U_j) is an indirect antagonistic group (indirect a-group) if it is a frequent o-group and $aconf(U_i, U_j) \geq \sigma$.*

Definition 3.1.5. ***(Closed Indirect Antagonistic Group):*** *An indirect a-group (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 3.1.1. *Consider the example vote database in Table 3.1. Suppose $\lambda = 0.5$ and $\sigma = 0.5$. Both $(\{a\}, \{d\})$ and $(\{a\}, \{b, d\})$ are indirect a-groups. 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 indirect a-group and is subsumed by* $(\{a\},\{b,d\})$. *Hence,* $(\{a\},\{d\})$ *is considered as redundant. On the other hand, both* $(\{a\},\{b\})$ *and* $(\{a\},\{b,c\})$ *are closed indirect a-groups even though both* $aconf(\{a\},\{b\})$ *and* $aconf(\{a\},\{b,c\})$ *has the same value which is* $\frac{2}{3}$. *This is so as* $count(\{a\},\{b\}) \neq count(\{a\},\{b,c\})$ *and* $antcount(\{a\},\{b\}) \neq antcount(\{a\},\{b,c\})$.

Item	User votes
i_1	a-positive, b-negative, d-negative
i_2	a-positive, b-negative, d-negative
i_3	a-positive, b-positive, d-positive
i_4	a-positive, b-negative, c-negative
i_5	a-positive, b-negative, c-negative
i_6	a-positive, b-positive, c-negative

Table 3.1: Example Vote Database 1 - DB_{EX1}

Note 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. We can show this using the vote database example in Table 3.2. 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 $(count(\{d\},\{e\}) = 2) > (count(\{d\},\{e,f\}) = 1)$.

Item	User votes
i_1	a-positive, b-negative, c-negative
i_2	a-positive, b-negative, c-negative
i_3	a-positive, b-negative, c-positive
i_4	d-positive, e-negative, f-negative
i_5	d-positive, e-positive

Table 3.2: Example Vote Database 2

Definition 3.1.6. ***(Indirect Antagonistic Group Mining Problem):*** *Given a set of items I voted by a set of users U (the vote database), the indirect a-*

group mining problem is to find all closed indirect a-groups with the given support threshold λ and antagonistic confidence threshold σ.

3.2 Algorithm for Mining Indirect Antagonistic Group

We develop a new algorithm to mine indirect a-groups from a database of votes. For simplicity, we will drop the term "indirect" in the rest of this chapter. Our algorithm systematically traverses the search space of possible antagonistic groups and find for groups that are antagonistic while using a search space pruning strategy to remove unfruitful search spaces.

3.2.1 Basic Framework

The a-group mining algorithm known as *Clagmine* (Closed A-group Mining) runs for multiple passes. In the initialization pass, we calculate the *count* and *antcount* of all the size-2 a-group candidates opposing groups and determine which of them are frequent o-groups. In the next pass, with the set of frequent o-groups found in the previous pass, we generate new potential frequent o-groups, which are called *candidate* set. We then count the actual *count* and *antcount* values for these candidates. At the end of this pass, we determine the frequent o-groups from these candidates, and they are used to generate frequent o-groups for the next pass. After that, we filter the previous frequent o-group set with the newly generated frequent o-group set by removing non-closed frequent o-groups. Next,

we move on to the next pass. This process continues until no larger frequent o-groups are found. After successful mining of all frequent o-groups, we derive the a-groups from them.

Input: λ; σ; vote database
Output: valid and closed a-group of all size
1 L_1 = frequent user set;
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=antGrpMining-gen(L_{k-1});
6 **end**
7 root← buildHashTree(k,C_k);
8 **foreach** *item* $t \in \mathcal{D}$ **do**
9 C_t=subset(t,root);
10 **foreach** *candidate* c *in* C_t **do**
11 update count and antcount of c;
12 **end**
13 **end**
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);
16 **end**
17 G=$\{g \in \bigcup_k L_k | \frac{antcount(g)}{count(g)} \geq \sigma\}$;
18 **Output** G;

Algorithm 2: Closed Antagonistic Group Mining Algorithm – $Clagmine(\lambda,\sigma,DB_R)$

Algorithm 2 shows the *Clagmine* algorithm. Two basics data structures are maintained namely L_k the intermediary set of frequent o-groups of size k and C_k a candidate set for a-groups checking. The first two lines of the algorithm simply calculate size-2 candidates to get the frequent size-2 o-groups. They form the base for subsequent processing. A subsequent pass, say pass k, consists of three phases. First, at line 5, the frequent o-groups in L_{k-1} found in $k-1$ pass are used to generate the candidate frequent o-group set C_k, using the **antGrpMining-gen** method described in Algorithm 3. It is to ensure we have a minimum sized and complete set of candidates for the next pass. Next, the database is scanned and

the count and antcount of candidates in C_k are updated (line 7 to 13). We make use of the hash-tree data structure described in [9] to hold C_k and we then use a subset function to find the candidates overlap with voters of an item. After we marked all the overlapping candidates, we update the count and antcount of them. Frequent o-groups can be determined by checking count and antcount against the support threshold and asupport threshold respectively. Following that, L_{k-1} is filtered with the newly generated frequent o-groups to remove non-closed frequent o-groups (line 15). After all the passes, the a-groups are determined from the frequent o-group set(line 17). The following subheadings elaborate the various components of the mining algorithm in more detail.

3.2.2 Candidate Generation and Pruning

Input: size-$(k-1)$ frequent o-group set L_{k-1}
Output: size-k candidate frequent o-group set
1 **foreach** $p, q \in L_{k-1}$ **do**
2 $g_k \leftarrow merge(p, q)$;
3 add g_k to C_k;
4 **forall** *$(k-1)$-subsets s of g_k* **do**
5 **if** $s\neg \in L_{k-1}$ **then**
6 delete g_k from C_k;
7 **end**
8 **end**
9 **end**
10 return C_k;

Algorithm 3: antGrpMining-gen(L_{k-1})

Candidate Generation and Pruning. The **antGrpMining-gen** function described in Algorithm 3 takes as argument L_{k-1}, the set of all frequent $(k-1)$ o-groups. It returns a superset of all frequent size-k o-groups. The function works as follows. First, we *merge* all the elements in L_{k-1} that can be merged ($diff$ in

Input: frequent o-group (U_i,U_j); frequent o-group (U'_i,U'_j)
Output: merged result of the two input frequent o-groups
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**
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**
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**
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**
13 return *null*;

Algorithm 4: $merge((U_i,U_j),(U'_i,U'_j))$

the *merge* returns the number of different users in two user sets). We add them to C_k. Next, in the pruning stage, we delete all element $g_k \in C_k$ if some $(k-1)$ subset of g_k is not in L_{k-1}.

The pruning stage's correctness is guaranteed by the property 3.1.1 of frequent o-groups. From the property, if g_k is a frequent o-group, all its $(k-1)$ subsets must be frequent o=groups. In other words, if any one $(k-1)$ subset of an opposing group g_k is not a frequent o-group, g_k is not eligible to be frequent. We thus delete or prune such g_ks.

The correctness of the **antGrpMining-gen** function can be easily seen from Lemma 1 described below.

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

Proof. From property 3.1.1, any subset of a frequent o-group must also be fre-

quent. Hence, if we extend each frequent o-group 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 frequent o-groups in L_k. In the Algorithm 3, first we perform a merge process which is equivalent to extending L_{k-1} with all possible users in the database (line 2) and then at lines 4-8, we delete candidates whose $(k-1)$-subsets are not in L_{k-1}. Thus after we merge the and deletion steps, all size-k frequent o-groups must be a subset of the returned candidate set. □

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

Example 3.2.1. *Let* L_3 *be* $\{(\{u1\},\{u2,u3\}),\ (\{u5\},\{u2,u3\}),\ (\{u1,u4\},\{u2\}),\ (\{u1,u5\},\{u2\}),\ (\{u4,u5\},\{u2\})\}$. *After the merge step,* C_4 *will be* $\{(\{u1,u5\},\{u2,u3\}),\ (\{u1,u4,u5\},\{u2\})\}$. *The deletion step serving as apriori-based pruning, will delete the candidate a-group* $(\{u1,u5\},\{u2,u3\})$ *because* $(\{u1,u5\},\{u3\})$ *is not in* L_3. *We will then left with only* $\{(\{u1,u4,u5\},\{u2\})\}$ *in* C_4.

3.2.3 Subset Function

Subset Function. The candidate frequent o-groups are stored in a hashtree. The node of the hashtree contains either a hashtable (interior node), or a list of candidates (leaf). Each node also contains a user label representing the user associated with this node. The hashtable of the node refers to the next level nodes, with the hashkey being the user label of the next level nodes. The building process of the hashtree is shown in Algorithm 5. Every candidate is sorted according to

Input: k:level of the tree; C_k:size-k candidate set
Output: root of the tree
create *new* node root;
foreach *candidate* c_i *in* C_k **do**
    sort users in c_i by userID;
    tempNode← root;
    foreach *user* u *in* c_i **do**
        if *tempNode has descendant* d *labeled* u **then**
            tempNode← d;
        else
            create node d with label u;
            set d as descendant of tempNode;
            tempNode← d;
        end
        if u *is the last user in* c_i **then**
            set tempNode to leaf;
            add c_i to the leaf;
        end
    end
end
return root;

Algorithm 5: buildHashTree(k,C_k)

userID first, and then inserted into the hashtree level by level.

The subset function finds all the candidate frequent o-groups which are subsets of users of item t. The users of item t is first sorted according to userID. The users 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. For a user 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 interior or leaf. If it is an interior node, we add it to the pointer list and if it is a leaf, all the candidates stored in the leaf are marked as subset of users of t. The current node is removed from the pointer list if all its child nodes are visited. The process repeats through all the users of item t. At the end, all the candidates which are

Input: t:item in database; root:root of hashtree
Output: set of candidate contained in t

```
1  C_t ← ∅;
2  pointerRef ← empty vector of node;
3  pointerRefSuffix ← empty vector of node;
4  add root to pointerRef;
5  foreach voter u of t do
6      foreach node node_i in pointerRef do
7          if node_i has descendant d_i with label u then
8              node_i's descendant count−−;
9              if node_i's descendant count==0 then
10                 remove node_i from pointerRef;
11             end
12             if d_i is leaf then
13                 add candidates stored in d_i to C_t;
14             else
15                 add d_i to pointerRefSuffix;
16             end
17         end
18     end
19     append pointerRefSuffix to pointerRef;
20     pointerRefSuffix← empty vector of node;
21 end
```

Algorithm 6: subset(t,root)

subsets of users of t will be marked.

3.2.4 Filtering of Non-Closed Antagonistic Groups

Filtering Non-Closed A-Groups. As our a-groups are derived from frequent o-groups, we ensure our a-groups are closed by filtering out non-closed frequent o-groups (closed frequent o-groups are the ones not subsumed by any other frequent o-groups with the same count and antcount). Note that as a closed frequent o-group could potentially subsume a combinatorial number of sub-groups. Removal of non closed frequent o-groups potentially reduces the number significantly.

The filtering of non-closed frequent o-groups is guaranteed by line 15 in Algorithm 2. Its pseudo code is shown in Algorithm 7. The function works as follows. For each frequent o-group g_k in L_k, we traverse through every frequent o-group g_{k-1} in L_{k-1}. If g_k subsumes g_{k-1}, and the count and antcount of the two frequent o-groups are equal, g_{k-1} can be filtered. This step ensures all the frequent o-groups in L_{k-1} are closed. By iterating through k, we can have all the non-closed frequent o-groups of any size filtered. Only closed frequent o-groups will remain.

Input: frequent o-group set L_{k-1}; frequent o-group set L_k
Output: closed frequent o-group 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**
6 **end**
7 **end**
8 return L_{k-1};

Algorithm 7: prune(L_{k-1}, L_k)

3.2.5 Validation of the Algorithm

Correctness of the algorithm. The correctness of the algorithm is guaranteed by the following Theorems 3.2.1 and 3.2.2. The theorems guarantee that a complete set of closed a-groups will be reported by the *Clagmine* algorithm.

Theorem 3.2.1. *Mined a-group set G contains all the closed a-groups.*

Proof. Suppose we have an arbitrary closed a-group g. Hence, $\frac{count(g)}{|I|} \geq \lambda$ 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 3.1.3, g is a frequent o-group. According to lemma 1, g will be in $C_{|g|}$. g can be captured by line 5 of Algorithm 2. As g fulfills both $\frac{count(g)}{|I|} \geq \lambda$ and $\frac{antcount(g)}{|I|} \geq \lambda \times \sigma$, g will be captured by line 14 of Algorithm 2. Since g is closed, g will remain in $L_{|g|}$ after line 15 of Algorithm 2. Finally, due to $\frac{antcount(g)}{count(g)} \geq \sigma$, g will be added to G by line 17 of Algorithm 2. Hence, every closed a-group will be contained in G. □

Theorem 3.2.2. *Mined a-group set G contains only closed a-groups.*

Proof. Suppose an opposing group $g \in G$ is not an a-group. We then have $\frac{count(g)}{|I|} < \lambda$ or $\frac{antcount(g)}{count(g)} < \sigma$. From line 17 of Algorithm 2, we know that $g \in \bigcup_k L_k$, and $\frac{antcount(g)}{count(g)} \geq \sigma$. In addition, every frequent o-group g_k in $\bigcup_k L_k$ has $\frac{count(g_k)}{|I|} \geq \lambda$. Thus $\frac{count(g)}{|I|} \geq \lambda$. It contradicts our condition that $\frac{count(g)}{|I|} < \lambda$ or $\frac{antcount(g)}{count(g)} < \sigma$. Thus, g must be an a-group. Hence, G contains only a-groups. The closure property of G can be guaranteed by line 15 of Algorithm 2. Every a-group in G will be checked to filter away the non-closed ones. The filtering method will not leave any non-closed a-groups in G, by Algorithm 7. □

3.3 A Divide and Conquer Strategy

The memory required to store all candidates could be prohibitive when the data size is large or the support threshold is small. This is due to too many L_2 patterns generated in step 14 of Algorithm 2. To address this issue, we perform a divide and conquer strategy by partitioning the database, mining for each partition, and merging the results. This detailed steps of this strategy can be found in [50]. For completeness of this thesis, we include a brief description of the proposed strategy. We first state some new definitions and describe a property.

Definition 3.3.1 (***User Containment***)**.** *Consider a entry* $m = it_{id} \mapsto PairSet$ *in* DB_R*. We say that a user* u_i *is contained in the entry, denoted by* $u_i \in m$*, iff* $\exists$ (u_i, v) *where* $v \in \{positive, negative, neutral\}$ *and* (u_i, v) *is in* $PairSet$*. We also say that a user* u_i *is in an a-group* $a = (S_1, S_2)$ *iff* $(u_i \in S1 \vee u_i \in S2)$

Example 3.3.1. *To illustrate, consider the first entry itm-usr in the example vote database shown in Table 3.1. The first entry itm-usr contains users a, b and d: a* $\in$ *itm-usr, b* $\in$ *itm-usr, and d* $\in$ *itm-usr.*

Definition 3.3.2 (***Database Partition***)**.** *Consider a user* u_i *and a database of vote* DB_R*. The partition of* DB_R *with respect to user* u_i*, denoted as* $DB_R[u_i]$ *is defined as:*

$$DB_R[u_i] = \{itm\text{-}usr | u_i \in itm\text{-}usr \wedge itm\text{-}usr \in DB_R\}$$

Example 3.3.2. *To illustrate, consider the example vote database shown in Table 3.1. Projection of the database with respect to user d is shown in Table 3.3.*

Item	User votes
i_1	a-positive, b-negative, d-negative
i_2	a-positive, b-negative, d-negative
i_3	a-positive, b-positive, d-positive

Table 3.3: Projected Vote Database 1 on User d

Having given the above two definitions, we now define a lemma to support the divide and conquer mining process.

Lemma 2 (***Divide and Merge***)**.** *Consider a database of vote DB_R, support threshold λ, and confidence threshold σ. Let U_{set} be the set of users in DB_R and Cm be the shorthand of the Clagmine operation in Algorithm 2. The following is guaranteed:*

$$Cm(\lambda, \sigma, DB_R) = \quad \bigcup_{u_i \in U_{Set}} \{g | u_i \in g \wedge g \in Cm(\tfrac{\lambda \times |DB_R|}{|DB_R[u_i]|}, \sigma, DB_R[u_i])\}$$

The proof of Lemma 2 can be found in [50].

Based on Lemma 2, the algorithm to perform divide and conquer is shown as *Clagmine-partitional*(λ,σ,DB_R) in Algorithm 8. The algorithm partitions the database one item at a time and subsequently calls the original closed a-group mining algorithm defined in Algorithm 2. The theorem to guarantee that the mined result is correct and a complete set of a-groups are mined by Algorithm 8 can be found in [50].

Note that the *Clagmine-partitional* reduces memory cost but potentially increases the runtime since the database would now need to be scanned more number of times. In our experiment mentioned in chapter 4, we would always employ *Clagmine* Algorithm and employ *Clagmine-partitional* only when the former is prohibitively expensive to run.

Input: λ; σ; vote database
Output: closed a-groups of all size
1 U_{Set} = Set of all users in DB_R;
2 $G = \{\}$;
3 **foreach** $u_i \in U_{Set}$ **do**
4 $\quad G = G \cup \{ag | u_i \in ag \wedge ag \in Clagmine(\frac{\lambda \times |DB_R|}{|DB_R[u_i]|}, \sigma, DB_R[u_i])\}$;
5 **end**
6 **Output** G;

Algorithm 8: *Clagmine-partitional*(λ,σ,DB_R)

Chapter 4

Experiment Results and Analysis

In this chapter, we describe our experiments on mining indirect a-groups. We conduct experiments on synthetic datasets generated by our synthetic data generator. We then describe our experiments on four real datasets, namely, Amazon book rating, Epinions product rating, Slashdot friend/enemy vote and Wikipedia vote datasets. In this chapter, a-group refers to indirect a-group.

4.1 Experiments on Synthetic Datasets

We first evaluate the scalability of our proposed algorithms. To do so, we rely on synthetic datasets generated based on various parameter settings. We develop a synthetic data generator for this purpose. Next, the experiment results on some synthetic datasets are compared and analyzed.

4.1.1 Synthetic Data Generator

To investigate the scalability of our mining solution on various data characteristics, similar to the evaluation measure applied to mining association rules [9, 46, 42], we develop a synthetic data generator engine. The engine takes a set of user-given input parameters as shown in Table 4.4. The procedure of synthetic data generation is shown below and the detailed algorithm is shown in Algorithm 9.

$\|U\|$	Number of users
$\|I\|$	Number of items
N_G	Average size of potential closed a-groups
N_L	Number of potential closed a-groups
s	Selection probability of each user

Table 4.4: Synthetic Data Generator Input

- Generate the set of potential closed a-groups L:(lines 1-10)

  The a-groups in L have an average size of N_G. For each a-group g in L, we pick the size of g by a Poisson distribution with mean = N_G. If the size of g is larger than $|U|$, we resample another size. Let the k-th selected size be N_k. For the first a-group, we randomly select a group of users (denoted by U_{g_1}) from U and construct an a-group with equal splits of users in U_{g_1} into the opposing user sets (U_1, U_2). The a-group is then added to L. For each subsequent a-group g_i, we randomly select $N_k \cdot q$ users from the previous generated a-group denoted by g_{i-1} and $N_k \cdot (1-q)$ users from $(U - g_{i-1})$. The q value is picked from an exponential distribution with mean = 0.5. Again, if q is larger than 1, we resample. The generated g_i with equal splits of users into the opposing user sets is then added to L.

- Determine the probabilities of a-groups:(lines 11-17)

Input: $|U|$; $|I|$; N_G; N_L; s
Output: votes of users in U to items in I

$L \leftarrow \emptyset$;
$N_1 \leftarrow poisson(N_G)$;
$g_1 \leftarrow$ randomly pick N_1 users from U;
add g_1 to L;
for *i from 2 to* N_L **do**
    $N_i \leftarrow poisson(N_G)$;
    $q \leftarrow expo(0.5)$;
    $g_i \leftarrow$ {randomly pick $N_i \times q$ users from g_{i-1}}$\bigcup${randomly pick $N_i \times (1-q)$ users from (U-g_{i-1})};
    add g_i to L;
end
for *i from 1 to* N_L **do**
    $p_{g_i} \leftarrow expo(1)$;
end
$sum \leftarrow \sum_{i=1}^{N_L} p_{g_i}$;
for *i from 1 to* N_L **do**
    $p_{g_i} \leftarrow p_{g_i}/sum$;
end
foreach *item t* **do**
    $GSet_t \leftarrow \emptyset$;
    $M_t \leftarrow poisson(|U| \times s)$;
    while *number of users in* $GSet_t < M_t$ **do**
        randomly pick g_t from L and add g_t to $GSet_t$;
        if *number of users in* $GSet_t > M_t$ **then**
            retain partial users from last picked g_t such that *number of users in* $GSet_t = M_t$;
        end
    end
end
foreach *item t* **do**
    foreach g_t *in* $GSet_t$, *suppose* $g_t = (U_1, U_2)$ **do**
        forall *user u from* U_1 **do**
            vote$(u,t) \leftarrow$ *positive*;
        end
        forall *user u from* U_2 **do**
            vote$(u,t) \leftarrow$ *negative*;
        end
    end
    forall *user u in* $U - \{users\ in\ GSet_t\}$ **do**
        vote$(u,t) \leftarrow$ *neutral*;
    end
end

Algorithm 9: Synthetic Data Generation Procedure

We associate a probability p_i of picking each a-group g_i in L. The probabilities p_i's are decided using an exponential distribution with mean = 1. They are then normalized such that $\sum p_i = 1$. Hence, we have N_L different p_i's.

- Assign a-groups to each item:(lines 18-27)

  For each item t, pick a number of users M_t to be included for the item. M_t is selected using a Poisson distribution with mean equals $|U| \cdot s$ where s is a number between 0 and 1. We pick a set of a-groups $GSet_t$, a subset of L, using a biased dice with N_L sides and each side having a probability of p_i. Note that $M_t \leq$number of users in $GSet_t$. If $M_t <$ number of users in $GSet_t$, we can assign only subset of users from an a-group in $GSet_t$ to item t. The assigned a-groups should not overlap by user, which means the same user cannot exist in two a-groups in $GSet_t$ concurrently.

- Assign votes to each item:(lines 28-40)

  The assignment of vote(u, t) (user u votes on item t) can be done as follows. We only consider three types of votes, positive, negative and neutral. For an a-group (U_1, U_2) associated with item t. We assign all vote(u_p, t)'s as positive for all u_p's belonging to U_1, and all vote(u_n, t)'s as negative for all u_n's belonging to U_2. We continue the process until all the a-groups associated with item t processed, we assign neutral to vote(u_o, t)'s for all u_o's which are not in the a-groups associated with item t.

4.1.2 Performance Study on Synthetic Datasets

We now design experiments to evaluate the scalability of our proposed algorithms using several synthetic datasets. Our synthetic data generator accepts as input $|U|$ (in '000), $|I|$ (in '000), P (i.e., $|U| \times s$, the expected number of users voting an item), N_G, and N_L (in '000). We generate four datasets using the following parameter settings.

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

All the experiments have adopted confidence threshold σ=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 4.1. Figure 4.1(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 4.1(b) shows the number of a-groups found at various support thresholds. Finally in Figure 4.1(c), we plot a graph showing the number of a-groups at different sizes while keeping support threshold at 0.002.

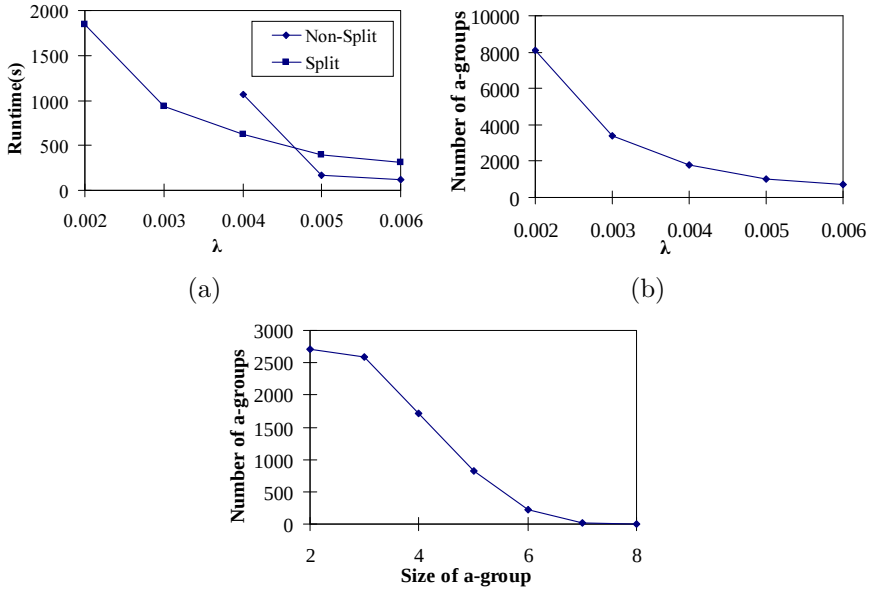


Figure 4.1: Runtime and Patterns: DS_1 at various support values.

The result shows that the runtime decreases with increasing support threshold. This decrease in runtime is accompanied by the decrease in the number of a-groups mined. Figure 4.1(c) also shows that a-groups mined have small sizes.

For DS_2, we consider a larger number of users. The results for various support thresholds with σ=0.7 are shown in Figure 4.2.

For the third dataset, we use smaller number of users and larger expected number of users voting an item. The results for various support thresholds are shown in Figure 4.3.

For the fourth dataset, we consider fewer number of items and larger number of users. The results for various support thresholds are shown in Figure 4.4.

The performance study showed that the algorithm is able to run well on various settings. The lower the support threshold the larger the runtime and number of

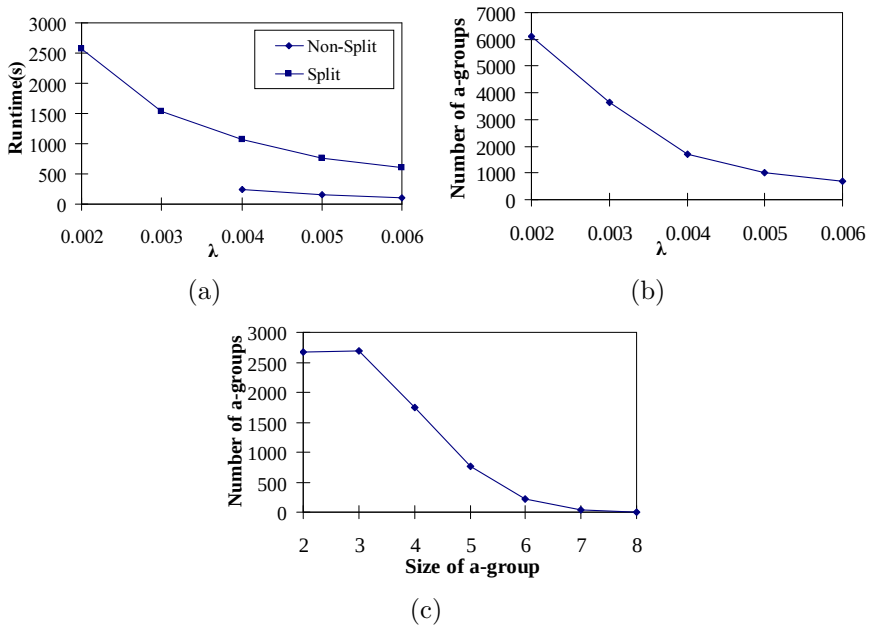


Figure 4.2: Runtime and Patterns: DS_2 at various support values.

a-groups. DS_1 and DS_2 have the same parameter settings except that the user number 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, the more time consuming it is to mine. Similarly, by comparing DS_2 and DS_4, we conclude that the larger the number of items, the more time consuming it is to mine. Comparing DS_3 and DS_1, we conclude that the larger the expected number of users voting an item, the more time consuming to it is to mine.

4.2 Experiments on Amazon Dataset

4.2.1 Performance Study on Amazon Dataset

We received the Amazon dataset from University of Illinois at Chicago. In this dataset, there are a total of 99,255 users rating 108,142 books in 935,051 reviews.

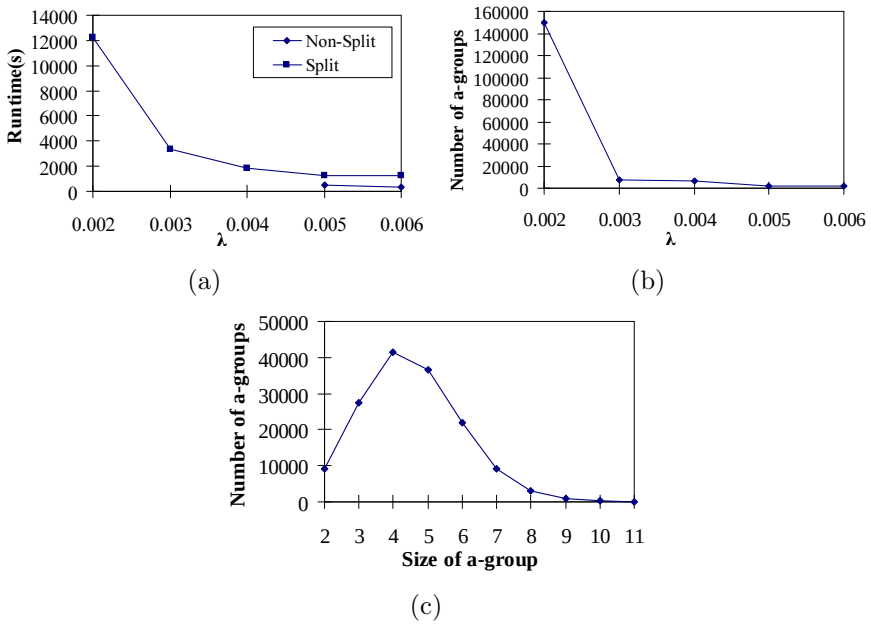


Figure 4.3: Runtime and Patterns: DS_3 at various support values.

Each review is associated with a rating issued to the item by the user. A rating to an item is represented as an inlink to the item. A rating issued by a user is represented as an outlink from the user. The rating ranges from 1 to 5. We map ratings of 4-5 to positive votes, ratings of 1-2 to negative votes, and the rest are mapped to neutral votes. Among the 935,051 ratings, 699,925 (74.9%) are positive, 108,013 (11.6%) are negative, and 104,373 (11.2%) are neutral. The indegree distribution of items and outdegree distribution of users are shown in Figure 4.5. They follow power law distribution. This agrees with the "power law degree distribution" of large networks [38, 10]. However, there are some outlying points. In Figure 4.5(a), when indegree equals 1 or 2, the number of nodes is much fewer than the power law values. Similar cases exist in Figure 4.5(b). This suggests that Amazon book rating dataset has a small number of nodes with extremely low indegree or outdegree.

The experiment for the Amazon dataset is conducted with σ=0.5 and different

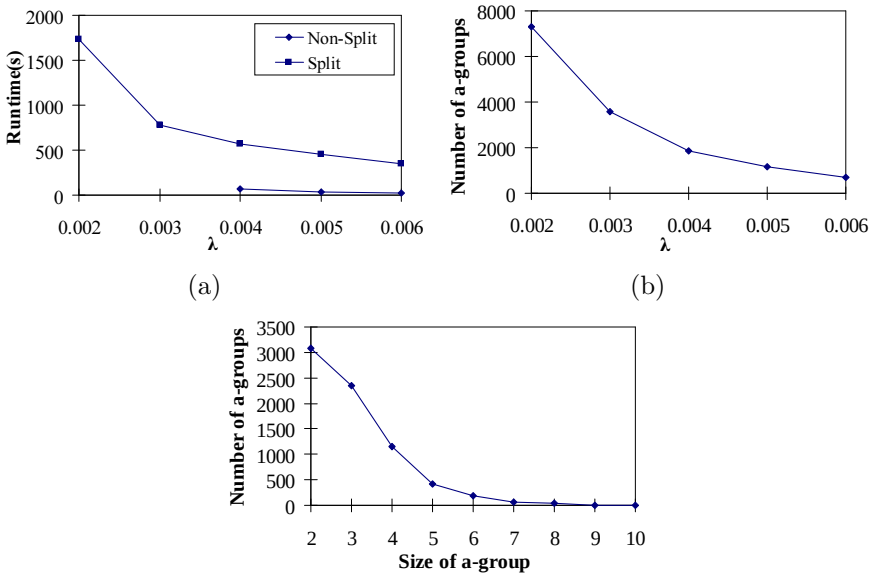


Figure 4.4: Runtime and Patterns: DS_4 at various support values.

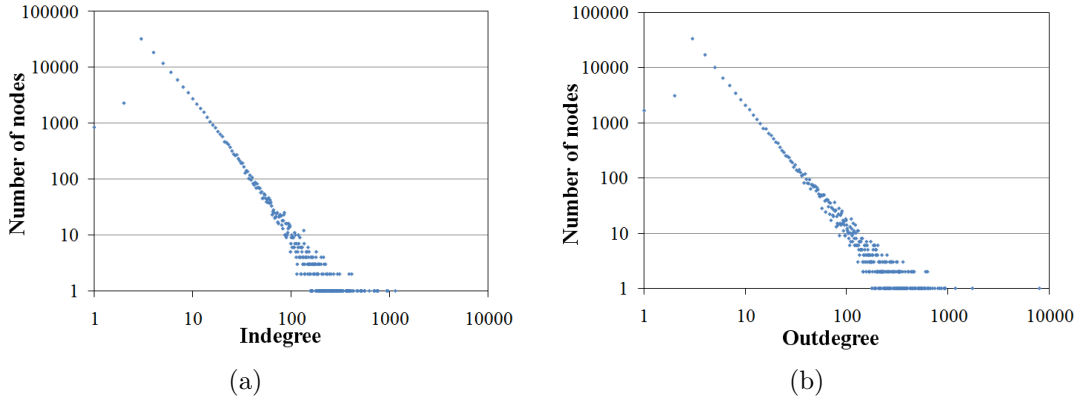


Figure 4.5: Amazon Book Ratings Dataset Indegree and Outdegree Distribution.

λ values. The results are shown in Figure 4.6. Figure 4.6(c) is obtained with absolute λ=10 (we call $\lambda \times |I|$ absolute λ).

Figure 4.6(b) shows that the number of a-groups mined is small even with low support thresholds. Most of the a-groups are of size 2. The reason could be that Amazon dataset does not contain large groups of people with opposite opinions.

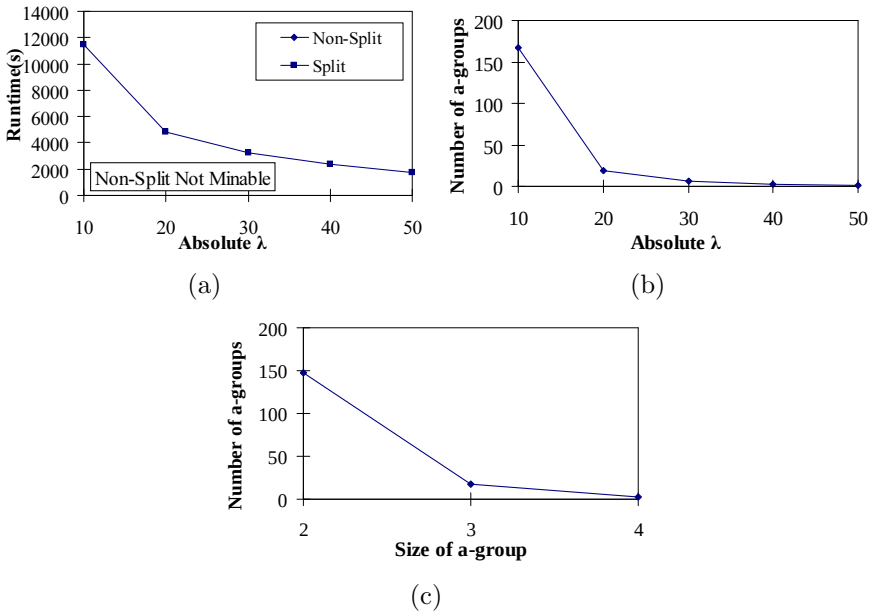


Figure 4.6: Amazon Dataset: Runtime and Distribution of A-groups.

4.2.2 Antagonistic Group Study

Several interesting a-groups are discovered from the Amazon dataset. They are obtained by running the mining algorithm on Amazon dataset with absolute λ=10 and σ=0.5. The program runs for 11469 seconds with 167 a-groups generated. 147 of the a-groups are of size 2, 18 of them are of size 3 and 2 of them are of size 4. We post-process the a-groups with the following criteria:

- Antagonistic confidence: Only retain a-groups with antagonistic confidence > 0.7. This criteria is to ensure the a-groups are of sufficient antagonism level.

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

ID	Antagonistic group	Commonly rated (Oppositely rated)	Totally rated by 1st user(% of commonly rated)	Totally rated by 2nd user(% of commonly rated)	Totally rated by 3rd user(% of commonly rated)
1	({Jason},{Luke})	12 (12)	56(21%)	13(92%)	-
2	({Jason, K.Jump},{Luke})	10 (10)	56(17.8%)	61(16.4%)	13(76.9%)
3	({Jason, C.Hill},{Luke})	10 (7)	56(17.8%)	106(9.4%)	13(76.9%)
4	({Jeffrey},{Luke})	10 (10)	137(7.3%)	13(76.9%)	-
5	({Konrad},{T.M.Sklarski})	14 (10)	452(3.1%)	22(63.6%)	-

Table 4.5: Interesting Examples from Amazon Book Rating Dataset

After postprocessing, we found five most interesting a-groups. They are presented in Table 4.5. As most of them involve Luke, we examine the first a-group in detail as follows:

- *High antagonistic level*: We observe that the two users, Luke and Jason rated items with high level of antagonism. Among Jason's 56 rated books, 12 have opposite ratings with Luke. Similarly for Luke, 12 of all his 13 rated books (about 92%) have opposite ratings with Jason. This demonstrates a significantly high level of antagonism exists in this a-group.

- *Antagonistically rated books*: Based on our mining result, we examined the Amazon website. We found the 12 books rated oppositely by Jason and Luke. The books and their ratings are shown in Table 4.6. Interestingly, Jason rated all the 12 books high while Luke gave very low ratings. There is a high tendency that the books disliked by Luke are liked by Jason.

- *Antagonistically behaved user*: It is interesting to note that Luke appears in four out of five interesting a-groups. He tends to rate books against what others rate. His ratings are opposite to other four users for at least 10 books. This very different judgement compared with others will motivate us to examine his behaviors further.

ID	Book title	Jason's rating	Luke'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

Table 4.6: Jason and Luke's Ratings on Their Commonly Rated Books

4.2.3 Comparison between Antagonistic Group Voted Items and Other Items

In this section, we compare the set of items voted by at least one a-group (called a-group voted item set) and the remaining items (called general item set). The a-groups are obtained with absolute λ=10 and σ=0.5. We adopt the item metrics for comparing a-group voted item set and general item set as follows:

1. **Positive inlink ratio**=$\frac{positive\_indegree}{positive\_indegree+negative\_indegree}$. This metric reflects the controversial level of an item. The closer the metric to 0.5, the more controversial the item.

2. **Biased inlink**=$positive\_indgree + negative\_indegree$. This metric reflects how many biased votes (positive and negative votes) an item attracts.

3. **Biased inlink ratio**=$\frac{positive\_indegree+negative\_indegree}{total\_indegree}$. This metric reflects how biased users' opinions are on an item. The larger the metric, the more biased the users' opinions .

For a given item set (which can be a-group voted item set or general item set), 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 population of a-group voted item set is different from the general item set. Considering each set has population size>30, and the standard deviations of the distributions of the two sets are unknown, we use the following equation to calculate z:

$$z = \frac{(\overline{x_1} - \overline{x_2}) - (\mu_1 - \mu_2)}{\sqrt{\frac{{s_1}^2}{n_1} + \frac{{s_2}^2}{n_2}}} \tag{1}$$

where,

$$s_k = \sqrt{\frac{1}{n-1}\sum_i (x_{ki} - \overline{x_k})^2}, (k \ is \ set \ number \ 1 \ or \ 2). \tag{2}$$

Our z-test results are shown in Table 4.7. For the two populations to be similar at 99% confidence, we need their z-value to be within [-2.57,2.57]. Table 4.7 shows that the z-values of the three metrics are all out of the interval. Hence, we can say with 99% confidence, the two populations are different with respect to the three metrics. Furthermore, we observe that:

1. In terms of positive inlink ratio, both a-group voted item set and general item set receive more positive votes than negative ones. The positive and negative votes of the a-group voted item set are more balanced than that of general item set. Hence, the a-group voted item set attracts significantly more opposing votes than general item set.

2. In terms of biased inlink, a-group voted item set attracts significantly more biased votes than general item set.

3. In terms of biased inlink ratio, both a-group voted item set and general item set have biased inlink ratios larger than 86%. Hence, items attract highly biased opinions.

Metric		Positive inlink ratio	Biased inlink	Biased inlink ratio
A-group voted item set	Size	1379	1379	1379
	Mean	0.767	39.381	0.862
	Std Dev.	0.189	75.148	0.112
General item set	Size	106582	106763	106763
	Mean	0.875	7.059	0.891
	Std Dev.	0.206	12.105	0.157
z value		-21.137	15.969	-9.284

Table 4.7: Z-test of Inlink Metrics for Amazon Dataset

As the positive inlink ratio is the key measure of how controversial the item is, we examine this metric of the a-group voted item set and general item set in detail. We divides the positive inlink ratios of items into buckets of width 0.05. The distributions of positive inlink ratios of the two item sets are shown in Figure 4.7.

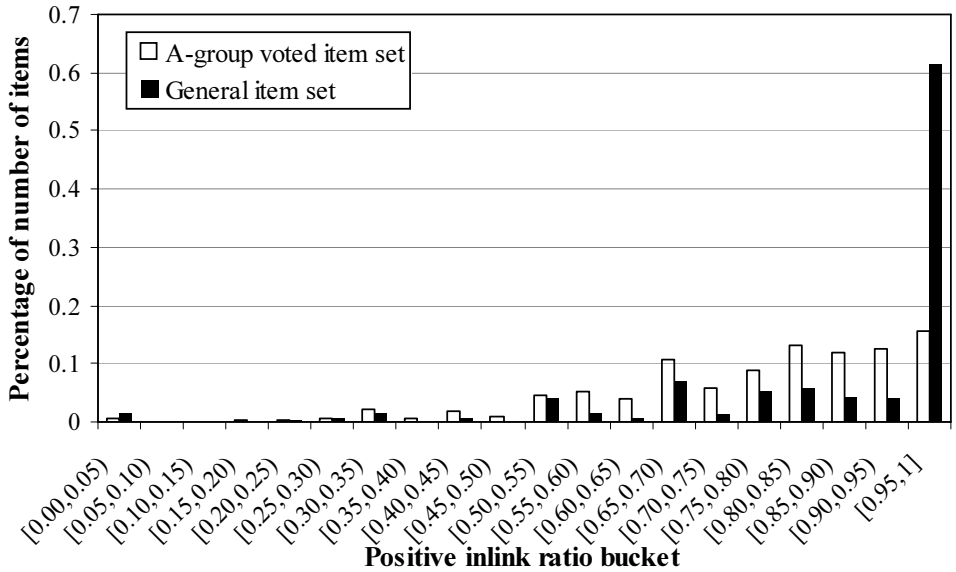


Figure 4.7: Distribution of Positive Inlink Ratio.

There are several observations made from the figure:

1. For the a-group voted item set, most of the items (i.e. 92.2%) have positive inlink ratios distributed over [0.50,1], with 4.0% in [0.60,0.65) being the lowest and 15.5% in [0.95,1] being the highest. The rest of the items have their positive inlink ratios distributed over [0,0.50). The number of items in buckets on [0,0.50) is small, with 2.2% in [0.30,0.35) being the highest.

2. For the general item set, 61.5% of the items have positive inlink ratios in [0.95,1]. Most of the rest items have their positive inlink ratios distributed over [0.50,0.95). Few items have their positive inlink ratios distributed over [0,0.50), with 1.6% in [0.30,0.35) being the highest.

These observations agree with the observation made from Table 4.7 that a-group voted item set receive more balanced positive and negative votes than general item set.

4.2.4 Comparison between Antagonistic Group Users and Other Users

In this section, we compare the set of users appearing in a-groups (called a-group user set) and the remaining users (called general user set). The a-groups are obtained with absolute λ=10 and σ=0.5. We adopt the user metrics for comparing the two user sets as follows:

1. **Positive outlink ratio**=$\frac{positive\_outdegree}{positive\_outdegree+negative\_outdegree}$. This metric reflects whether an user's opinions are biased towards positive or negative.

2. **Biased outlink**=$positive\_outdegree + negative\_outdegree$. This metric reflects the activeness level of an user.

To compare the two user sets, we retain only users with biased outlink $\geq$ (absolute λ) $\times$ σ. This is due to that users in a-group user set need to vote at least (absolute λ) $\times$ σ items as positive or negative. Thus, their biased outlink is at least (absolute λ) $\times$ σ. To be consistent with a-group user set, we also impose the restriction on the general user set. We apply the z-test similar to the one in Section 4.2.3. The z-test results are shown in Table 4.8. The z-values of the two metrics are all out of [-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 observe that:

1. In terms of positive outlink ratio, both a-group user set and general user set give more positive votes than negative ones. A-group users give more balanced positive and negative votes than general users.

2. In terms of biased outlink, a-group users give significantly more biased votes than general users.

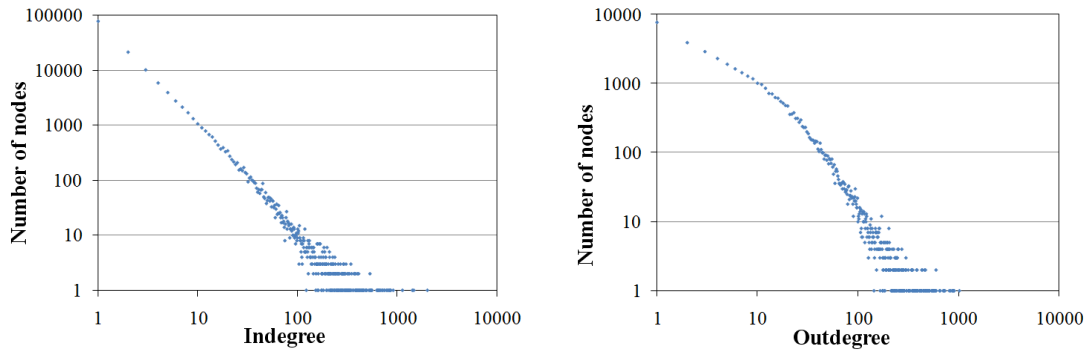
Metric		Positive outlink ratio	Biased outlink
A-group user set	Size	166	166
	Mean	0.693	208.572
	Std Dev.	0.264	634.060
General user set	Size	39517	39517
	Mean	0.846	15.108
	Std Dev.	0.196	29.820
z value		-7.468	3.931

Table 4.8: Z-test of Outlink Metrics for Amazon Dataset

4.3 Experiments on Epinions Dataset

4.3.1 Performance Study on Epinions Dataset

Epinions dataset is obtained from [2]. It was crawled from epinions.com by Paolo Massa. The dataset is a result of a 5-week crawl in November/December 2003. It contains 49,290 users who rated 139,738 different items in 664,823 reviews. The ratings scale from 1 to 5. Again, we map ratings of 4-5 to positive votes and ratings of 1-2 to negative votes. The rest are mapped to neutral votes. Among the 664,823 ratings, 495,392 (74.5%) are positive, 93,906 (14.1%) are negative, and 75,525 (11.4%) are neutral. The indegree distribution of items and the outdegree distribution of the users are shown in Figure 4.8. Both the indegree and outdegree distributions follow power law.



(a) (b)

Figure 4.8: Epinions Dataset Indegree and Outdegree Distribution.

We apply our indirect a-group mining algorithm on this dataset with σ=0.5 and different λ values. The results are shown in Figure 4.9. Figure 4.9(c) is obtained with absolute λ=10.

The results of Epinions dataset are similar to that of Amazon dataset. The

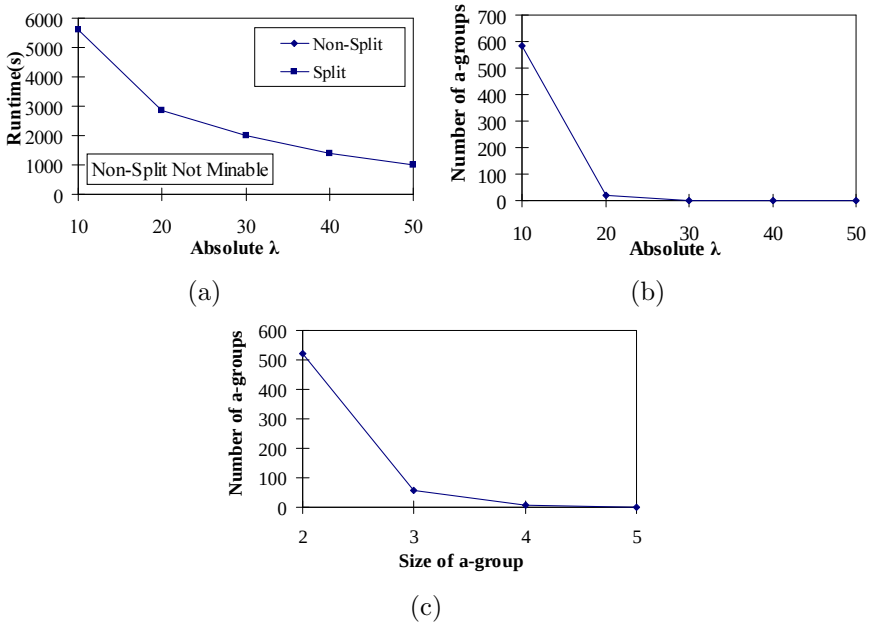


Figure 4.9: Epinions Dataset: Runtime and Distribution of A-groups.

number of a-groups is small even with very low support threshold. Most of the a-groups are of size 2. The antagonistic behavior is not so much apparent in this dataset.

4.3.2 Comparison between Antagonistic Group Voted Items and Other Items

In this section, we compare the a-group voted item set and the general item set. The a-groups are obtained with absolute λ=10 and σ=0.5. We compare the two sets using the same item metrics as shown in Section 4.2.3. We perform a similar z-test as the one in Section 4.2.3. Our z-test results are shown in Table 4.9. The z-values of the three metrics are all out of [-2.57,2.57]. Hence, we can say with 99% confidence that the two populations are different with respect to the three metrics. We also observe that:

1. In terms of positive inlink ratio, similar to the Amazon dataset, both of the two sets receive more positive votes than negative ones. The positive and negative votes received by a-group voted item set are more balanced than that of general item set.

2. In terms of biased inlink, a-group voted item set receives significantly more biased votes than general item set.

3. In terms of biased inlink ratio, both sets have biased inlink ratios larger than 85%. It suggests that items in both sets attract highly biased opinions.

Metric		Positive inlink ratio	Biased inlink	Biased inlink ratio
A-group voted item set	Size	1503	1503	1503
	Mean	0.732	93.582	0.858
	Std Dev.	0.236	124.059	0.100
General item set	Size	128015	138235	138235
	Mean	0.874	3.246	0.882
	Std Dev.	0.293	7.872	0.277
z value		-23.145	28.229	-8.744

Table 4.9: Z-test of Inlink Metrics for Epinions Dataset

Similar to the Amazon dataset, we examine the positive inlink ratio. We adopt a similar approach by dividing the positive inlink ratios of items into buckets of width 0.05. The distributions of positive inlink ratios of the two item sets are shown in Figure 4.10.

We can make the following observations from the figure:

1. For the a-group voted item set, most of the items (i.e. 83.2%) have positive inlink ratios distributed over [0.50,1], with 4.1% in [0.70,0.75) being the lowest and 15.9% in [0.95,1] being the highest. The remaining items have

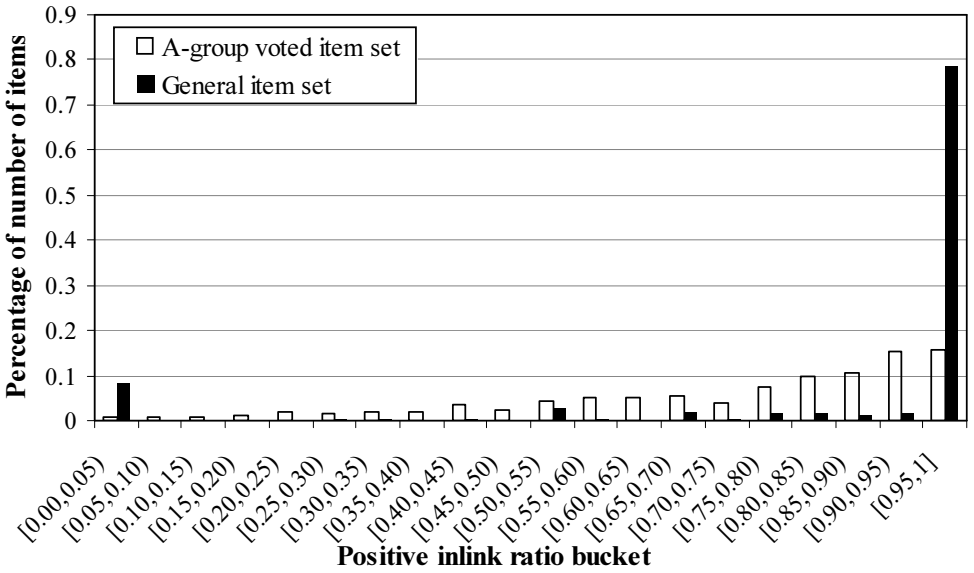


Figure 4.10: Distribution of Positive Inlink Ratio.

their positive inlink ratios distributed over [0,0.50) with 0.7% in [0.05,0.1) being the lowest and 3.6% in [0.4,0.45) being the highest.

2. For the general item set, 86.8% of the items have positive inlink ratios distributed at the two ends. 78.6% of the items have positive inlink ratios in [0.95,1] and 8.2% have positive inlink ratios in [0,0.05). The rest of the items have their positive inlink ratios distributed over [0.05,0.95), with 0.02% in [0.05,0.1) being the lowest and 2.9% in [0.5,0.55) being the highest.

These observations are consistent with the observation made from Table 4.9 that the positive and negative votes of a-group voted item set are more balanced than that of general item set.

4.3.3 Comparison between Antagonistic Group Users and Other Users

In this section, we compare the a-group user set and the general user set using the same user metrics as shown in Section 4.2.4. The a-groups are obtained with absolute λ=10 and σ=0.5. We perform a same z-test as the one in Section 4.2.3. Our z-test results are shown in Table 4.10. 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 positive outlink ratio, similar to the Amazon dataset, both a-group user set and general user set give more positive votes than negative ones. A-group users give more balanced positive and negative votes than general users.

2. In terms of biased outlink, a-group users tend to give significantly more biased votes than general users.

Metric		Positive outlink ratio	Biased outlink
A-group user set	Size	434	434
	Mean	0.772	146.150
	Std Dev.	0.136	128.980
General user set	Size	21873	21873
	Mean	0.845	22.461
	Std Dev.	0.132	32.039
z value		-11.108	19.966

Table 4.10: Z-test of Outlink Metrics for Epinions Dataset

4.4 Experiments on Slashdot Dataset

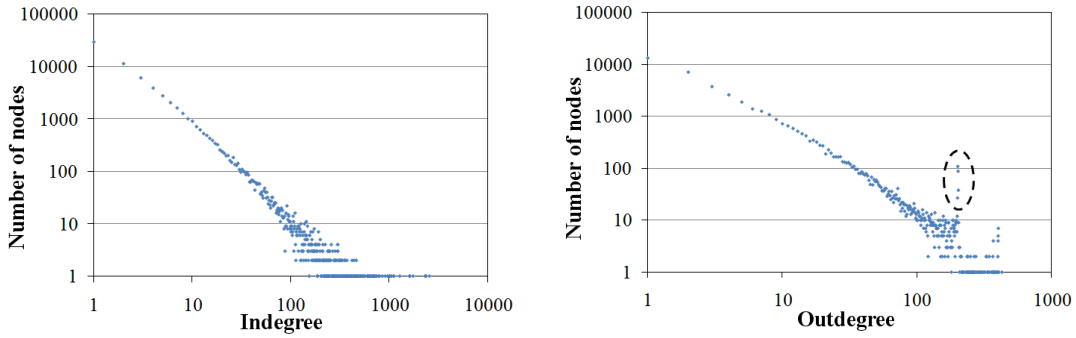
4.4.1 Performance Study on Slashdot Dataset

We downloaded the Slashdot dataset from [4]. Different from our previous two datasets, Slashdot dataset is not people voting items. It is people voting other people. In this dataset, a person vote another as "friend" (positive vote) or "enemy"(negative vote). There are no neutral votes in this dataset. Items here refer to the people receiving at least one vote and users refer to the people giving at least one vote.

A vote from a user to an item is represented by a link from the user to the item (outlink from the user and inlink to the item). The number of people nodes in our Slashdot dataset is 82,144. 44,044 (53.6%) of them have at least one outlink and 70,284 (85.6%) have at least one inlink. Many users (about 46%) do not have outlinks. The difference between number of nodes with at least one inlink and one outlink is 26,240. It indicates that there are some nodes with large number of outgoing links. We will find these nodes in the outdegree distribution to be shown later. There are 549,202 links with 425,072 (77.4%) positive and 124,130 (22.6%) negative. This suggests that users in Slashdot dataset give much more positive votes than negative votes.

The indegree and outdegree distribution of nodes are shown in Figure 4.11. As shown in Figure 4.11(a), the indegree is strictly power law distributed. Figure 4.11(b) shows that the outdegree follows power law too except four nodes in

the dashed circle. They show the nodes having outlinks to many other nodes. We further studied the nodes with many outlinks and found that there are few common nodes in their voted node set. They do not form any cliques (voting each other as friends or enemies) and they do not vote commonly on some groups of nodes. It seems these nodes vote a lot of other nodes without any special purposes.



(a) (b)

Figure 4.11: Slashdot Dataset Indegree and Outdegree Distribution.

We conduct our a-group mining on Slashdot dataset with σ=0.7 and absolute $\lambda \in \{20,30,40,50\}$. The result is shown in Figure 4.12. Figure 4.12(c) is obtained with absolute λ=20.

Figure 4.12(a) shows that the runtime decreases with larger λ. This is due to smaller number of a-groups as shown in Figure 4.12(b). Unlike the Epinions and Amazon datasets, most of the a-groups are of size 3 and some large size a-groups are mined. For example, we have around 200 a-groups of size 8. As the size of a-group increases, the number of a-groups decreases. This result implies that the Slashdot dataset tends to divide nodes into groups with opposing opinions.

Based on the setting of absolute λ=20 and σ=0.7, we select two a-groups, one small and another large for further case study analysis. The a-groups are:

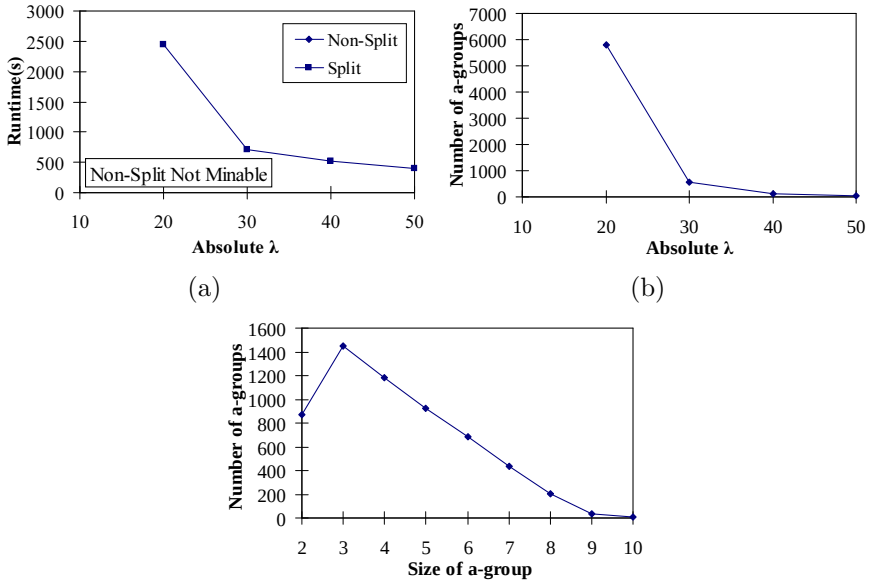


Figure 4.12: Slashdot Dataset: Runtime and Distribution of A-groups.

- g_1=({u1930,u2619},{u1499})
- g_2=({u2108,u13245,u58334},{u3445,u4693,u6818,u11020,u11044})

A-group g_1 has count(g_1)=20, antcount(g_1)=17 (aconf(g_1)=0.85). The number of commonly voted items of g_1 and their votes are shown in Table 4.11.

The a-group g_2 has count(g_2)=20, antcount(g_2)=18 (aconf(g_2)=0.9). The number of commonly voted items of g_2 and their votes are shown in Table 4.12.

Number of items	User's votes		
	1st sub-community of g_1		2nd sub-community of g_1
	u1930	u2619	u1499
3	−	+	+
17	+	+	−

Table 4.11: Votes of Users of g_1

Number of items	User's votes							
	1st sub-community of g_2			2nd sub-community of g_2				
	u2108	u13245	u58334	u3445	u4693	u6818	u11020	u11044
2	−	−	−	−	+	+	+	+
18	−	−	−	+	+	+	+	+

Table 4.12: Votes of Users of g_2

4.4.2 Comparison between Antagonistic Group Voted Items and Other Items

Same as the previous two datasets, we compare the a-group voted item set and the general item set. The a-groups are obtained with absolute λ=20 and σ=0.7. There is no neutral votes in this dataset. Hence, we compare the two sets using only item metric 1 and 2 as shown in Section 4.2.3. We perform a same z-test as the previous datasets. Our z-test results are shown in Table 4.13. The z-values indicate that, with 99% confidence, we can say the two populations are different with respect to the two metrics. We also observe that:

1. In terms of positive inlink ratio, similar to our previous two datasets, both sets receive more positive votes than negative ones. A-group voted item set receive more balanced positive and negative votes than general item set.

2. In terms of biased inlink, a-group voted item set receive significantly more biased votes than general item set.

Similar to our previous two datasets, we divides the positive inlink ratios of items into buckets of width 0.05. The distributions of positive inlink ratios of the two item sets are shown in Figure 4.13.

Metric		Positive inlink ratio	Biased inlink
A-group voted item set	Size	2556	2556
	Mean	0.647	81.172
	Std Dev.	0.220	147.063
General item set	Size	67728	67728
	Mean	0.779	5.046
	Std Dev.	0.347	12.632
z value		-28.986	26.167

Table 4.13: Z-test of Inlink Metrics for Slashdot Dataset

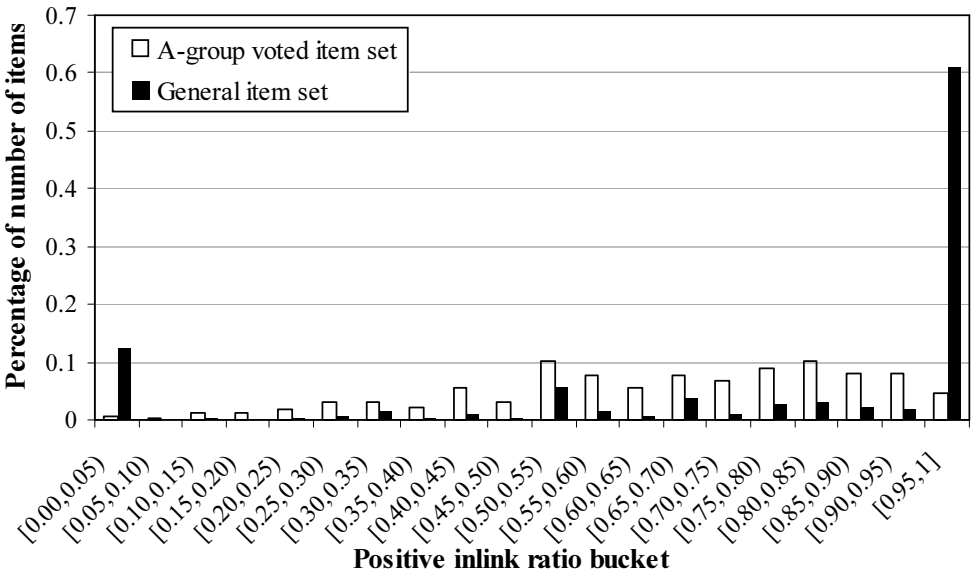


Figure 4.13: Distribution of Positive Inlink Ratio.

Some observations can be made from the figure:

1. For the a-group voted item set, 83.5% of the items have positive inlink ratios evenly distributed over [0.50,1], with 4.7% in [0.95,1] being the lowest and 10.3% in [0.80,0.85) being the highest. The gap between the lowest and the highest is smaller compared to our previous two datasets. The rest of the items have their positive inlink ratios distributed over [0,0.50) with 0.3% in [0.05,0.1) being the lowest and 5.6% in [0.4,0.45) being the highest.

2. For the general item set, similar to the Epinions dataset, a large amount (i.e. 73.5%) of the items have positive inlink ratios distributed at the two

ends. 61.1% of the items have positive inlink ratios in [0.95,1] and 12.4% have positive inlink ratios in [0,0.05).The rest of the items have their positive inlink ratios distributed over [0.05,0.95), with 0.05% in [0.05,0.1) being the lowest and 5.7% in [0.5,0.55) being the highest.

Similar to our previous two datasets, these observations show that a-group voted item set indeed receives more balanced positive and negative votes than general item set.

4.4.3 Comparison between Antagonistic Group Users and Other Users

In this section, we compare the a-group user set and the general user set based on the two user metrics as shown in Section 4.2.4. The a-groups are obtained with absolute λ=20 and σ=0.7. We perform the same z-test as our previous datasets. Our z-test results are shown in Table 4.14. The z-values indicate that with 99% confidence we can say the two populations are different with respect to the two metrics. We also observe that:

1. In terms of positive outlink ratio, similar to our previous two datasets, both a-group user set and general user set give more positive votes than negative ones. A-group users give more balanced positive and negative votes than general users.

2. In terms of biased outlink, a-group users give significantly more biased votes than general users.

Metric		Positive outlink ratio	Biased outlink
A-group user set	Size	399	399
	Mean	0.682	182.727
	Std Dev.	0.344	91.403
General user set	Size	7704	7704
	Mean	0.790	46.106
	Std Dev.	0.245	44.925
z value		-6.199	29.672

Table 4.14: Z-test of Outlink Metrics for Slashdot Dataset

4.5 Experiments on Wikipedia Vote Dataset

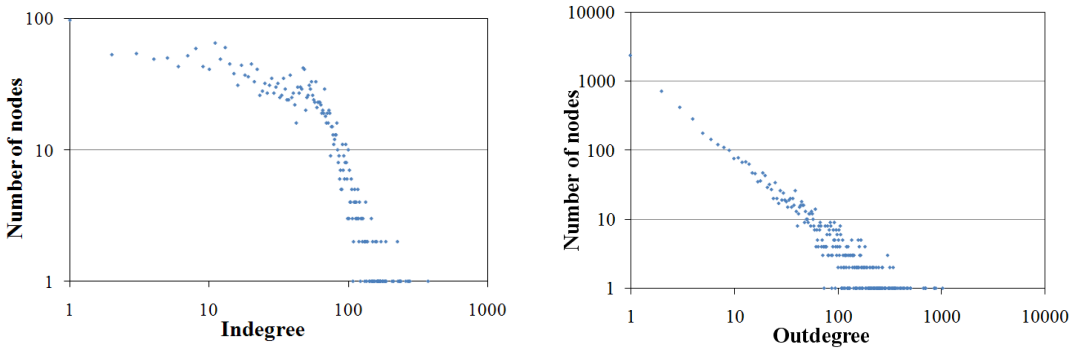
4.5.1 Performance Study on Wikipedia Vote Dataset

We downloaded the Wikipedia vote dataset from [4]. In this dataset, people are voted by others for promoting to administrators. The voting process is divided into sessions. The vote can be 0 (neutral), 1 (support) or -1 (against). We map 1 to positive vote, -1 to negative vote and 0 to neutral vote. The person is promoted if at least 75% of his/her votes are positive [12]. A person can be nominated more than once and voted in multiple sessions. Each session has a unique time stamp and the votes for the same person in different sessions are different. We treat sessions as items, and voters as users.

Similar to the Slashdot dataset, a vote from a user to an item is represented by a link from the user to the item (outlink from the user and inlink to the item). The number of item is 2794. Number of items with at least one inlink (been voted) is 2794 (100%). Number of users is 8274. Number of users with at least one outlink (vote others) is 6210 (75.1%). The total number of links is 114,040. Among the links, 83,962 (73.6%) are positive, 23,118 (20.3%) are negative and

6960 (6.1%) are neutral.

The indegree distribution of items and the outdegree distribution of users are shown in Figure 4.14. Figure 4.14(a) shows that the indegree is not strictly power law distributed. This is due to the small number of items. Figure 4.14(b) shows the outdegree obeys the power law distribution.



(a) (b)

Figure 4.14: Wikipedia Voting Dataset Indegree and Outdegree Distribution.

Our experiments on mining a-groups are conducted with σ=0.7 and absolute λ $\in$ {20,30,40,50}. The results are shown in Figure 4.15. Figure 4.15(c) is obtained with absolute λ=20.

Figure 4.15(a) shows both Split version and Non-Split version can cop with this dataset. This is due to that the dataset is relatively small compared to our previous datasets. Figure 4.15(a) also shows that the runtime decreases as λ becomes larger. This is due to the smaller number of a-groups minable as shown in Figure 4.15(b). Most of the a-groups mined are of size 3 and 4. We also find some large size a-groups. For example, we find 40 a-groups of size 6. The number of a-groups decreases as size of a-group increases. The results show that Wikipedia vote dataset is likely to divide its nodes into groups of opposing opinions.

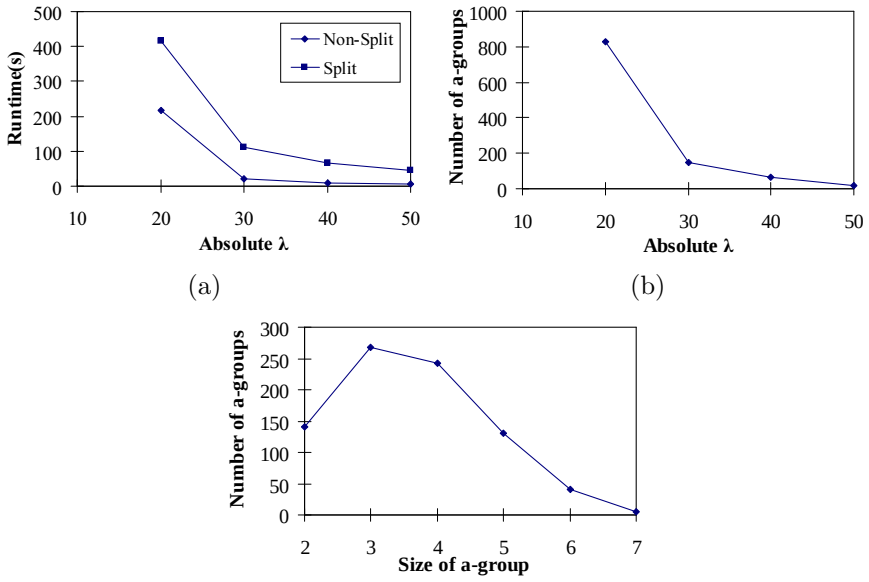


Figure 4.15: Wikipedia Voting Dataset: Runtime and Distribution of A-groups.

Based on the setting of absolute λ=20 and σ=0.7, we select an a-group of size six for analysis. The a-group is:

- g=({u1133,u2237,u2565,u2713,u3352},{u2229})

A-group g has count(g)=20, antcount(g)=15 (aconf(g)=0.75). The number of commonly voted items of g and their votes are shown in Table 4.15.

Number of items	User's votes					
	1st sub-community of g					2nd sub-community of g
	u1133	u2237	u2565	u2713	u3352	u2229
1	+	+	+	−	−	−
4	+	+	+	+	−	−
15	+	+	+	+	+	−

Table 4.15: Votes of Users of g

4.5.2 Comparison between Antagonistic Group Voted Items and Other Items

We now compare the a-group voted item set and the general item set using the three item metrics in Section 4.2.3. The a-groups are obtained with absolute λ=20 and σ=0.7. We perform the same z-test as our previous datasets. Our z-test results are shown in Table 4.16. We observe that:

1. In terms of positive inlink ratio, opposite to our previous three datasets, the mean positive inlink ratio of a-group voted item set is larger than that of general item set. The standard deviation of positive inlink ratios of general item set is nearly twice of that of a-group voted item set.

2. In terms of biased inlink, similar to our previous datasets, a-group voted item set receives significantly more biased votes than general item set.

3. In terms of biased inlink ratio, both sets have biased inlink ratios larger than 91%. This suggests items in both sets attract highly biased opinions.

Metric		Positive inlink ratio	Biased inlink	Biased inlink ratio
A-group voted item set	Size	841	841	841
	Mean	0.742	59.377	0.926
	Std Dev.	0.238	40.082	0.079
General item set	Size	1945	1953	1953
	Mean	0.533	29.260	0.919
	Std Dev.	0.419	26.749	0.126
z value		16.683	19.960	1.648

Table 4.16: Z-test of Inlink Metrics for Wikipedia Voting Dataset

In Wikipedia, the nominee in each session (i.e. item in our context) is promoted to an administrator if at least 75% of his/her votes are positive [12]. We now study

whether there are any difference in successful promotion rate of the a-group voted item set and the general item set. The election results of the two sets are shown in Table 4.17.

	A-group voted item set	General item set
Number of sessions	841	1953
Number/Percentage of successful promotion	456/54.2%	792/40.6%
Number/Precentage of failed promotion	385/45.8%	1161/59.4%

Table 4.17: Election Results of the Two Sets

The table shows that the a-group voted item set has higher successful promotion rate than the general item set. The difference is around 15%. We further plot the distributions of positive inlink ratios for both sets in Figure 4.16.

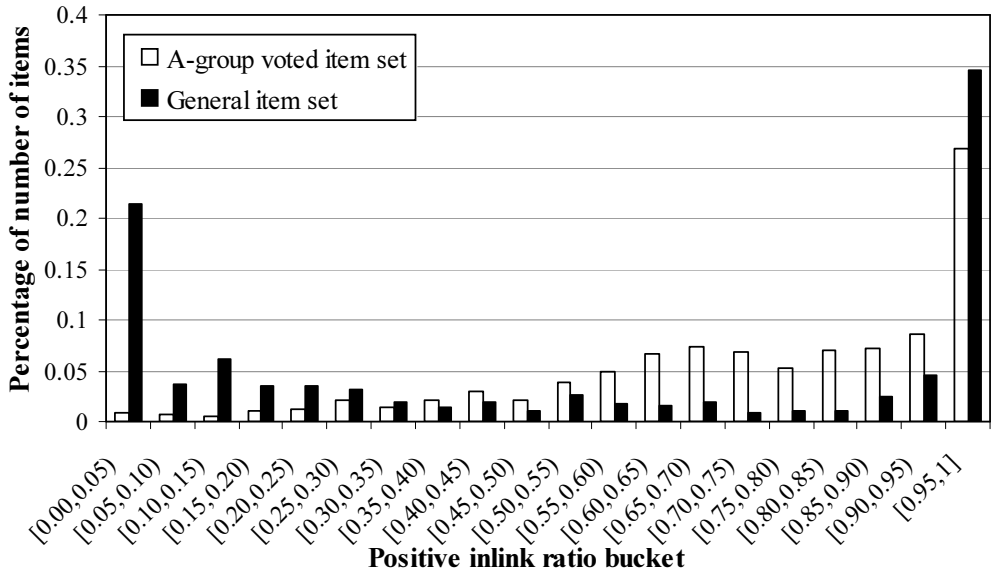


Figure 4.16: Distribution of Positive Inlink Ratio.

As shown in Figure 4.16, the a-group voted item set has 55.2% of the items having positive inlink ratios in [0.75,1] and general item set has 43.8% of the items having positive inlink ratios in [0.75,1]. The two numbers are slightly different from the successful promotion rates in Table 4.17, as the promotion is not solely based on votes and other factors such as voter's reasons are also considered [12].

The gap between these two numbers is nearly 15%. This agrees with the results in Table 4.17.

The figure also shows that the items of the a-group voted item set are evenly distributed over [0.50,0.95). The exception is the interval [0.95,1], which has 26.9% of the items. These items pull up the successful promotion rate of the a-group voted item set and they also cause the mean positive inlink ratio of a-group voted item set to be higher than that of general item set.

To find out the reasons why a-group voted item set has so many items in [0.95,1], we investigated into the 827 a-groups mined. We found three users forming many a-groups. User u3569 forms 61 a-groups. User u1029 forms 151 a-groups. User u2229 forms 497 a-groups. We found that u3569 gave negative votes to 41 items and positive votes to only 2 items. u1029 gave negative votes to 117 items and positive votes to 20 items. u2229 gave negative votes to 129 items and positive votes to 17 items. We infer that as these three uses give negative votes to items, their opposing users give positive votes. In addition, a-groups formed by these three users are all in the form of "u3569/u1029/u2229,uAuBuC..."("/" means either one, A, B and C are user Id). This means there are many users giving positive votes to the a-group voted items voted by the three users. Thus, the positive inlink ratios of a-group voted items voted by the three users are high. These three users voted a total of 319 items (7 of them are concurrently voted by two or three). 314 of the items appear in a-group voted item set. Among the 314 items, 117 items have positive inlink ratios in [0.95,1]. Hence, 51.7% (i.e. more than half) of the items in [0.95,1] are contributed by items voted by these three

users. They are the key factors pulling up the number of items in [0.95,1].

4.5.3 Comparison between Antagonistic Group Users and Other Users

In this section, we compare the a-group user set and the general user set using the same metric in Section 4.2.4. The a-groups are obtained with absolute λ=20 and σ=0.7. We perform a z-test same as our previous datasets. The z-test results are shown in Table 4.18. The z-values indicate that there is no significant difference in positive outlink ratios between the two sets, and they both give more positive votes than negative votes. However, a-group users give significantly more biased votes than general users.

Metric		Positive outlink ratio	Biased outlink
A-group user set	Size	162	162
	Mean	0.794	182.598
	Std Dev.	0.225	156.759
General user set	Size	1254	1254
	Mean	0.780	50.766
	Std Dev.	0.179	48.874
z value		0.777	10.637

Table 4.18: Z-test of Outlink Metrics for Wikipedia Voting Dataset

Chapter 5

Direct Antagonistic Group Mining

In this chapter, we examine the mining of direct antagonistic groups. The definition and properties are given first, followed by the mining algorithm. The experiment results are introduced finally.

5.1 Preliminaries

Definition 5.1.1 (***Strongly Connected Subgraphs***)**.** *A strongly connected subgraph (SCS) is a sub-graph G' in a larger graph G where: For each node n' in G', there exists a series of edges in G' connecting n' to every other node in G'.*

Definition 5.1.2 (***Strongly Connected Component***)**.** *A strongly connected component (SCC) is a strongly connected sub-graph that is maximal in size.*

An example of a strongly connected component (SCC) is shown in Figure 5.1.

v3, v4 and v5 form a strongly connected subgraph (SCS) but is not an SCC as there is a larger SCS, i.e. {v1,v2,v3,v4,v5}.

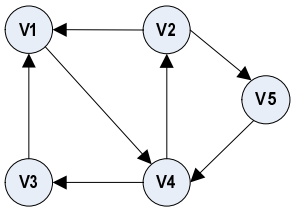


Figure 5.1: A Strongly Connected Component

Definition 5.1.3 (***Bi-Cliques***)**.** *A bi-clique is a graph whose nodes could be decomposed of two sets of nodes where:*

1. *There is no edge among the nodes in each set*
2. *Each node is connected to every nodes in the other set.*

We denote a bi-clique as (L,R), where L and R are the two sets of nodes having the characteristics described above.

An example of a bi-clique is shown in Figure 5.2.

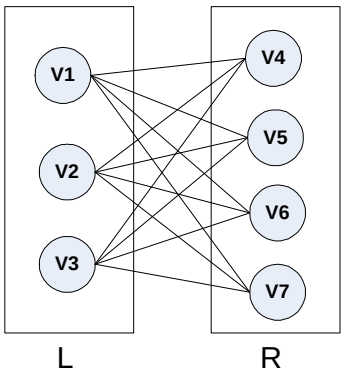


Figure 5.2: A Bi-Clique.

Definition 5.1.4 (***Sub-Bi-Clique***)**.** *Consider a bi-clique C = (L,R). We define a sub-bi-clique of C, as a bi-clique C' = (L',R') where either L' $\subseteq$ L and R' $\subset$ R, or L' $\subset$ L and R' $\subseteq$ R. A sub-bi-clique of a bi-clique is a bi-clique. The set of all possible sub-bi-cliques of C is denoted as sbc(C).*

To illustrate the sub-bi-clique operation, two sample sub-bi-cliques of the one shown in Figure 5.2 are shown in Figure 5.3.

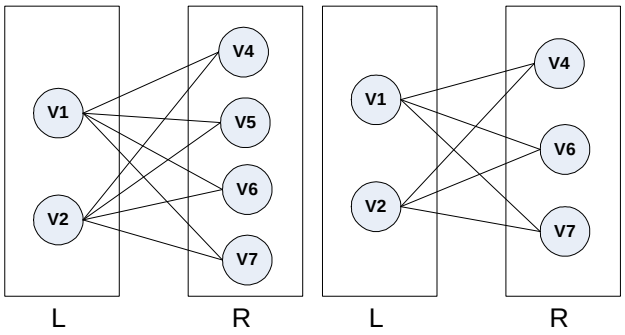


Figure 5.3: Example Sub-Bi-Cliques

Definition 5.1.5 (***Transaction DB and Mapping Function***)**.** *A transaction is a set of items from a domain D. A transaction database DB consists of a bag of transactions. Let map(S) be a mapping between a set of items S to the identifiers of the transactions in the DB containing S.*

Definition 5.1.6 (***Itemset Patterns***)**.** *An itemset pattern is a set of items. Consider a transaction database DB, the support of a pattern P, is the number of transactions in the database that are super-sets of P. The support of P is denoted as sup(P).*

Definition 5.1.7 (***Frequent Itemsets***)**.** *An itemset P is a frequent itemset with respect to a transaction database DB and a minimum support threshold min_sup, if the support of P is larger than min_sup.*

Definition 5.1.8 (***Closed Patterns***)**.** *An itemset P is a closed pattern, if P is frequent and there is no P' where P' $\supseteq$ P and sup(P') = sup(P).*

An example of a transaction database is shown in Table 5.1. The itemset {A,B,C} is supported by three transactions namely T1, T2 and T3. The support

TID	Itemset
T1	{A,B,C,D,E}
T2	{A,B,C,D,K}
T3	{A,B,C,D}
T3	{A,C,D}
T4	{E,K,F}

Table 5.19: A Sample Transaction DB

of the itemset is therefore 3. Considering a minimum support of 3, the itemset is a frequent one. However, since there exists a longer itemset {A,B,C,D} with the same support, the itemset {A,B,C} is not closed. Itemset {A,B,C,D} however is closed.

Definition 5.1.9 (***Graph to Transaction DB***)**.** *We define a new operation GTD to convert a graph G to a transaction database DB. For each node g ∈ G, we create a new set of transactions t = {g'|(g,g') ∈ G.Edges} and affix identifier g to t. The resultant set of transactions is the result of the operation GTD(G).*

The conversion from a graph to a transaction database is illustrated in Figure 5.4.

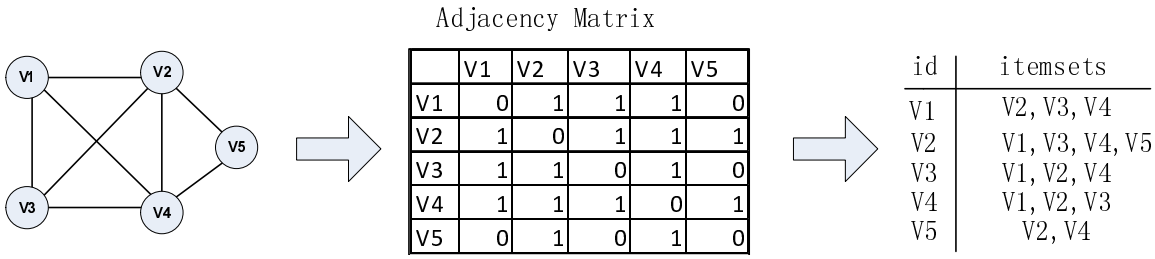


Figure 5.4: Graph to Transaction DB Operation

Property 5.1.1 (***Bi-cliques and Patterns: Duality***)**.** *Consider a graph G and a transaction database GTD(G). Let us mine all closed patterns CLS from the resultant database. The set of all bi-cliques correspond to the set {(c,map(c))|c∈ CLS}*

Proof. The above property has been proven in [31]. □

We take as input a network of users expressing positive and negative relationships among themselves. We refer to this network as a Positive-Negative network defined in Definition 5.1.10.

Definition 5.1.10 (***Positive-Negative Network***)**.** *A positive-negative (P-N) network is a graph whose nodes represent individuals and edges represent positive or negative relationships among them. The edges are directed and are labeled with either: positive (P) or negative (N). The nodes are labeled with the identifiers of respective individuals. A positive-negative network could then be denoted as* $G=(N,E,N_L,E_L)$ *where* N, E, N_L, *and* E_L *correspond to the nodes, edges, a mapping from nodes to labels, and a mapping from edges to labels.*

We are to mine groups of two sub-communities with positive relationship among nodes in the same sub-communities and negative relationship among nodes in opposing sub-communities. We refer to such a group of sub-communities as direct antagonistic group.

Definition 5.1.11 (***Direct Antagonistic Group***)**.** *A Direct Antagonistic Group (direct a-group) is composed of two sub-communities L and R. L and R are both SCCs* **with respect to the directed positive edges**. *L and R together form a bi-clique considering* **bidirectional distrust edges**.

An example of such a direct a-group is shown in Figure 5.5.

We are interested in direct a-groups obeying a minimum size requirement, i.e.,

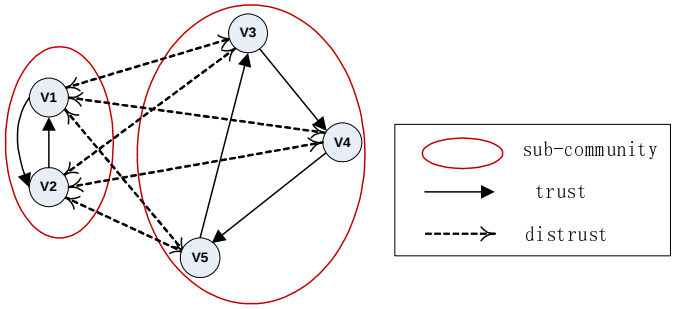


Figure 5.5: A Direct Antagonistic Group.

$|L| \geq$ min_size and $|R| \geq$ min_size. We refer to such direct a-groups as *significant* direct a-groups. The example direct a-group shown in Figure 5.5 is significant if the minimum size threshold is set at 3; it would not be significant if the minimum size threshold is set at 4.

Property 5.1.2 (***Membership***)**.** *Consider a node n in graph G, if n is not part of any SCCs of size min_size, n could not be part of any significant direct a-groups.*

Proof. From Definition 5.1.11, each sub-community in the direct a-group must be an SCC of size not smaller than min_size. Hence, if a node is not part of any SCCs, it could not be part of any direct a-group. □

Definition 5.1.12 (***Redundant Direct Antagonistic Group***)**.** *Consider a set of direct a-groups ASET. A direct a-group a in ASET is deemed as* redundant *iff there exists another direct a-group a', where a is a sub-bi-clique of a'.*

With the above concepts and definitions, our direct antagonistic group mining problem definition is given as follows:

Definition 5.1.13 (***Direct Antagonistic Group Mining Problem***)**.** *Given a positive-negative network and a minimum size threshold min_size, find all significant non-redundant direct a-groups obeying the min_size threshold.*

The algorithm to mine direct antagonistic group has been proposed in [5]. Readers can refer to Appendix B for details.

5.2 Experiment Results

To evaluate the scalability and efficacy of the mining algorithm, experiments on both synthetic datasets and real datasets are conducted. Detailed experimental results and analysis are available in [5]. Here, we show the experiment results with an Epinions dataset consisting of a positive-negative network [3]. The Epinions dataset has 405,176 user nodes and about 840K links. We remove those nodes without positive or negative links. We obtain 131,828 nodes, with 717,667 (about 85%) positive links and 123,705 (about 15%) negative links.

5.2.1 Efficacy Study on Epinions Dataset

To test the efficacy of our idea, we first pair the users in a direct a-group. For a direct a-group (U_l, U_r), we call user pairs $\{(u_i, u_j)|u_i \in U_l, u_j \in U_r\}$ opposing user pairs and $\{(u_i, u_j)|u_i, u_j \in U_l\} \cup \{(u_i, u_j)|u_i, u_j \in U_r\}$ allied user pairs. We make comparison between the behaviors of allied user pairs and opposing user pairs. We expect the behaviors of allied user pairs to be friendly towards each other, while those of opposing user pairs to be unfriendly towards each other.

The first study is on whether allied user pairs tend to give higher ratings to each other's reviews than opposing pairs. The dataset contains a total of 1.2M reviews and about 4.5M ratings. For users u_i and u_j of a pair, u_i may rate k

reviews written by u_j with k ratings $\{r_{ij1}, \cdots, r_{ijk}\}$.

In Figure 5.6, we show the user pair distribution on rating scores. The general user pairs refer to user pairs not involved in direct a-groups. A larger proportion of allied user pairs have high rating scores than opposing user pairs and general user pairs. 95% of allied user pairs have ratings of 5, while 80% of opposing user pairs have such ratings. Smaller proportion of both allied user pairs and general user pairs have ratings of 2, while 10% of opposing user pairs have such ratings. This shows allied user pairs tend to give each other high ratings, while opposing user pairs tend to give each other low ratings due to their unfriendly relationship. We have also performed hypothesis testing at 0.01 level of significance and we find that the rating scores among allied user pairs differ significantly from opposing user pairs.

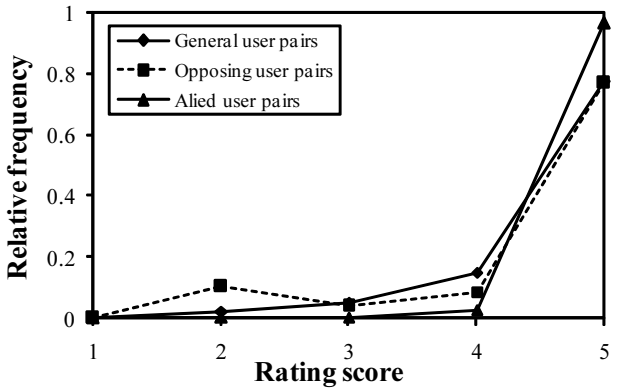


Figure 5.6: Rating-scores: Opposing vs Allied User Pairs

Figure 5.7 shows the distribution of the number of ratings of allied user pairs and opposing user pairs in the mined direct a-groups. It can be noted that the members of allied user pairs tend to give each other more ratings than opposing user pairs. This suggests that people maintaining good relationship tend to rate each other more than people in hostile relationship. This matches our intuition. We have also performed hypothesis testing at 0.01 level of significance and we find

that the distribution of number of ratings of allied user pairs differs significantly from opposing user pairs.

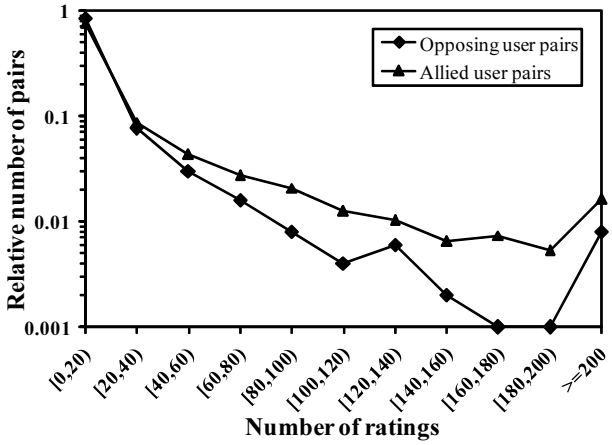


Figure 5.7: Number-of-ratings: Opposing vs Allied User Pairs

Chapter 6

Conclusion and Future Work

6.1 Summary of Research

In this thesis, we conduct research on antagonistic communities in social networks. We defined the problem of mining indirect and direct a-groups.

For indirect a-group mining, we propose an Apriori like algorithm to mine all the closed a-groups from vote databases. Our algorithm traverses the search space of all possible a-groups and adopts several pruning strategies to prune search spaces with no valid a-groups. In addition, we have developed a divide and conquer strategy to allow the algorithm running on large databases.

To test the efficiency and effectiveness of our algorithm, we develop a synthetic data generator and experimented on several synthetic datasets. The results show that our algorithm can run well on various data settings. We have also conducted extensive experiments on real datasets. The results show that our algorithm is able to mine interesting a-groups in various real datasets. We have also compared the

items voted by at least one a-group and the ones voted by non-a-group users. The results show that the two item sets differ significantly. It indicates that a-group users and their voted items are special, and they are worth further exploration.

For direct a-group mining, we propose the concept of direct a-group and study its properties. We conducted experiments to study the characteristics of direct a-group. We compare the behaviors of nodes in allied pairs and opposing pairs. The results show that nodes of allied pairs are more friendly to each other than nodes of opposing pairs.

Although our proposed solutions can solve the a-group mining problem well and show interesting results in the experiments, there are some limitations. For indirect a-group mining, our algorithm is still very time consuming in candidates generation. This causes efficiency bottleneck in mining large real datasets. Secondly, our mined a-groups (both direct and indirect) are all of small size. The largest one is of around ten users. We may need to adopt a more flexible a-group definition to accommodate large a-groups.

6.2 Future Work

In this section, some of the important future works are discussed.

1. Improvement on the candidate generation in indirect antagonistic group mining algorithm.

   As shown in chapter 3, our candidate generation employs a merge and a

prune stage. In the merge stage, currently, we only consider merging the frequent o-groups of the same size. This results in an incremental generation of the candidates. Alternatively, we could consider merging frequent o-groups of different sizes and with one common sub-community. It may speed up our candidates generation process. However, the increase in the merging time is a drawback. To overcome this drawback, some novel and efficient algorithms for finding frequent o-groups eligible to be merged should be developed.

2. Enhancement of split version of indirect antagonistic group mining algorithm.

   Currently, our split version algorithm (Algorithm 8) splits the database by user. Hence, the number of sub-databases is the same as number of users, which is typically a large number. We may consider splitting the database based on a group of users. This decreases the number of sub-databases and hence, decreases the running time. In addition, if there are many long transactions in the database, our original splitting approach will duplicate the transactions as many as the number of items in such transactions. Splitting by groups of users will reduce the number of duplicates of such transactions and hence, decreases the storage cost. Careful selection of the group of users to split on is a must to minimize the storage and time cost. However, this new splitting way will increase the database splitting time as checking of which groups are contained in each transaction is needed. Nevertheless, this database splitting time should not differ significantly from the original splitting way, since the database needs to be scanned only once.

3. Investigation on relationship between indirect antagonistic group and direct antagonistic group.

   For some datasets like Wikipedia vote dataset and Epinions dataset (the one used in direct a-group mining), people nodes give votes to other people nodes. We can view the networks from two perspectives. The first perspective is viewing the people nodes as both voters and votees. Hence, we can mine indirect a-groups from them. The second perspective is viewing the people nodes having positive and negative relationships with each other. Hence, we can mine direct a-groups from them. Therefore, for the same network, we get both indirect and direct a-groups. We may conduct further studies on the relationships between these two kinds of a-groups, such as whether they overlap, is there violations (two nodes are friends in one kind of a-group, which they are enemies in the other kind of a-group), etc..

4. Developing Depth First based indirect antagonistic group mining algorithm.

   Currently, our mining algorithm (Algorithm 2) for indirect a-group is Breadth First (BF) based. It has the shortcomings of the other BF based frequent itemset mining algorithms, such as time consuming candidates generation and multiple database scans. Since our problem has similar nature as frequent itemset mining, we may consider employing Depth First based mining algorithm, such as pattern growth [22]. In this way, we can improve the efficiency of the mining process and we may mine a-groups from larger datasets and get larger size a-groups.

Bibliography

[1] Amazon website. http://www.amazon.com/.

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

[3] Review from Epinions. http://www.epinions.com/.

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

[5] Technical report. http://www.mysmu.edu/faculty/davidlo/papers/www2011.pdf.

[6] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. Depth-first generation of large itemsets for association rules. *IBM Tech. Report*, RC(21538), 1999.

[7] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent item sets. *J. Parallel Distrib. Comput.*, 61(3):350–371, 2001.

[8] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD'93*, pages 207–216, 1993.

[9] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94*, pages 487–499, 1994.

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

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

[12] M. Burke and R. Kraut. Mopping up: modeling wikipedia promotion decisions. In *CSCW'08*, pages 27–36, 2008.

[13] D. Cai, Z. Shao, X. He, X. Yan, and J. Han. Community mining from multi-relational networks. In *PKDD'05*, volume 3721, pages 445–452, 2005.

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

[15] I. Dasgupta. 'living' wage, class conflict and ethnic strife. *Journal of Economic Behavior & Organization*, 72(2):750–765, Nov 2009.

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

[17] I. de Sola Pool and M. Kochen. Contacts and influence. *Social Networks*, 1(1):5 – 51, 1978-1979.

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

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

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

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

[22] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD'00*, volume 29, pages 1–12, 2000.

[23] F. Harary. On the notion of balance of a signed graph. *Michigan Math. Journal*, 2(2):143–146, 1953.

[24] M. Houtsma and A. Swami. set-oriented mining of association rules. *Research Report*, RJ(9567), 1993.

[25] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241–254, 1967.

[26] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, January 1953.

[27] G. Labianca, D. J. Brass, and B. Gray. Social networks and perceptions of intergroup conflict: The role of negative relationships and third parties. *The Academy of Management Journal*, 41(1):pp. 55–67, 1998.

[28] S. Labovitz and R. Hagedorn. A structural-behavioral theory of intergroup antagonism. *Social Forces*, 53(3):444–448, Mar 1975.

[29] E. A. Leicht and M. E. J. Newman. Community structure in directed networks. *Physics Review Letter*, 100(11):118703, Mar 2008.

[30] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *WWW'10*, pages 641–650, 2010.

[31] J. Li, G. Liu, H.Li, and L. Wong. Maximal biclique subgraphs and closed pattern pairs of the adjacency matrix: A one-to-one correspondence and mining algorithms. *IEEE TKDE*, 19(12):1625–1637, Dec 2007.

[32] R. E. Nelson. The strength of strong ties: Social networks and intergroup conflict in organizations. *The Academy of Management Journal*, 32(2):pp. 377–401, 1989.

[33] M. E. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Physics Review E*, 64(2 Pt 2), August 2001.

[34] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physics Review*, E(69), 2004.

[35] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.

[36] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabasi. Hierarchical Organization of Modularity in Metabolic Networks. *Science*, 297(5586):1551–1555, 2002.

[37] J. Scott. *Social network analysis: a handbook*. Sage Publications, 2000.

[38] S. H. Strogatz. Exploring complex networks. *Nature*, 410(6825):268–276, March 2001.

[39] R. Tarjan. Depth-first search and linear graph algorithms. *JSIAM Journal on Computing*, 1(2):146–160, 1972.

[40] 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.

[41] V. A. Traag and J. Bruggeman. Community detection in networks with positive and negative links. *Physical Review E*, 80(3):036115, Sep 2009.

[42] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *ICDE'04*, pages 79–90, 2004.

[43] J. Wang, J. Han, and J. Pei. Closet+: searching for the best strategies for mining frequent closed itemsets. In *KDD'03*, pages 236–245, 2003.

[44] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[45] 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.

[46] X. Yan, J. Han, and R. Afhar. CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of SIAM International Conference on Data Mining*, 2003.

[47] B. Yang, W. Cheung, and J. Liu. Community mining from signed social networks. *IEEE Trans. on Knowl. and Data Eng.*, 19(10):1333–1348, 2007.

[48] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[49] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *KDD'97*, pages 283–286, 1997.

[50] K. Zhang, D. Lo, and E.-P. Lim. Mining antagonistic communities from social networks. In *PAKDD (1)*, pages 68–80, 2010.

Appendix A

Antagonistic Group Mining Software

A.1 Indirect Antagonistic Group Mining Software

A.1.1 Mining Program Configuration

To perform indirect a-group mining, two programs are needed. One is the ANTGroup-Mining.java which is the main mining program and the other is Node.java which is used by the first program. To run the programs on a vote dataset, we need first to map the different types of votes in the dataset to voting scores (numerical value). We then generate an input.txt file to be stored in the same directory of the two programs as the input file. The input.txt file is of the following format:

|U| |I|

Negative_threshold Positive_threshold λ σ

uA,v_{A0} uB,v_{B0} uC,v_{C0} ...

uD,v_{D1} uE,v_{E1} uF,v_{F1} ...

.

.

uH,v_{Hi} uI,v_{Ii} uJ,v_{Ji} ...

.

.

uO,$v_{O(|I|-1)}$ uP,$v_{P(|I|-1)}$ uQ,$v_{Q(|I|-1)}$ ...

The top two lines of the input file specify the parameters of the dataset and various thresholds. Their meanings are shown in Table A.20. Negative_threshold is the threshold for mapping voting scores to negative votes. Any voting scores less than or equal to negative_threshold will be mapped to negative votes (e.g. for voting scheme of score 1-5, we may specify voting scores less than or equal to 2 to be negative). Similarly, any voting scores more than or equal to positive_threshold will be mapped to positive votes. For the rest, each line represents the (user,vote) pair list of an item. The item index starts from 0 and increment one as the line number increases. For each (user,vote) pair list, the pairs are separated by space. The (user,vote) pair is in the format of uK,v_{Ki} where u is fixed character representing "user", K is the user Id (numerical value), v_{Ki} is the voting score given to item i by user K . Each (user,vote) pair list needs to be sorted in ascending order by user Ids.

After the input file and programs are stored in the same directory, we can run the program using the command line "java -Xmx1g ANTGroupMining" to

Parameter	Meaning
|U|	Number of users
|I|	Number of items
Negative_threshold	Threshold for mapping voting scores to negative vote
Positive_threshold	Threshold for mapping voting scores to positive vote
λ	Support threshold
σ	Antagonistic confidence threshold

Table A.20: Parameters of Mining Program Input File

execute the program. The command option "-Xmx1g" is used to increase the memory usable by this program to 1GB.

The output of the program is as follows:

uA,uB $aconf_1$ $count_1$

uCuD,uE $aconf_2$ $count_2$

.

.

.

n

Each line is an a-group, followed by its aconf and count. An example of an a-group is uCuD,uE. The two opposing sub-communities are separated by ",". For each sub-community, it is a series of users. Letter u represent "user" and is fixed, C, D and E represent user Ids (numerical value). The sub-community can be of any number of users. In the last line, n is the total number of a-groups mined.

A.1.2 Synthetic Data Generator Configuration

Our synthetic data generator generates vote dataset according to our requirements. The generating program is DataGenerator.java. The input to the program is file "Dinput.txt". An example of Dinput.txt is:

G=6

L=2000

I=100000

U=50000

s=0.0004

The meanings of the parameters are shown in Table A.21. s is the probability of each user voting an item. Hence, U×s is the expected number of users voting an item.

Parameter	Meaning
G	Average size of potential closed a-groups
L	Number of potential closed a-groups
I	Number of items
U	Number of users
s	Probability of each user voting an item

Table A.21: Parameters of Synthetic Data Generator Input File

There are two outputs from the data generator. One is Dantigroup.txt. The first line of this file is the number of generated a-groups. The remaining lines are the generated a-groups used to generate the vote dataset. The second output is an input.txt which can be fed to the mining program. We may need to change the various parameters according to our needs.

A.2 Direct Antagonistic Group Mining Software

To perform direct a-group mining, we needs the software package "TRAJAN" and lcm.exe. Before running the software, we needs to have "cygwin" installed in the computer so as to run lcm.exe. We need first add the path of lcm.exe to system path. We can then open the "Trajan.sln" in Visual Studio. After we run the Form1.cs, a UI will pop up. In the "Min Size" input box, we can input the size threshold of a-groups. We can then click the "Combined Execution" button to get the a-groups. After execution, a pop up will show the running time and number of a-groups mined. The results are stored in outpair.txt under the directory specified by the Form1.cs. In outpair.txt, each line is a direct a-group. An example of the direct a-group is {A,B}-{C,D}, where the two sub-communities are separated by "-", A, B, C, D represent user Ids (numerical values).

Appendix B

Direct Antagonistic Group Mining Algorithm

A direct a-group has two basic requirements based on the positive and negative relationships. On one hand, each community must form positive network in the form of strongly connected component. On the other hand, members of one community must form negative relationships with all members of the other community. To mine for direct a-groups, we perform the following steps:

1. Project input positive-negative network g, to a graph g_t keeping only positive edges in graph g.

2. Extract SCCs from g_t of size more than the minimum support threshold min_size. These are candidate communities of direct a-groups. Nodes that are not part of SCCs of size at least min_size could not be part of any direct a-groups (see Property 5.1.2). We keep the set of nodes $N_+ = \{\text{n}|\text{n is a node in the identified SCCs}\}$.

3. Project the input positive-negative network g, to a graph g_d keeping only nodes in N_+ and negative edges.

4. Identify the set of maximal bi-cliques BCQ from g_d using Property 5.1.1.

5. For each bi-clique in BCQ with set of nodes nb, project the input positive-negative network g, to a graph g_{n_b} keeping only nodes in n_b and their positive edges. Each bi-clique in $\text{sbc}(g_{n_b})$ satisfying min_size is a direct a-group.

6. Eliminate redundant direct a-groups. There could still be redundant direct a-groups at the end of step 5. This is the case as although we mine for maximal bi-cliques at step 4, the direct a-groups are sub-bi-cliques of the maximal one. We iterate through the set of direct a-groups generated at step 5 and remove redundant ones based on Definition 5.1.12.

The following paragraphs describe the above steps in more details.

Pruning by Positive Relationship: Steps 1 and 2. First, we prune candidate nodes based on positive relationships. Negative edges are removed from the projected graph. Based on this positive relationship graph, our goal is to throw away nodes which is not part of any large enough positive relationship network. Due to the nature of the positive relationship network, the number of links the nodes in the network follow power law, i.e., most nodes are not connected to any other nodes. Hence, a large number of nodes could be removed from consideration.

To realize this goal, we employ Tarjan's algorithm [39], that could compute maximal SCCs by a single depth-first search pass on the positive relationship

network. Hence, it is very scalable as the runtime cost is linear to the size of the graph. We extract nodes that are part of a maximal SCCs with size $\geq$ min_size.

Pruning by Negative Relationship: Steps 3 and 4. At these steps, we focus on the *strong* (i.e., bi-directional) negative relationships. We project the input positive-negative network, by removing positive edges and non bi-directional negative edges. Two sub-communities in a direct a-group must form a bi-clique with respect to the bi-directional negative edges.

To realize the goal, we adapt a recent algorithm in [31] that extracts maximal bi-cliques from a graph following Property 5.1.1. The algorithm would return all maximal bi-cliques from the input bi-directional negative network.

Formation of Direct A-groups : Step 5. Each maximal bi-cliques mined at step 4 is not necessarily a direct a-group as each of the two sets in the bi-clique is not necessarily a positively connected community. A bi-clique could map to 0, 1, 2, or more direct a-groups.

Following Definition 5.1.4, every sub-biclique of the bi-clique is a bi-clique and hence satisfies the negative relationship requirement. Hence, we could extract sub-bi-cliques SBQ from each bi-clique in which each of the two sets of nodes form an SCS of size larger than min_size.

To realize this, we process each bi-clique BCQ identified in step 4. For each of the two sets of nodes in BCQ, i.e., BCQ.L and BCQ.R, we find SCSs on the projected positive relationship network containing nodes in BCQ.L/BCQ.R.

These operations would result in two sets of SCSs. Pairing one SCS from one set with another from another set, would form a direct a-group which could then be outputted.

Removal of Redundant Direct A-groups : Step 6. Usually, there are no or few redundant direct a-groups left at the end of step 5. However, there exist corner cases where redundant direct a-groups are present. This is the case as mined direct a-groups are sub-bi-cliques of the maximal bi-cliques mined at steps 3 and 4. We remove redundant direct a-groups by analyzing the list of direct a-groups mined at step 5 and detect for redundancies based on Definition 5.1.12.

The algorithm's pseudocode is shown in Algorithm 10.

Input: *min_size*: Minimum size threshold; G: positive-negative network
Output: direct a-groups with each group's size $\geq$ *min_size*
1 Let G_t = Project positive relationship network from G;
2 Let SCCList = Get maximal SCCs from the graph G_t by running [39];
3 Let $N_+ = \{\text{n} \mid \text{n} \in \text{s1} \wedge \text{s1} \in \text{SCCList} \wedge |\text{s1}| \geq min\_size\}$;
4 Let G_d = Project bi-directional negative relationship network in G for nodes in N_+;
5 Let T_d = GTD(G_d);
6 Let CP = Mine for closed itemsets from T_d with minimum support = min_size
7 **foreach** $p \in CP$ **do**
8 **if** $|p| \geq min\_size$ **then**
9 Let BC = Form bi-clique (p,map(p));
10 Let LT = Construct positive relationship SCCs from nodes in BC.L;
11 Let RT = Construct positive relationship SCCs from nodes in BC.R;
12 Remove SCCs from LT and RT with size < min_size;
13 **foreach** *pair* $l \in LT$ *and* $r \in RT$ **do**
14 Create a new direct a-group *dag* from l and r;
15 Add *dag* to Result;
16 **end**
17 **end**
18 **end**
19 Remove redundant direct a-groups from Result;
20 **Output** Result;

Algorithm 10: Direct A-group Mining Algorithm – *Dagmine*(*min_size*,G)