1-2016

On detecting maximal quasi antagonistic communities in signed graphs

Ming GAO
East China Normal University

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

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

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

DOI: https://doi.org/10.1007/s10618-015-0405-2

Citation

GAO, Ming; LIM, Ee-Peng; LO, David; and PRASETYO, Philips Kokoh. On detecting maximal quasi antagonistic communities in signed graphs. (2016). *Data Mining and Knowledge Discovery*. 30, (1), 99-146. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/2858

Published in Data Mining and Knowledge Discovery,
Volume 30, Issue 1, January 2016, Pages 99-146.
https://doi.org/10.1007/s10618-015-0405-2

On detecting maximal quasi antagonistic communities in signed graphs

Ming Gao · Ee-Peng Lim · David Lo · Philips Kokoh Prasetyo

Abstract Many networks can be modeled as signed graphs. These include social networks, and relationships/interactions networks. Detecting sub-structures in such networks helps us understand user behavior, predict links, and recommend products. In this paper, we detect dense sub-structures from a signed graph, called quasi antagonistic communities (*QAC*s). An antagonistic community consists of two groups of users expressing positive relationships within each group but negative relationships across groups. Instead of requiring complete set of negative links across its groups, a *QAC* allows a small number of inter-group negative links to be missing. We propose an algorithm, MASCOT, to find all maximal quasi antagonistic communities (*MQAC*s). MASCOT consists of two stages: pruning and enumeration stages. Based on the properties of *QAC*, we propose four pruning rules to reduce the size of candidate graphs in the pruning stage. We use an enumeration tree to enumerate all *strongly connected subgraphs* in a top–down fashion in the second stage before they are used to construct *MQAC*s. We have conducted extensive experiments using synthetic signed graphs and two real networks to demonstrate the efficiency and accuracy of the MASCOT

Responsible editor: Charu Aggarwal.

M. Gao (✉)
Institute for Data Science and Engineering, East China Normal University, Shanghai, China
e-mail: mgao@sei.ecnu.edu.cn

M. Gao · E.-P. Lim · D. Lo · P. K. Prasetyo
School of Information Systems, Singapore Management University, Singapore, Singapore
e-mail: eplim@smu.edu.sg

D. Lo
e-mail: davidlo@smu.edu.sg

P. K. Prasetyo
e-mail: pprasetyo@smu.edu.sg

algorithm. We have also found that detecting *MQAC*s helps us to predict the signs of links.

Keywords Signed graph · Bi-clique · Quasi antagonistic community · Enumeration tree · Power law distribution

1 Introduction

1.1 Motivation

The recent surge of online social networks and social media has radically changed the way social communities are studied. Traditional social science research defines social community to be a tightly knitted group of users in a social network of friendship links. In the last several decades, researchers have introduced several community definitions each with a distinct criteria for dense connectivities among the community's members. For example, community can be defined as a clique or quasi-clique, or a subgraph that contains much denser internal links than external links (Wasserman and Faust 1994; Palla et al. 2005). With these definitions, a large body of community detection algorithms have been developed.

In all these above-mentioned research, the common assumption is that there are only positive links among users in the social networks. Many of today's social networks are however signed graphs with positive and negative links. The positive links represent friendship or trust while the negative links represent foe or distrust. Although the traditional definitions of community and community detection algorithms are still applicable to these signed social networks by ignoring the negative links, there are interesting community structures including negative links that should be studied.

In this paper, we focus on pairs of *antagonistic sub-communities* such that users of the same sub-community share many positive relationships with one another, while users between a pair of antagonistic sub-communities have many negative relationships. A pair of antagonistic sub-communities may represent two political fractions (e.g., republicans vs democrats), supporters of two rival product brands (e.g., Apple vs Samsung), or fans of two competing artists (e.g., Justin Bieber vs Conor Maynard[1]). In these examples, users within the same sub-community enjoy positive relationships among themselves, while users from opposing sub-communities are likely to have negative relationships. Figure 1 depicts an example pair of antagonistic sub-communities, $\{u_1, u_2, u_3\}$ and $\{u_4, u_5, u_6\}$.

Antagonistic sub-communities in signed networks have not been studied much in the social science research literature. In the context of signed network, the social balance theory says that for any three users in triadic relationships, their triad is balanced when either only positive relationships exist among them, or one of them has negative relationships with the remaining two users who are positively related as shown in Fig. 2. A complete signed network therefore has exactly two user groups emerging when all users attempt to reduce cognitive dissonance among themselves by

[1] Justin Bieber and Conor Maynard are two teens who enjoy wide success in their singing career.

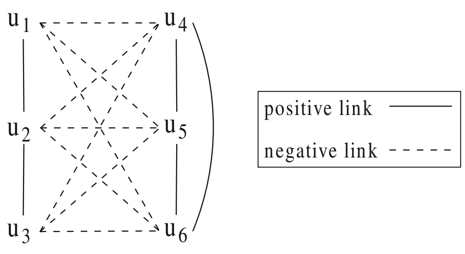


Fig. 1 An antagonistic community

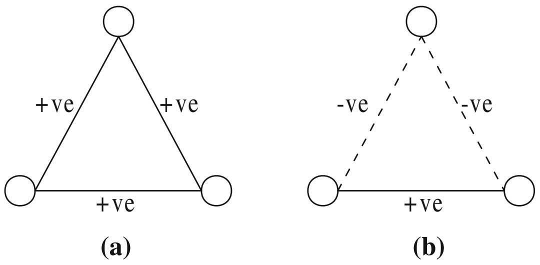


Fig. 2 Balanced triads

selecting the appropriate signs for their links with other users (Heider 1946; Cartwright and Harary 1956). Every user of a group will then have a positive link with every other user in the same group, but a negative link with every user from the other group. As most social networks are not complete, the above scenario can rarely be observed. Nevertheless, if the social balance theory is applied to only parts of a social network, one should find multiple pairs of antagonistic user groups in the network.

In this paper, we are therefore interested to find the antagonistic sub-communities as the localized effect of social balance theory. The sub-community level hostility may prevent the users across sub-communities from good collaboration and friendly interaction. The interaction among the members from opposing sub-communities may largely focus on topics that are contentious. The antagonistic user sub-communities can also have impact to their neighborhoods as more users may decide to join the antagonism. Given these negative implications, it becomes an important research task to discover antagonistic sub-communities and to intervene them as early as possible.

Beyond detecting them, one could use antagonistic sub-communities to predict link polarity, user preferences, and product adoption. In link prediction, the balanced triads in antagonistic sub-communities can be used to predict the link polarity (Heider 1946; Cartwright and Harary 1956). For user preference and product adoption prediction, antagonistic sub-communities can be used to infer the user preferences as the intra-community users of antagonistic sub-communities are more likely to share similar preferences. For example, Apple and Windows users might be antagonistic to one another, and thus they have different preferences in the products that they purchase or adopt. To the best of our knowledge, exploiting antagonistic sub-communities for predicting polarity, user preferences and product adoption is an entirely unexplored research territory. The obvious reason here is the lack of prior work on antagonistic sub-communities.

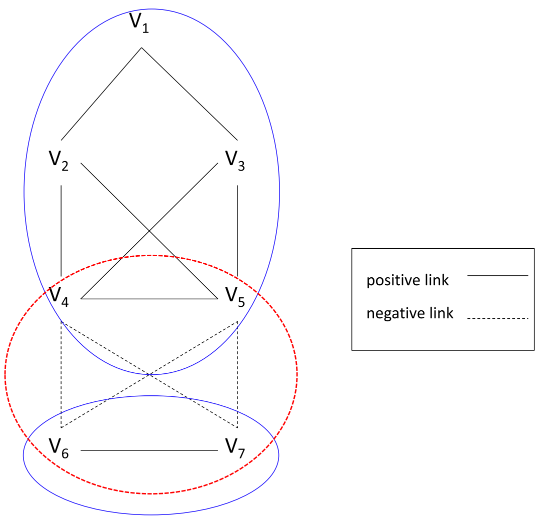


Fig. 3 Antagonistic sub-community as a local structure

An antagonistic sub-community is a local community structure as opposed to the global community structure that has been studied in many previous community discovery research (Girvan and Newman 2004; Karrer and Newman 2011; Ball et al. 2011; Ronhovde and Nussinov 2009). A global community structure is an optimal partition of the whole network, where each vertex belongs to at least one community. The partition maximizes some global objective functions, rather than local properties. In contrast, an antagonistic sub-community defines a dense subgraph associated with some local properties. It is therefore a local structure. For example, Fig. 3 shows a network where there is an antagonistic sub-community (in dashed red circle) involving vertices v_4, v_5, v_6, v_7 and these vertices belong to two global communities (in solid blue circle).

The previous works on antagonistic communities can be divided into *indirect* (Zhang et al. 2010, 2013) and *direct* (Lo et al. 2011, 2013) antagonistic sub-communities. The former is applicable to user-rate-item networks where a negative relationship exists between two users when they have significant disagreements in their ratings on some common items. A pair of indirect antagonistic sub-communities involves users of different sub-communities have conflicting ratings on the commonly rated items. Lo et al. defined direct antagonistic community (*DAC*) over a signed network (Lo et al. 2011, 2013). A *DAC* consists of two sub-communities. Each sub-community is a strongly connected subgraph w.r.t. positive links, and the two sub-communities form a bi-clique w.r.t. negative links. Due to the sparsity of social networks, missing links between users are very common. However, the bi-clique requirement is overly restrictive as a user from a sub-community may not interact with every user from the other sub-community. Hence, it is necessary to relax the condition on inter-community negative links.

1.2 Research objectives

In this paper, we introduce a new dense local structure called quasi antagonistic community (*QAC*) which consists of two connected sub-communities with positive links and two sub-communities form a quasi-biclique of negative links. With this definition, our objective is to detect all maximal quasi antagonistic communities (*MQACs*) from a given signed network since any *QAC* must be a subgraph of the corresponding *MQAC*s. We only mine maximal communities to reduce the number of mined communities. The number of small communities can be exponential to the number of maximal communities that contain them. Thus if we mine all the small communities, the number of communities will be too large to enumerate efficiently and output. Furthermore, many of these small communities would be very similar to one another.

The above problem is challenging as the maximum vertex quasi-biclique problem is NP-hard (Liu et al. 2008). Hence, computing *QAC*s is also a NP-hard problem because the number of *QAC*s to be examined grows exponentially with the number of edges. In addition, existing algorithm for detecting *DAC*s cannot apply to find *QAC*s as some members of a *QAC* may not fully connect to all members on the opposing sub-community.

We address the *MQAC*s detection problem using two main ideas, namely: (a) efficient pruning of search space, and (b) efficient enumeration of *MQAC* candidates. We summarize the main research contributions of this work as follows:

– We define *QAC*, a novel dense local structure, to model antagonistic community in a signed network. Compared with the earlier antagonistic community definition (Lo et al. 2011, 2013), *QAC* is less restrictive as it permits some missing negative links between its two antagonistic sub-communities. We derive two variants of *QAC*s, *absolute* and *relative quasi antagonistic communities*.
– We develop a novel algorithm called Mascot to detect all *MQAC*s in two stages: pruning stage and enumeration stage. In the pruning stage, we propose four pruning rules, namely *degree pruning*, *distance pruning*, *strongly connected component pruning* and *interaction graph pruning*, to reduce the size of candidate graphs to be used for generating *MQAC*s. These rules are all based on the *QAC* properties. In the enumeration stage, we enumerate all strongly connected subgraphs of a sub-community in a top–down manner, construct and verify the associated *MQAC*s.
– We conduct an extensive set of experiments on synthetic graphs and two real social networks to show the efficiency and effectiveness of our proposed Mascot algorithm. We also examine a set of example cases of *MQAC*s discovered from the real signed networks. In addition, we find that detecting *MQAC*s is helpful to predict signs of links in the signed networks.

1.3 Paper outline

The paper is organized as follows. We describe the related work in Sect. 2. In Sect. 3, we introduce the essential concepts and define the variants of quasi antagonistic community (*QAC*). Section 4 presents the steps of our proposed Mascot algorithm for finding all *MQAC*s. Sections 5 and 6 elaborate the pruning and enumeration steps in MASCOT respectively. Subsequently, Sect. 7 presents the algorithm to detect all *MQAC*s. Sections 8 and 9 cover our experimental studies on synthetic graphs and two real networks respectively. Finally, we give the concluding remarks in Sect. 10.

2 Related work

2.1 Community detection

Uncovering the community structure is crucial to understand the structure of complex and large networks. Many techniques have been proposed so far. These works in the literature can be grouped into two kinds of community structures.

The first kind considers global community structures. Such communities divide the entire graphs into smaller dense subgraphs. The literature offers many global criteria to identify the global communities. Examples are vertex similarity (Leicht et al. 2006), spectral algorithm (Donetti and Munoz 2004), modularity (Girvan and Newman 2004; Karrer and Newman 2011; Ball et al. 2011), and Potts model approach (Ronhovde and Nussinov 2009) etc. Among them, modularity has been frequently used to evaluate the goodness of community structure of networks. Louvain method (Blondel et al. 2008) is an efficient approach to detect communities by local optimal modularity score. Communities detection for a graph with million nodes using the Louvain method can take only a few minutes.

The second kind involves local community structures. Local community structures focus on subgraphs that satisfy specific local properties, but neglecting the rest of the graph. The corresponding structure are mostly maximal subgraphs, which cannot be enlarged by the addition of new vertices and edges without violating the local properties. A clique is defined in a very strict sense as subgraphs whose vertices are all adjacent to one another (Luce and Perry 1949). In the context of bipartite graph, a bi-clique is a complete subgraph of the bipartite graph (Groshaus and Szwarcfiter 2010). Triangles are the simplest cliques, and are frequent in real networks. But larger clique less frequent. The similar observation can be obtained for bi-clique.

It is however possible to relax the concept of clique, defining subgraphs which are still clique-like structures. The first kind of possibility is to use properties related to the existence of paths between vertices. An *n-clique* is a maximal subgraph such that the distance of each pair of its vertices is not larger than n (Alba 1973). Another two possible alternatives, the *n-clan* and the *n-club* are defined by Mokken (1979). An *n-clan* is an *n-clique* whose diameter is not larger than n (Mokken 1979; Luce 1950). An *n-club* is a maximal subgraph of diameter n (Mokken 1979; Jamali and Abolhassani 2006). The second kind of possibility is to restrict the adjacency of vertices. The idea is that a vertex must be adjacent to some minimum number of other vertices in the subgraph. A *k-plex* is a maximal subgraph in which each vertex is adjacent to all other vertices of the subgraph except at most k of them (Everett 1982). Similarly, a *k-core* is a maximal subgraph such that each vertex is adjacent to at least k other vertices of the subgraph (Giatsidis et al. 2011; Alvarez-Hamelin et al. 2008). The third kind of possibility is to restrict the density of subgraph. Comparing to clique, a quasi clique is a maximal subgraph in which density of the subgraph is not less than a pre-defined threshold (Abello et al. 2002; Liu and Wong 2008). A quasi bi-clique is to relax the adjacency of its vertices (Li et al. 2008).

2.2 Community structure on signed networks

Community detection on unsigned networks take into account only positively valued links. However, many actual networks also feature as signed networks with both positive and negative links. In the context of signed graph, the community structures can be also categorized into the global community structures and the local community structures.

For global community structures, Doreian and Mrvar (1996) detects communities from signed networks by optimizing frustration, where frustration is defined by the sum of the number of positive inter-community links and the number of negative intra-community links. Anchuri and Ismail propose a spectral approach that tries to maximize modularity and minimize frustration (Anchuri et al. 2012). Traag and Bruggeman further extend Potts model by adapting the concept of modularity to detect communities in signed networks (Traag and Bruggeman 2009). The paper by Mucha and Porter (2010) and Mucha et al. (2010) proposed an approach to detect community from arbitrary multi-slice networks, where each slice can be any kind of network. A signed network is treated as a two-slice network: positive slice and negative slice. Correlation clustering is to partition a complete signed network such that maximizes the number of positive links with clusters, plus the number of negative links between clusters (Bansal et al. 2004). But complete signed networks is very rare in real applications.

The global community structures of a network ignore the local structure of its vertices. The community is significantly different from local community structures, which this paper focuses. The closest to our work are our previous work (Zhang et al. 2010, 2013; Lo et al. 2011, 2013). Zhang et al. (2010, 2013) proposed an approach to mine antagonistic communities from rating networks. The work focuses on detecting conflicting ratings on some commonly rated objects and determining antagonistic communities from them. Lo et. al proposed an approach to mine antagonistic communities from explicit trust networks (Lo et al. 2011, 2013). The definition of antagonistic community is however very restrictive and does not work well when some links are missing or noisy. In this paper, we therefore define quasi antagonistic community by relaxing requirement of negative inter-sub-communities links.

3 Preliminary

Our quasi antagonistic community consists of a pair of sub-communities such that each sub-community is a strongly connected subgraph using positive links, and a quasi-biclique of negative links exist between the two sub-communities. In this section, we first introduce the concepts related to antagonistic community. We then formally define our problem.

Definition] 1 An **undirected signed graph** G is a triple (V, E^+, E^-), where V is a vertex set, E^+ and E^- represent positive edge set and negative edge set respectively.

Definition] 2 A strongly connected subgraph (SCS) of G is a subgraph G' such that there exists a series of edges in G' connecting every pair of vertices in G'.

Definition] 3 A strongly connected component (SCC) of G is a strongly connected subgraph that is maximal in size.

Definition] 4 The **positive neighborhood** of a vertex v in a signed graph $G = (V, E^+, E^-)$, denoted as $\Gamma^+(v)$, is defined as

$$\Gamma^+(v) = \{u|(v, u) \in E^+\};$$

We define the *projection of positive neighborhood* of the vertex v to a set of vertices $U \subset V$ w.r.t. positive links as

$$\Gamma^+_U(v) = \{u|(v, u) \in E^+ \wedge u \in U\};$$

The *positive neighborhood of a vertex set* S in a signed graph $G = (V, E^+, E^-)$, denoted as $\Gamma^+(S)$, is defined as

$$\Gamma^+(S) = \{u|(v, u) \in E^+ \wedge v \in S\}.$$

The definitions of *negative neighborhood* and *projection of negative neighborhood* of vertex v, and *negative neighborhood of a vertex set* S are defined in a similar manner.

We define an antagonistic community as a subgraph with two vertex sets densely linked with each other by negative links. Such a subgraph is also known as a quasi-biclique (QB) (Li et al. 2008). There are the absolute and relative versions of QB.

Definition] 5 Let V_1 and V_2 be two disjoint vertex sets and E be a set of edges between V_1 and V_2. $H = \langle V_1, V_2, E\rangle$ is a μ-**tolerant absolute QB** if for each $v \in V_i, i \in \{1, 2\}$,

1. v is disconnected from at most μ vertices in V_j, and
2. v is adjacent to at least μ vertices in V_j[2];

where $j \neq i$.

μ-**tolerant absolute QB** does not always look like a densely connected QB. Figure 4a depicts a μ-tolerant absolute QB with $\mu = 1$. Note that the members of each sub-community are not connected and the subgraph does not look like a community. We therefore propose (ϵ, min_size) **absolute QB** where ϵ and min_size are minimum threshold of missing links and minimum sub-community size respectively and $min\_size > 2\epsilon$.

[2] Another version of absolute QB that does not include condition (2) was proposed in a subsequent work (Sim et al. 2006).

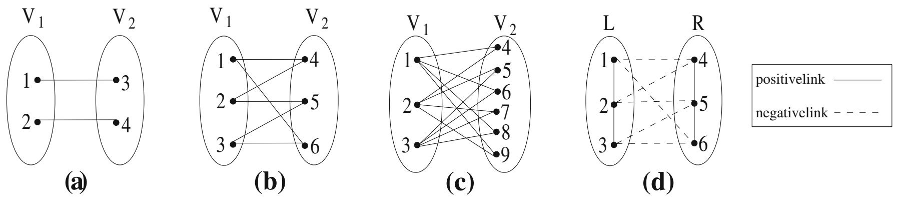


Fig. 4 **a** Disjoint *QB*, **b** (1,3) absolute *QB*, **c** $\left(\frac{1}{3}, 3\right)$ relative *QB* and **d** quasi antagonistic community

Definition] 6 Let V_1 and V_2 be two disjoint vertex sets (or sub-communities), E be a set of edges between V_1 and V_2, and $min\_size > 2\epsilon$. $H = \langle V_1, V_2, E\rangle$ is a (ϵ, min_size) **absolute QB** if for each $i \in \{1, 2\}$,

1. $|V_i| \geq min\_size$;
2. $|V_j - \Gamma_{V_j}(v)| \leq \epsilon$, for each $v \in V_i$ and $j \neq i$.

(ϵ, min_size) absolute *QB* requires each vertex in one vertex set of the biclique to be disconnected to at most ϵ vertices from another vertex set, and the condition $min\_size > 2\epsilon$ ensures that the biclique is connected. The proof of (ϵ, min_size) absolute *QB* is connected will be shown in Sect. 5.2. For example, Fig. 4a is not a (1,2) absolute *QB* because $min\_size \not> 2\epsilon$. Figure 4b depicts a (1,3) absolute *QB*.

Note that Fig. 4b is also a 1-tolerant absolute *QB*. It turns out that (ϵ, min_size) absolute *QB* is a special class of μ-tolerant absolute *QB* where $\epsilon = \mu$. According to definition of (ϵ, min_size) absolute *QB*, every vertex in a vertex set, say V_i, is connected to at least $|V_j| - \epsilon$ vertices in another vertex set V_j. Obviously, $|V_j| - \epsilon \geq min\_size - \epsilon > \epsilon$ because $min\_size > 2\epsilon$. Hence, a (ϵ, min_size) absolute *QB* must be a $\mu-$tolerant absolute *QB* when $\mu = \epsilon$. As ϵ is an absolute value, it may not work well for large size antagonistic communities. We now introduce the relative versions of *QB*.

Definition] 7 Let δ be a small value between 0 and 1, V_1 and V_2 be two disjoint vertex sets (or sub-communities) and E be a set of edges between V_1 and V_2. $H = \langle V_1, V_2, E\rangle$ is a δ**-tolerance relative QB** if for each $v \in V_i$, $i \in \{1, 2\}$, v is disconnected from at most $\delta \cdot |V_j|$ vertices in V_j where $j \neq i$ (Li et al. 2008).

Similar to the μ-tolerant absolute *QB* definition, δ-tolerant relative *QB* could be disconnected when $\delta \geq \frac{1}{2}$. For example, Fig. 4a depicts a disconnected $\frac{1}{2}$-tolerant *QB*. To allow only connected *QB*'s, we introduce the (δ, min_size) relative *QB*.

Definition] 8 Let V_1 and V_2 be two disjoint vertex sets, E be a set of edges between V_1 and V_2, and $\delta < \frac{1}{2}$. $H = \langle V_1, V_2, E\rangle$ is a (δ, min_size) **relative QB** if for each $i \in \{1, 2\}$,

1. $|V_i| \geq min\_size, i = 1, 2$;
2. $|V_j - \Gamma_{V_j}(v)| \leq \delta \cdot |V_j|$, for each $v \in V_i$ and $j \neq i$.

We require $\delta < \frac{1}{2}$ so as to ensure that a vertex is connected to most vertices from another vertex set and the graph stays connected. For example, Fig. 4b depicts a $\left(\frac{1}{3}, 3\right)$

relative *QB*. By definition, a (δ, min_size) relative *QB* must be a δ-tolerant relative *QB*.

According to Definitions 6 and 8, (ϵ, min_size) absolute *QB* is also a (δ, min_size) relative *QB* when $\delta \geq \frac{\epsilon}{min\_size}$. This is due to the fact that

$$\epsilon \leq \delta \cdot min\_size \leq \delta \cdot |V_j|; \tag{1}$$

$$|\Gamma_{V_j}(v)| > |V_j| - \epsilon \geq |V_j| - \delta \cdot |V_j| \geq (1-\delta)|V_j|. \tag{2}$$

On the other hand, a (δ, min_size) relative *QB* may not be a (ϵ, min_size) absolute *QB* when $\delta \geq \frac{\epsilon}{min\_size}$. For example, Fig. 4c depicts a $(\frac{1}{3}, 3)$ relative *QB* which is not a (1,3) absolute *QB*. Moreover, if a vertex set has fewer than 3 vertices, then ϵ and $\delta \cdot min\_size$ must be equal to 0.

In our example, the $\left(\frac{1}{3}, 3\right)$ relative *QB* in Fig. 4b can also be found in another larger $\left(\frac{1}{3}, 3\right)$ relative *QB* in Fig. 4c. Instead of finding all combinations of $\left(\frac{1}{3}, 3\right)$ relative *QB*, we introduce *maximal QB* in Definition 9.

Definition] 9 $H = \langle V_1, V_2, E\rangle$ is a **maximal QB** if there is no other *QB* $H' = \langle V_1', V_2', E'\rangle$ such that $H \neq H'$ and $V_1 \subseteq V_1'$, $V_2 \subseteq V_2'$ and $E \subseteq E'$.

For example, the graph in Fig. 4b is not a maximal 1-tolerant absolute *QB* because its supergraph is also 1-tolerant absolute *QB*. The graph in Fig. 4c is a maximal $\left(\frac{1}{3}, 3\right)$ relative *QB* if there is not supergraph that is a $\left(\frac{1}{3}, 3\right)$ relative *QB*.

Based on the definitions of *QB*, we now define *quasi-antagonistic community* for a signed graph in Definition 10.

Definition] 10 A *QB* = $\langle L, R, E\rangle$ is a quasi-antagonistic community (*QAC*) if L and R are *strongly connected subgraphs (SCSs)* involving positive edges only.

In this paper, we consider two versions of *QAC*, namely the *absolute quasi antagonistic community* (*aQAC*) if we use (ϵ, min_size) absolute *QB*, and *relative quasi-antagonistic community* (*rQAC*) if we use (δ, min_size) relative *QB*.

Figure 4d depicts an *aQAC* with (1,3) absolute *QB* for the negative edges, or a *rQAC* with $\left(\frac{1}{3}, 3\right)$ relative *QB* for the negative edges. The vertices in each vertex set (or sub-community) form a strongly connected subgraph by positive links.

The above *QAC* definition relaxes the direct antagonistic community (*DAC*) in our earlier paper (Lo et al. 2011) which requires the negative links between sub-communities to form a complete biclique.

As a *QAC* may be a subgraph of another larger *QAC*, we now define the *maximal quasi antagonistic community*.

Definition] 11 $H = \langle L, R, E\rangle$ is a maximal quasi antagonistic community (*MQAC*) if there is no other *QAC* $H' = \langle L', R', , E'\rangle$ such that $H \neq H'$ and $L \subseteq L'$, $R \subseteq R'$, and $E \subseteq E'$.

For example, the graph in Fig. 4d is a maximal *aQAC* with $min\_size = 3$ and $\epsilon = 1$ or a maximal *rQAC* with $min\_size = 3$ and $\delta = \frac{1}{3}$ because: (1) two sets of vertices are *SCS*s w.r.t. positive links only; (2) two sets of vertices form a maximal $(1, 3)$ absolute *QB* and a maximal $\left(\frac{1}{3}, 3\right)$ relative *QB* w.r.t. negative links.

Unless otherwise specified, *MQAC* denotes that maximal *aQAC* or maximal *rQAC* in this paper. We also use $\langle L, R\rangle$ to represent a *QB* $\langle L, R, E\rangle$ or a *quasi-antagonistic community (QAC)* $\langle L, R, E\rangle$ in the remainder of the paper to simplify the notations.

We now define the research problem of our work to be finding all *MQAC*s in a given signed graph. Solving this problem is challenging because the number of *MQAC*s can grow exponentially with the graph size. In this paper, we therefore aim to design an algorithm to find all maximal *aQAC*s and *rQAC*s efficiently. We will present our proposed solution framework in Sect. 4.

4 Mascot: a proposed framework for detecting *MQAC*s

Given an input signed graph, a naive method to return all *MQAC*s is to enumerate and test all subgraphs. The obvious disadvantage of this method is a very large number of candidate *MQAC*s which are costly to generate and test. We therefore propose a framework called Mascot that incorporates three processing steps grouped under pruning and enumeration stages as shown in Fig. 5.

There are four pruning rules that can be used to reduce the search space. The framework performs Step 1 that reduces an input signed graph according to *degree pruning* rule (Rule 1). The main idea is to remove vertices that do not satisfy some degree requirements. Given that the removal of a set of vertices may change the degree of other vertices, this pruning can be repeated till there are no more vertices that can be removed by the rule. Since each *MQAC* must be the subgraph of an induced graph as shown in Property 3 in Sect. 5, detecting *MQAC*s of a given signed graph is converted into detecting *MQAC*s from its induced graphs. For each vertex in the pruned input graph, we construct an induced graph which consists of a pair of left and right vertex sets. The induced graphs are further pruned using a combined set of pruning rules in Step 2 (combined pruning) before they are used as candidate graphs for computing

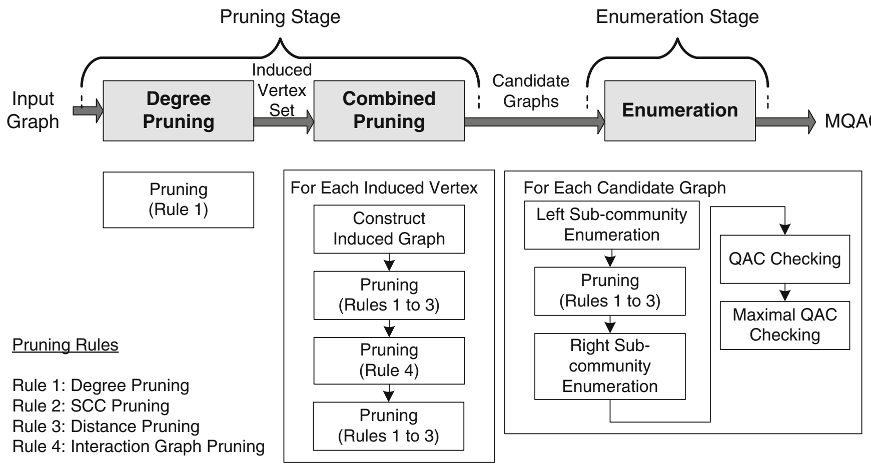


Fig. 5 Mascot framework

*MQAC*s in Step 3 (enumeration). These rules will be elaborated in Sect. 5. As we will show later, every *MQAC* is a subgraph of some pruned induced graphs (see Sect. 5.2).

In Step 3 (enumeration), we want to efficiently enumerate all *QAC*s in each candidate graph. The naive approach is to list all subgraphs of a candidate graph and verify if they are *MQAC*s. Even when candidate graphs are smaller, this approach is not efficient because the number of subgraphs in a candidate graph can still be large. Since each sub-community of a *QAC* forms a *SCS* w.r.t. positive edges, we therefore enumerate all connected subgraphs of each vertex set in a candidate graph. We first enumerates all the *SCS*s of the left vertex set of the candidate graph. After applying the pruning rules possibly repeatedly, we obtain new subgraphs with *SCS*s in both left and right vertex sets. These subgraphs are verified to demonstrate the *MQAC*'s properties before they are returned as the final results.

5 Pruning rules

In this section, we present the pruning rules used in Step 1 (degree pruning) and Step 2 (combined pruning) of the MASCOT framework. The four pruning rules used are: (i) *degree pruning*, (ii) *strongly connected component (SCC) pruning*, (iii) *distance pruning*, and (iv) *interaction graph pruning*. In the following, we elaborate each of these rules.

5.1 Input graph pruning

We first reduce the input graph by removing vertices and edges that are not required in finding *QAC*s. If a vertex v is contained in a *QAC*, it must have some neighbors connected by positive links and negative links.

Property 1 *Every vertex v in a QAC with $min\_size > 1$ must have $deg^+(v) \geq 1$, where $deg^+(v)$ denotes the positive degree of vertex v.*

As each sub-community of a *QAC* is a *SCS*, there is a path with positive links between any two vertices in the same sub-community. Thus, the positive degree of a vertex cannot be 0.

Property 2 *1. If v is a vertex in an aQAC, then $deg^-(v) \geq min\_size - \epsilon$;*
2. If v is a vertex in a rQAC, then $deg^-(v) \geq min\_size(1 - \delta)$;
where $deg^-(v)$ denotes the negative degree of vertex v.

The above properties define the lower bounds of the negative and positive degrees in a *QAC*. Due to the nature of scale-free networks, both the positive and negative degrees follow power law distribution (Dandekar 2010; Beyene et al. 2008). In other words, a large number of vertices could be removed as they do not satisfy Properties 1 and 2. Based on these properties, we propose the *degree pruning* rule in Pruning Rule 1.

Pruning Rule 1 *(Degree pruning rule) Given a signed graph $G(V, E^+, E^-)$, we remove a vertex v and its edges if*

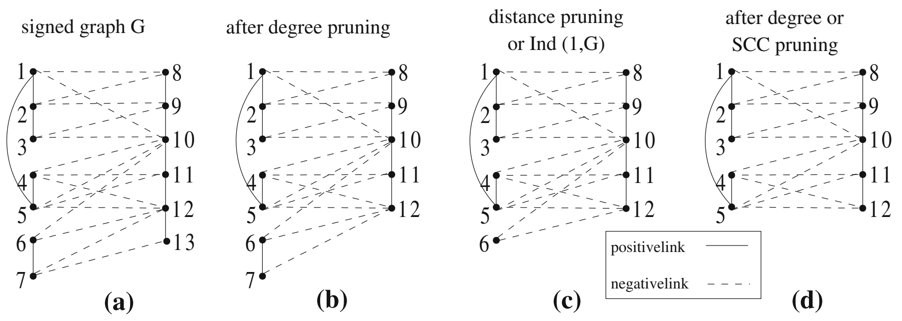


Fig. 6 Example of applying pruning rules

- *For aQAC:*
  1. $deg^+(v) = 0$*; or*
  2. $deg^-(v) < min\_size - \epsilon$.
- *For rQAC:*
  1. $deg^+(v) = 0$*; or*
  2. $deg^-(v) < min\_size(1 - \delta)$.

Proof According to Properties 1 and 2, the correctness of this pruning rule can be derived easily. □

For example, Fig. 6a is an input signed graph and we want to find *aQAC*s with $min\_size = 3$ and $\epsilon = 1$. Using Pruning Rule 1, vertex 13 is removed since its negative degree is smaller than $min\_size - \epsilon = 3 - 1 = 2$ as shown in Fig. 6b. The removal of vertex 13 will cause vertex 7 to be removed as its negative degree is now smaller than $min\_size - \epsilon$.

5.2 Induced graph pruning

Every vertex may be a member of a *QAC* after the above pruning step. Hence, all *QAC*s can be obtained from combining the *QAC*s involving different vertices. For each vertex, we induce a candidate graph and compute all *QAC*s involving the vertex within the candidate graph. We want to keep the candidate graphs small for efficient computation.

5.2.1 Induced graph

We first define for each vertex $v \in V$ in a signed graph $G = (V, E^+, E^-)$ three $k-$th hop negative vertex neighbor sets $S_1(v)$, $S_2(v)$ and $S_3(v)$ based on the negative distance from v as follows:

$$S_1(v) = \Gamma^-(v) \setminus \{v\}, \tag{3}$$

$$S_2(v) = \Gamma^-(S_1(v)), \tag{4}$$

$$S_3(v) = \Gamma^-(S_2(v)) \setminus \{v\}. \tag{5}$$

From the above three vertex sets, we define the candidate graph for computing *QAC*s as an induced graph as follows.

Definition] 12 Given a vertex v of a signed graph $G = (V, E^+, E^-)$, the **induced graph** of v is a signed graph $Ind(v, G) = (\bigcup_{i=1}^{3} S_i(v), E^{+'}(v), E^{-'}(v))$, where

$$E^{+'}(v) = \{(u_1, u_2)|(u_1, u_2) \in E^+, u_1, u_2 \in S_1(v) \cup S_3(v)\} \\ \cup\{(u_1, u_2)|(u_1, u_2) \in E^+, u_1, u_2 \in S_2(v)\}, \tag{6}$$

$$E^{-'}(v) = \{(u_1, u_2)|(u_1, u_2) \in E^-, u_1, u_2 \in \bigcup_{i=1}^{3} S_i(v)\}. \tag{7}$$

For simplicity, we may represent $Ind(v, G)$ by its two vertex sets $S_2(v)$ and $S_1(v) \cup S_3(v)$, i.e., $Ind(v, G) = \langle S_2(v), S_1(v) \cup S_3(v)\rangle$. Formula 6 says that positive edges exist in each vertex set, not across the two vertex sets. Meanwhile, Formula 7 says that negative links exist between any pair of vertices. For the example in Fig. 6a, the induced graph $Ind(1, G)$ is shown in Fig. 6c and the left and right vertex sets of $Ind(1, G)$ are:

$$S_2(1) = \{1, 2, 3, 4, 5, 6\}$$
$$S_1(1) \cup S_3(1) = \{8, 10\} \cup \{8, 9, 10, 11, 12\}$$

The interesting property of the induced graph is that every *QAC* is a subgraph of some induced graphs. Before we formally introduce this in Property 3, we first present the following Lemmas 1 and 2.

Lemma 1 *If vertices v_i and v_j come from the same vertex set of a QAC, then they are adjacent to at least one common vertex in the other vertex set w.r.t. negative links.*

Proof Firstly, we consider the *aQAC* $\langle L, R\rangle$. Without loss of generality, suppose that $v_i, v_j \in L$. Both $\Gamma_R^-(v_i)$ and $\Gamma_R^-(v_j)$ have size no smaller than $|R| - \epsilon$. Since $|R| \geq min\_size > 2\epsilon$, we have

$$|\Gamma_R^-(v_i)| + |\Gamma_R^-(v_j)| \geq 2|R| - 2\epsilon > |R|; \tag{8}$$

Thus, $\Gamma_R^-(v_i)$ and $\Gamma_R^-(v_j)$ share at least one common vertex.

For *rQAC* with $\delta < \frac{1}{2}$, we can also obtain a similar property as shown in Eq. 9.

$$|\Gamma_R^-(v_i)| + |\Gamma_R^-(v_j)| \geq 2(1 - \delta)|R| > |R|; \tag{9}$$

Lemma 2 *(Constraint on Negative Distance) Suppose a signed graph $G(V, E^+, E^-)$ be a QAC. For any pair of vertices $v_i, v_j \in V$, the following constraint holds*

$$dist^-(v_i, v_j) = \begin{cases} 1 \; or \; 3, & if\, v_i \text{ and } v_j \text{ come from different vertex sets;} \\ 2, & if\, v_i \text{ and } v_j \text{ come from the same vertex set.} \end{cases}$$

where $dist^-(v_i, v_j)$ denotes the length of shortest negative paths between v_i and v_j.

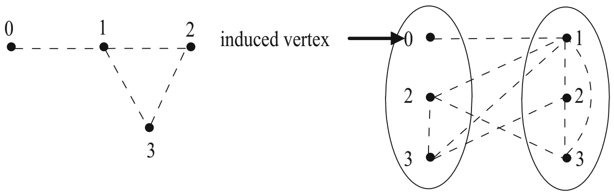


Fig. 7 Induced graph with common vertices

Proof Assume v_i and v_j are from the same vertex set, and without any loss of generality, suppose $v_i, v_j \in L$. According to the Lemma 1, there is at least one vertex $v_0 \in R$, which is adjacent to vertices v_i and v_j in L. Hence, we have $dist^-(v_i, v_j) = 2$.

Assume v_i and v_j are from different vertex sets, and suppose $v_i \in L$, and $v_j \in R$. The negative distance between v_i and v_j is 1 if they are connected. Otherwise, according to definition of *QB*, there is at least one vertex v_0 in L, which is adjacent to vertex $v_j \in R$. Furthermore, according to Lemma 1, $v_0 \in R$ and $v_i \in R$ should have at least one common neighbor in L. Thus, we have a shortest path from vertex v_i to vertex v_j with negative distance = 3. □

We can therefore conclude that any pair of vertices in a *QAC* are connected w.r.t. negative links.[3] Next, we shall state Property 3 as follows:

Property 3 *For each vertex $v \in V$ of a signed graph G, if it is contained in a QAC, then the QAC must be a subgraph of Ind(v,G).*

Proof Let $q = \langle L, R \rangle$ be a (ϵ, min_size) absolute or (δ, min_size) relative *QB* and vertex v be in the *QB*. Without loss of generality, let $v \in L$. For any other vertex $v_l \in L$, we have $dist^-(v, v_l) = 2$. Thus $v_l \in S_2(v)$.

According to Lemma 2, for each vertex $v_r \in R$, we have $v_r \in S_1(v)$ or $v_r \in S_3(v)$. That is to say, $v_r \in S_1(v) \cup S_3(v)$. Thus, the *QAC* containing vertex v must be a subgraph of $Ind(v, G)$.

Based on Property 3, a QAC only involves 3-hop vertices of the induced vertex. Given an induced vertex, k-hop ($k > 3$) vertices can be removed from the input graph for finding all QACs involving the induced vertex.

In addition, there may be vertices duplicated in both left and right vertex sets. This occurs when an unbalanced triangle appears in an induced graph. For example, in Fig. 7, vertices 1, 2 and 3 form an unbalanced triangle. Hence, the vertices 2 and 3 are duplicated in the induced graph of vertex 0.

5.2.2 Degree pruning

Comparing to the input signed graph, an induced graph may remove the vertices that are far away from the induced vertex w.r.t. negative links. We can therefore apply

[3] Connectivity of any pair of vertices in a (ϵ, min_size) absolute *QB* or (δ, min_size) relative *QB* can be proven in a similar manner.

degree pruning rule to prune the induced graph since removal of a vertex affects the degrees of the others. A vertex can be removed from an induced graph if its degrees are smaller than the lower bounds of the positive and negative degrees of a *QAC*. For example, Fig. 6c depicts *Ind(1, G)*, where *G* is a signed graph shown in Fig. 6a. Vertex 6 can be removed since its positive degree is 0.

5.2.3 SCC pruning

Each vertex set of a *QAC* must belong to the same *SCC* involving positive links only. We can therefore use the property to further reduce a candidate graph. This is achieved by the Pruning Rule 2 (SCC pruning) below.

Pruning Rule 2 *(SCC pruning). Let the induced graph of a vertex v Ind(v,G) be* $H = \langle L, R\rangle$. *All vertices of a SCC can be removed from H if:*

1. *The SCC overlaps within vertex set L and it does not contain v; or*
2. *The size of the SCC is less than* min_size.

As we wants to extract from $Ind(v, G)$ only the *QAC*s containing the induced vertex v. According to Definition 12, v is contained in the vertex set L. Therefore, only the *SCC* containing v should be retained. In addition, the vertex set of a *QAC* must be a subset of a *SCC*. As the definition of *QAC* requires each sub-community to have no less than min_size vertices, a *SCC* can be removed if its size is smaller than min_size.

The first condition of the rule guarantees that every candidate graph generated from an induced graph contains the induced vertex. The second condition says that the *SCC*s in L or R must have at least min_size vertices. Suppose $min\_size = 3$, and $\epsilon = 1$. Consider the induced graph of the induced vertex 1 in Fig. 6c, vertex 6 can be removed by both conditions because: (i) *SCC* $\{6\}$ does not contain the induced vertex 1; and (ii) size of *SCC* $\{6\}$ is smaller than min_size.

In addition, we can partition an induced graph $Ind(v, G) = \langle L, R\rangle$ into multiple candidate graphs $\{\langle C_l, C_r^i\rangle\}_{i=1}^k$ if there are k *SCC*s ($\{C_r^i\}_{i=1}^k$) in R with sizes no less than min_size, $v \in C_l \subset L$ and C_l is a *SCC* with size no less than min_size. In the remainder of this paper, the *SCC pruning* rule consists of Pruning Rule 2 and this reduction operation.

5.2.4 Distance pruning

Distance pruning is inspired by the maximum distance between the two vertices in a *QAC* using negative links. According to Lemma 2, vertices v and u cannot belong to the same *QAC* if the negative distance between v and u is larger than 3. The latter situation may occur in an induced graph when some of its vertices are removed from the graph due to *degree pruning* and *SCC pruning*. Hence, we derive the *distance pruning* rule in Pruning Rule 3.

Pruning Rule 3 *(Distance pruning) Let* $G_I^{'}$ *be subgraph of* $Ind(v, G)$ *after degree and SCC pruning. A vertex u can be removed from* $G_I^{'}$ *if* $dist^{-}(v, u)$ *in* $G_I^{'}$ *is larger than 3.*

For example, Fig. 6a depicts a subgraph of $ind(1, G)$ after some pruning. According to Pruning Rule 3, vertex 7 is removed from the graph since the negative distance between vertex 7 and vertex 1 is larger than 3. Similarly, vertex 13 is removed from the graph.

5.2.5 *Interaction graph pruning*

There are further room for pruning even with Pruning Rules 1 to 3. Consider the input signed graph induced by vertex 1 in Fig. 6a, $min\_size = 3$ and $\epsilon = 1$. We can get a candidate graph in Fig. 9a after pruning after applying *degree pruning*, *SCC pruning* and *distance pruning*. From this candidate graph, we can find only a *QAC* $\langle\{1, 2, 3\}, \{8, 9, 10\}\rangle$. There are therefore several vertices that can be removed from the candidate graph. For example, the negative distance between two vertices 8 and 11 in Fig. 9a is larger than 3, i.e., the two vertices cannot be in the same *QAC*. Based on this observation, we propose a new pruning rule based on the notion of interaction graph which is defined in terms of the distance w.r.t. negative links between any two vertices from a candidate graph. Before we define the interaction graph, we define the *surviving induced graph* in Definition 13.

Definition] 13 The **surviving induced graph** of vertex v, $sInd(v, G) = (V^{'}, E^{+^{'}}, E^{-^{'}})$, is a subgraph of $Ind(v, G)$, if $v \in V^{'}$ and no vertex in $V^{'}$ can be pruned away using *degree*, *SCC* and *distance pruning* rules.

We can easily infer that $sInd(v, G)$ is a super-graph of all *QAC*s involving vertex v. Based on $sInd(v, G)$, we define the *interaction graph* in Definition 14.

Definition] 14 Let $G = (V, E^+, E^-)$ be a signed graph. The **interaction graph** of G, denoted by $Inter(V, G)$, is a graph (V_I, E_I) where $V_I = V$, and

$$E_I = \{(u, v)|v \in sInd(u, G) \wedge u \in sInd(v, G) \wedge u \neq v\}, \tag{10}$$

We call E_I the set of **interaction links**. Specifically, for any vertex set $S \subset V$, $Inter(S, G) = (S, E_I^{'})$ is defined as a subgraph of $Inter(V, G)$ if

$$E_I^{'} = \{(u, v)|u, v \in S \wedge v \in sInd(u, G) \wedge u \in sInd(v, G) \wedge u \neq v\}. \tag{11}$$

Intuitively, two vertices are connected by an interaction link in the interaction graph if the distance between them w.r.t. negative links of a signed graph is no larger than 3 and they cannot be removed by another pruning rules. For example, consider the signed graph shown in Fig. 8a. The graph forms an *aQAC* with $min\_size = 3$ and $\epsilon = 1$ (or a *rQAC* with $min\_size = 3$ and $\delta = \frac{1}{3}$). The interaction graph of G is a clique as shown in Fig. 8b as every vertex u can be found in $sInd(v, G)$. We prove this property of *interaction graph* of a *QAC*.

Property 4 *Let $H = \langle L, R\rangle$ be a QAC. Both $Inter(L, H)$ and $Inter(R, H)$ are cliques of size $|L|$ and $|R|$ respectively.*

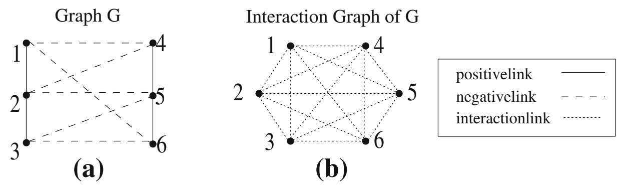


Fig. 8 Example of interaction graph

Proof Without loss of generality, assume any two vertices $u, v \in L$. According to Lemma 2 and Property 3, we have $dist(u, v) = 2$, $v \in sInd(u, H)$ and $u \in sInd(v, H)$ (Note that $Ind(v, H) = sInd(v, H)$). (v, u) therefore is an interaction link in $Inter(L, H)$. Hence, $Inter(L, H)$ is a clique of size $|L|$. For the same reason, $Inter(R, H)$ is a clique of size $|R|$. □

With Property 4, all vertices of a *QAC* in the interaction graph should form a clique of size $|L| + |R|$. In addition, L and R form a biclique. For a candidate graph induced by vertex v, we can remove a vertex u if all cliques or bicliques involving u do not contain v or their sizes cannot satisfy the min_size requirement. Note that finding all cliques and bicliques of $Inter(V, G)$ is more costly than finding all cliques of $Inter(L, G)$ or $Inter(R, G)$. In our next Pruning Rule 4, we find all maximal cliques of $Inter(L, H)$ and $Inter(R, H)$ to prune vertices.

Pruning Rule 4 *(Interaction graph pruning) Let signed graph $H = \langle L, R \rangle$ be a subgraph of Ind(v,G) and L contain induced vertex v. A vertex u can be removed from H if:*

1. *Its degree in $Inter(L, H)$ ($Inter(R, H)$) is smaller than $min\_size - 1$ if $u \in L$ ($u \in R$).*
2. *Every maximal clique involving vertex u in $Inter(L, H)$ does not contain the induced vertex v or the clique size is less than min_size; or*
3. *The size of every maximal clique involving vertex u in $Inter(R, H)$ is less than min_size.*

Proof Note that the induced vertex v is in $Inter(L, H)$. For the vertex u of $Inter(L, H)$, if every maximal clique involving vertex u in $Inter(L, H)$ does not contain the induced vertex v, then vertices v and u are disconnected in $Inter(L, H)$, i.e., vertices v and u do not exist in the same *QAC* which satisfies min_size. Therefore, vertex u can be removed from finding all *QAC*s contained the induced vertex v.

According to Definition 14 and Property 4, only vertices appearing in a maximal clique of $Inter(L, H)$ or $Inter(R, H)$ may be contained in the same *QAC*. If the size of every maximal clique of $Inter(L, H)$ or $Inter(R, H)$ involving a vertex u is less than min_size, then every *QAC* contained vertex u does not satisfy the min_size requirement. Therefore, vertex u can be removed from H.

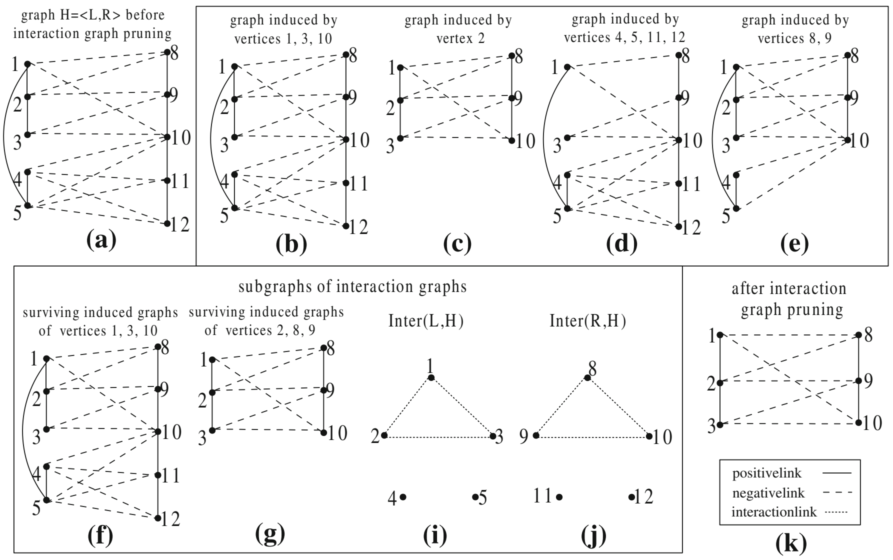

graph H=<L,R> before interaction graph pruning
graph induced by vertices 1, 3, 10
graph induced by vertex 2
graph induced by vertices 4, 5, 11, 12
graph induced by vertices 8, 9
(a)
(b)
(c)
(d)
(e)
subgraphs of interaction graphs
surviving induced graphs of vertices 1, 3, 10
surviving induced graphs of vertices 2, 8, 9
Inter(L,H)
Inter(R,H)
(f)
(g)
(i)
(j)
after interaction graph pruning
positivelink
negativelink
interactionlink
(k)

Fig. 9 Example of interaction graph pruning

Vertex u is not contained in any cliques with sizes no less than min_size if its degree is smaller than $min\_size - 1$. Therefore, vertex u can be also removed from H. □

This pruning rule says that vertex set of a maximal clique of $Inter(L, H)$ or $Inter(R, H)$ becomes a candidate vertex set of a *QAC* if its size is no less than min_size. Vertices of $Inter(L, H)$ are therefore discarded if their degrees in $Inter(L, H)$ are smaller than $min\_size - 1$. The same criteria can be applied to the vertices of $Inter(R, H)$. In addition, the maximal clique contains the induced vertex if the former is found in $Inter(L, H)$. Furthermore, a signed graph may decompose into multiple candidate graphs if multiple maximal cliques in $Inter(L, H)$ or $Inter(R, H)$ survive after this pruning rule. Formally, let $\{C_i^L\}_{i=1}^{k_l}$ ($\{C_j^R\}_{j=1}^{k_r}$) denote the set of maximal cliques of $Inter(L, H)$ ($Inter(R, H)$) after interaction graph pruning. Every pair of vertex sets C_i^L and C_j^R forms a candidate graph $\langle C_i^L, C_j^R \rangle$. We obtain $k_l \times k_r$ candidate graphs. In the remainder of this paper, the *Interaction graph pruning* rule consists of Pruning Rule 4 and this reduction operation.

```
Algorithm 1: Function $combinedPruning(v, Ind(v, G), min\_size, \epsilon)$
   Input: Induced graph $Ind(v, G)$, induced vertex $v$, parameters $min\_size$ and $\epsilon$;
   Output: $candGraphSet$: candidate graph set (initialized to be $\emptyset$);
1  $graphSet_1 \leftarrow repeatedPruning(v, Ind(v, G), min\_size, \epsilon)$ //integrate Pruning Rules 1-3;
2  $graphSet_2 \leftarrow interactionPruning(v, graphSet_1, min\_size, \epsilon)$ //interaction graph pruning;
3  for each candidate graph $cand \in graphSet_2$ do
4      $graphSet_3 \leftarrow repeatedPruning(v, cand, min\_size, \epsilon)$;
5      $candGraphSet \leftarrow candGraphSet \cup graphSet_3$;
6  end
7  return $candGraphSet$
```

For example, the signed graph $H = \langle L, R\rangle$ induced by vertex 1 in Fig. 9a has 5 vertices in both L and R. We show the induced graphs of every vertex of H in Fig. 9b–e. It is easy to check that the induced graphs for vertices 4, 5, 11 and 12 do not survive after *degree pruning*, *SCC pruning* and *distance pruning*. Only the graphs in Fig. 9f and g remain. There is no interaction link between vertices 1 and 4 because $1 \notin sInd(4, H)$ after pruning. From these pruned induced graphs, we construct $Inter(L, H)$ and $Inter(R, H)$ as shown in Fig. 9i and j.

According to Property 4, only the vertex sets $\{1, 2, 3\}$ and $\{8, 9, 10\}$ may form a *QAC* with $min\_size = 3$ and $\epsilon = 1$. Hence, we obtain a candidate graph as shown in Fig. 9k.

5.3 Combined pruning rules for an induced graph

Algorithm 2: Function $repeatedPruning(v, G, min\_size, \epsilon)$

```
    Input: Induced graph $G$, induced vertex $v$, parameters $min\_size$ and $\epsilon$;
    Output: $graphSet$: set of candidate graphs (initialized to be $\emptyset$);
1   insert $G$ into a queue $Q$;
2   while $Q$ is not empty do
3       $g \leftarrow$ remove the top graph from $Q$;
4       $prunedGraph \leftarrow degreePruning(g, min\_size, \epsilon)$ //degree pruning;
5       $prunedGraphSet \leftarrow sccPruning(prunedGraph, min\_size)$ // SCC pruning;
6       for each graph $g' \in prunedGraphSet$ do
7           $prunedGraph \leftarrow distancePruning(v, g', min\_size)$ //distance pruning;
8           if $prunedGraph = g$ // no vertex in g can be pruned by three pruning rules then
9               $graphSet \leftarrow graphSet \cup \{g\}$ //graph $g$ survives after three pruning rules;
10          end
11          else insert graph $prunedGraph$ into the queue $Q$;
12          ;
13      end
14  end
15  return $graphSet$
```

In this section, we combine four pruning rules to iteratively reduce the size of an induced graph as much as possible. $combinedPruning(v, Ind(v, G), min\_size, \epsilon)$ is the combined pruning operation shown in Algorithm 1, which takes a graph $Ind(v, G)$ induced by vertex v with parameter settings min_size and ϵ.

The function $repeatedPruning(v, G, min\_size, \epsilon)$ at Lines 2 and 5 combines *degree*, *SCC* and *distance pruning* rules together to reduce the size of G. Note that during repeated pruning, the function may generate many signed graphs of smaller sizes. The function terminates when every signed graph cannot be reduced further. In the combined pruning operation, interaction graph pruning $interactionPruning(v, temp, min\_size, \epsilon)$ is performed only once at Line 3 because the pruning rule employs Bron Kerbosch Algorithm to find all maximal cliques (Coen and Joep 1973). The algorithm dominates the major cost of pruning stage.

The function $repeatedPruning(v, G, min\_size, \epsilon)$ is shown in Algorithm 2. This algorithm takes a signed graph G induced by vertex v and adds it to a queue Q at Line 1. The algorithm reduces the size of the first graph of Q by degree pruning, *SCC* pruning and distance pruning at Lines 4, 5 and 7, respectively. Note that an induced

Table 1 Complexity analysis in the pruning stage

Component	Input	Time complexity
Compute induced graph	$(V^{'}, E^{+^{'}}, E^{-^{'}})$	$O(\vert E^{-^{'}}\vert)$
Pruning Rule 1	$(V^{'}, E^{+^{'}}, E^{-^{'}})$	$O(\vert V^{'}\vert + \vert E^{+}\vert + \vert E^{-}\vert)$
Pruning Rule 2	$(V^{'}, E^{+^{'}}, E^{-^{'}})$	$O(\vert V^{'}\vert + \vert E^{+^{'}}\vert)$ (Tarjan 1972)
Pruning Rule 3	$(V^{'}, E^{+^{'}}, E^{-^{'}})$	$O(\vert E^{-^{'}}\vert)$
Interaction graph construction	$(V^{'}, E^{+^{'}}, E^{-^{'}})$	$O(\vert V^{'}\vert \times \vert E^{-}\vert)$
Pruning Rule 4	$(V^{'}, E^{+^{'}}, E^{-^{'}})$	$O(3^{\vert V^{'}\vert/3})$ (Moon and Moser 1965)

graph may break up into smaller graphs after *SCC* pruning since *SCC* pruning keeps one *SCC* of left side and all *SCC*s of right side with size no less than min_size. In the implementation, we employ the Tarjan's Algorithm (Tarjan 1972) to find all *SCC*s of a vertex set. A signed graph g is added into the candidate graph set $graphSet$ if it cannot be pruned further at Line 9. Otherwise, the signed graph is added into the queue Q at Line 11. Algorithm 2 runs until no signed graph can be pruned further by the three pruning rules.

5.4 Complexity analysis

Let $(V^{'}, E^{+^{'}}, E^{-^{'}}) \subset G$ be the input graph on which pruning rules are to be applied. The complexities of them are shown in Table 1. Given an induced vertex, computing the induced graph and the Pruning Rule 3 searches 3-hop negative neighbors of the induced vertex, and its complexity is $O(|E^{-^{'}}|)$. Next, the complexities of computing both positive and negative degrees in Pruning Rule 1 are $O(|V^{'}|+|E^{+^{'}}|)$ and $O(|V^{'}|+|E^{-^{'}}|)$. The complexity of further checking degree of every vertex is $O(|V|)$. The overall complexity of Pruning Rule 1 is therefore $O(|V^{'}|+|E^{+}|+|E^{-}|)$. We employ Tarjan's algorithm to find all *SCS*s of a vertex set in the Pruning Rule 2. Its complexity is therefore $O(|V^{'}| + |E^{+^{'}}|)$ (Tarjan 1972). To sum up, the complexity of repeated pruning is $O(|V^{'}| + |E^{+}| + |E^{-}|)$.

To construct an interaction graph, we need to compute $|V^{'}|$ induced graphs. Hence, the complexity of computing the interaction graph is therefore $O(|V^{'}| \times |E^{-}|)$. According to Moon and Moser (1965), a graph can have at most $3^{|V^{'}|/3}$ maximal cliques. The Bron–Kerbosch algorithm can be shown to have worst-case running time $O(3^{|V^{'}|/3})$ (Moon and Moser 1965).

We will further present efficiency of our pruning rules and the whole algorithm using synthetic and real data in Sects. 8 and 9. Even though the complexity of Pruning Rule 4 is exponential to the graph size, it does not take too much time complete in practice due to the earlier steps. Astute readers may question why we propose this pruning rule. We will leave the answer to the next section that elaborates on the enumeration stage.

In addition, these pruning rules are not ad-hoc. Except for *SCC* pruning, the degree pruning, distance pruning and interaction graph pruning can be used to speed up detecting bicliques (Groshaus and Szwarcfiter 2010) and quasi-bicliques in Definitions 6 and 8 ((ϵ, min_size) **absolute QB** and (δ, min_size) **relative QB**) from a bipartite graph since these pruning rules only involve the edges between different sub-communities. Furthermore, all pruning rules can be used to detect *DAC* (Zhang et al. 2010, 2013; Lo et al. 2011, 2013).

6 Enumeration of maximal quasi-antagonistic communities

We now describe an efficient enumeration of all *MQAC*s within a candidate graph. We propose a data structure, called *top–down enumeration tree*, to list all *SCS*s of each vertex set of a candidate graph. We then verify each enumerated graph if it satisfies the criteria of a *QAC*.

6.1 Enumeration tree

Enumeration tree is a data structure that has been used to list all subgraphs of a given graph (Johnson et al. 1988). For example, Fig. 10 shows a traditional enumeration tree of a graph with vertex set $\{v_1, v_2, v_3, v_4\}$. Each node in the enumeration tree denotes a subgraph induced by a subset of vertices. The root node is an empty graph and every child node has one more vertex than its parent node. Hence, the ith level of an enumeration tree has $\binom{4}{i}$ nodes, $i = 0, 1, \ldots, 4$. The total number of nodes in the enumeration tree is $\sum_{i=0}^{4} \binom{4}{i} = 2^4 = 16$.

Assume that there is an input graph of n vertices with labels from 1 to n. To avoid duplicates in the enumeration tree, vertices in each node of the enumeration tree are

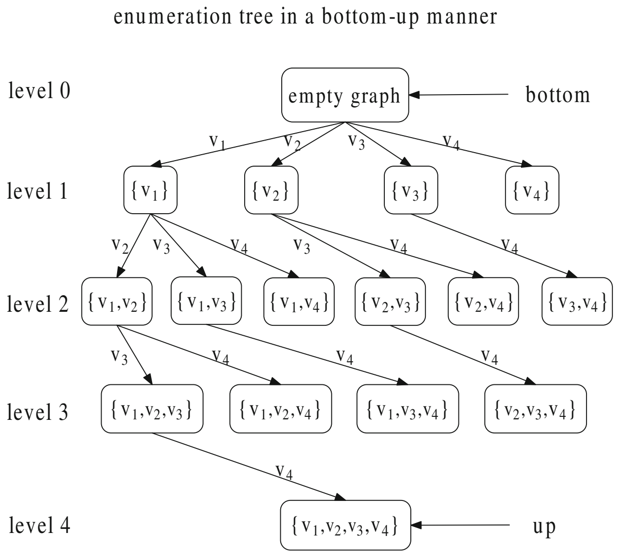
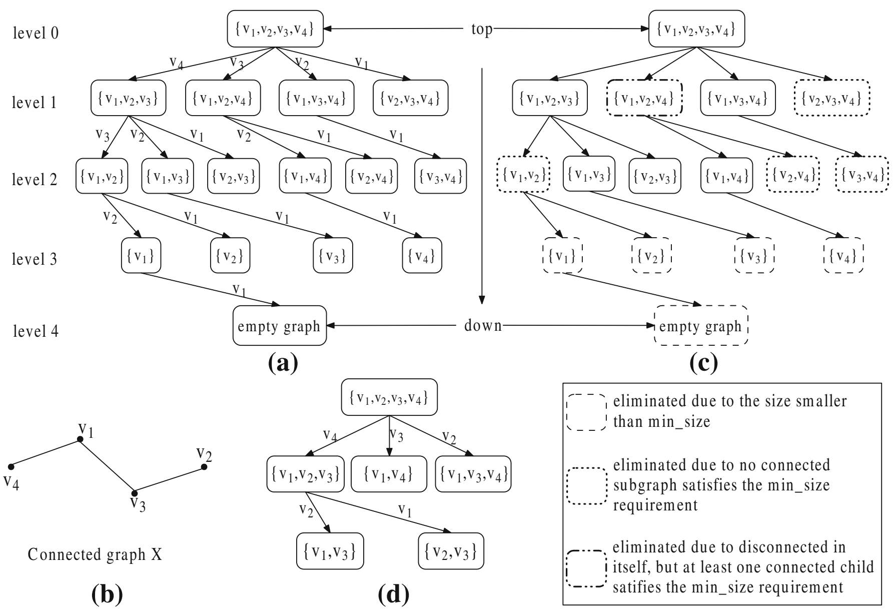


Fig. 10 Enumeration tree in bottom–up manner



level 0
level 1
level 2
level 3
level 4
top
down
(a)
(b)
(c)
(d)
Connected graph X
eliminated due to the size smaller than min_size
eliminated due to no connected subgraph satisfies the min_size requirement
eliminated due to disconnected in itself, but at least one connected child satifies the min_size requirement
empty graph

Fig. 11 Enumeration tree in top–down manner

sorted by their labels. Each a node w_i in the tree represents a subgraph with i_m vertices, denoted by $\{v_{i_1}, v_{i_2}, \ldots, v_{i_m}\}$. Each child of node w_i extends the set of vertices of w_i with one additional vertex from $i_m + 1, \ldots, n$. For example, the subgraph of the child node corresponding to $\{v_2, v_3\}$ in the second level has the vertex set $\{v_2, v_3, v_4\}$. A node has no child if it contains vertex v_4. Building an enumeration tree is time- and space-consuming since both the time and space complexities of constructing an enumeration tree for a graph of n vertices are $O(2^n)$.

Enumeration tree can also be used to enumerate a signed graph. The naive approach for finding all *MQAC*s is to enumerate all subgraphs of the signed graph. This is however not a scalable approach because the number of possible subgraphs grows exponentially even when no *MQAC* exists with an input graph.

In our proposed approach, we build two enumeration trees, one for each vertex set of a candidate graph. For example, consider the candidate graph in Fig. 11b. We enumerate all subsets of vertex set $\{v_1, v_2, v_3, v_4\}$ in a bottom–up manner as shown in Fig. 10 assuming that $min\_size = 2$ and $\epsilon = 0$. Such a bottom up approach has several drawbacks, namely: (1) there are nodes (e.g., at levels 0 and 1) that do not meet min_size requirement; and (2) some subgraphs of the vertex subsets are not *SCS*s, eg. $\{v_1, v_2\}$ and $\{v_2, v_4\}$. We unfortunately cannot remove them from the enumeration tree as their descendant nodes may be *SCS*s meeting the min_size requirement.

6.2 Enumerating vertex set

To overcome the above drawbacks, we adopt a *top–down* exploration of the enumeration tree to list all *SCS*s of a vertex set. For example, consider the same candidate graph

$\{v_1, v_2, v_3, v_4\}$ in Fig. 11b, a top–down enumeration tree is shown in Fig. 11a. The root node is the whole vertex set and every child node is a subgraph of its parent node, which has one more vertex than its child. For example, the subgraph with $\{v_1, v_3, v_4\}$ has a child node with vertices $\{v_3, v_4\}$ after removing the vertex v_1.

We employ the *depth-first search (DFS)* to traverse the top–down enumeration tree. The *DFS* starts from the root and explores as far as possible along each branch before backtracking. Formally, *DFS* explores by extending the first child node of the tree and further going deeper and deeper until it arrives a node that has no child. Note that the first child node always removes the last vertex from its parent node. Then the exploration backtracks, and returns to the most recent node which has not finished exploring. During the exploration, we do not explore child nodes which are children of an already explored node. For example, node $\{v_1, v_2, v_4\}$ may have three children: $\{v_1, v_2\}$, $\{v_1, v_4\}$ and $\{v_2, v_4\}$. However, $\{v_1, v_2\}$ is not child of node $\{v_1, v_2, v_4\}$ in the top–down enumeration tree since $\{v_1, v_2\}$ has already been child of $\{v_1, v_2, v_3\}$.

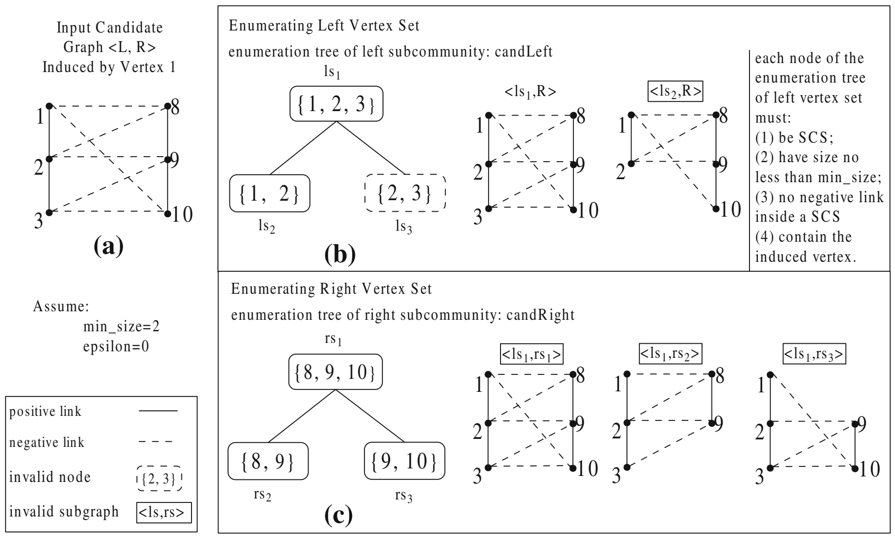
In the depth first exploration strategy, that a vertex set is a leaf-node if it does not contain vertex v_1. We can construct a counter example to prove this. Suppose a node i has m vertices, denoted as $\{v_{i_1}, v_{i_2}, \ldots, v_{i_m}\}$, and it does not contain vertex v_1. Consider a child node is obtained by dropping v_{i_j}, and the child node has vertices $\{v_{i_1}, \ldots, v_{i_{j-1}}, v_{i_{j+1}}, \ldots, v_{i_m}\}$. The child node is clearly also a child of $\{v_1, v_{i_1}, \ldots, v_{i_{j-1}}, v_{i_{j+1}}, \ldots, v_{i_m}\}$. In this case, we should not explore the same child node again.

Let min_size be 2. Our first observation from Fig. 11a is that all nodes below level 2 have sizes less than min_size. Thus, we can eliminate these nodes as shown in Fig. 11c. This is a significant advantage over bottom–up enumeration tree.

Moreover, in the top–down enumeration tree, we avoid generating non-connected subgraphs. For example, consider the vertex set in Fig. 11b, nodes $\{v_1, v_2\}$, $\{v_2, v_4\}$, $\{v_3, v_4\}$, $\{v_1, v_2, v_4\}$ and $\{v_2, v_3, v_4\}$ in Fig. 11a are non-connected graphs. Except for node $\{v_1, v_2, v_4\}$, they and their children in the tree can be eliminated directly since they do not have a child, which is a *SCS* meeting the min_size requirement.

However, node $\{v_1, v_2, v_4\}$ has a *SCS* child $\{v_1, v_4\}$ that satisfies the min_size requirement. To minimize the size of the enumeration tree, the current non-connected node is replaced by its child, which is a connected subgraph that satisfies the min_size requirement. Hence, we replace the node $\{v_1, v_2, v_4\}$ by $\{v_1, v_4\}$. Finally, we can obtain the enumeration tree shown in Fig. 11d.

We now illustrate the process of creating the enumeration tree as shown in Fig. 11d using $min\_size = 2$. In the beginning, we check the first candidate child of the root node: $\{v_1, v_2, v_3\}$ by removing the last vertex v_4 from the root node. Node $\{v_1, v_2, v_3\}$ is a child of root node since: (1) the number of its vertices satisfies the min_size requirement; and (2) it corresponds to subgraphs that is connected. We next explore the node $\{v_1, v_2, v_3\}$. The first candidate child of $\{v_1, v_2, v_3\}$ is $\{v_1, v_2\}$ corresponding to a non-connected subgraph with size 2. Therefore, $\{v_1, v_2\}$ and its children are dropped away from the tree. We backtrack to node $\{v_1, v_2, v_3\}$. The second candidate child of node $\{v_1, v_2, v_3\}$ is $\{v_1, v_3\}$ corresponding to a connected subgraph with size 2. Then $\{v_1, v_3\}$ becomes a child of $\{v_1, v_2, v_3\}$. However, all subgraphs of $\{v_1, v_3\}$ do not satisfy the min_size requirement. We therefore backtrack to node $\{v_1, v_2, v_3\}$



Input Candidate Graph <L, R> Induced by Vertex 1
(a)
Assume: min_size=2 epsilon=0
positive link
negative link
invalid node
invalid subgraph
Enumerating Left Vertex Set
enumeration tree of left subcommunity: candLeft
ls1 {1, 2, 3}
{1, 2} ls2
{2, 3} ls3
<ls1,R>
<ls2,R>
(b)
each node of the enumeration tree of left vertex set must: (1) be SCS; (2) have size no less than min_size; (3) no negative link inside a SCS (4) contain the induced vertex.
Enumerating Right Vertex Set
enumeration tree of right subcommunity: candRight
rs1 {8, 9, 10}
{8, 9} rs2
{9, 10} rs3
<ls1,rs1>
<ls1,rs2>
<ls1,rs3>
(c)

Fig. 12 Example of enumerating a candidate graph

again. In a similar way, the exploration of the enumeration tree continue until all nodes are explored or removed.

In the enumeration process, the left vertex set is treated differently from the right one since we expect the induced vertex to be part of a candidate graph. Hence, subgraphs not containing the induced vertex are eliminated when enumerating the left vertex set. In the end, we only obtain six nodes in the enumeration tree in Fig. 11d, rather than sixteen nodes shown in Fig. 11a.

6.3 Enumerating all *QAC*s of a candidate graph

Based on the enumeration tree, we presents an efficient way to find all *QAC*s of a candidate graph. The main idea is to enumerate the left and right vertex sets separately.

Algorithm 3 consists of three key steps, where enumerating two vertex sets are separated by repeated pruning. At first, the algorithm employs $enumerateSet(L, min\_size, \text{“}left''$) to construct the enumeration tree, $candLeft$, for the left vertex set at Line 1 such that: (i) each node of $candLeft$ is corresponding to a *SCS* w.r.t. positive links; (ii) there is not negative link for any pair of vertices in the *SCS*; (iii) the size of *SCS* is no less than min_size; (iv) each *SCS* must contain the induced vertex v. For example, consider the candidate graph $\langle L, R\rangle$ induced by vertex 1 in Fig. 12a, and suppose $min\_size = 2$ and $\epsilon = 0$. The vertex set L has three *SCS*s ls_1, ls_2 and ls_3. The algorithm generates an enumeration tree $candLeft$ containing two *SCS*s ls_1 and ls_2 as shown in Fig. 12b, where ls_3 is not a valid node since it does not contain induced vertex 1.

The second step is to reduce the size of subgraph by using repeated pruning. For each connected subgraph $ls \in candLeft$, we can get a signed graph $\langle ls, R\rangle$. Then we obtain a set of signed graphs by *degree*, *SCC* and *distance pruning* rules at Line 3 of Algorithm 3 and store them in $cand$. Any signed graph $\langle l, r\rangle$ in $cand$ can be discarded if $|l| < |ls|$ since l being a connected subgraph exists in the enumeration tree $candLeft$ and $\langle l, r\rangle$ can be obtained after pruning $\langle l, R\rangle$. In Fig. 12b, the algorithm constructs two graphs $\langle ls_1, R\rangle$ and $\langle ls_2, R\rangle$. The graph $\langle ls_2, R\rangle$ is discarded at repeated pruning. Only the graph $\langle ls_1, R\rangle$ remains after repeated pruning.

The final step is to enumerate the right vertex set of a subgraph, which remains after the second step. The algorithm calls the function $enumerateSet(r, min\_size, \text{“}right\text{”})$ to construct the enumeration tree $candRight$ at Line 6 such that: (i) each node rs is corresponding to a *SCS* w.r.t. positive links; (ii) there is not negative link between any pair vertices of rs; (iii) the size of rs is no less than min_size; (iv) there is not common vertex between ls and rs. Following previous example, we enumerate R in $\langle ls_1, R\rangle$, and obtain three *SCS*s for R as shown in Fig. 12c. Finally, we construct three signed graphs $\langle ls_1, rs_1\rangle$, $\langle ls_1, rs_2\rangle$ and $\langle ls_1, rs_3\rangle$.

```
Algorithm 3: Function $findQAC(v, \langle L, R\rangle, min\_size, \epsilon)$
   Input: candidate graph $\langle L, R\rangle$ induced by vertex $v$ with parameters $min\_size$ and $\epsilon$;
   Output: $Q$: all maximal $(\epsilon, minsize)$ $aQACs$ within $\langle L, R\rangle$;
1  $candLeft \leftarrow enumerateSet(L, min\_size, \text{"}left\text{"})$//enumerating left vertex set;
2  for each node $ls \in candLeft$ do
3      $cand \leftarrow repeatedPruning(v, \langle ls, R\rangle, min\_size, \epsilon)$;
4      for each $\langle l, r\rangle \in cand$ do
5          if $|ls| = |l|$ then
6              $candRight \leftarrow enumerateSet(r, min\_size, \text{"}right\text{"})$//enumerating right vertex set;
7              for each node $rs \in candRight$ do
8                  if $checkQAC(\langle ls, rs\rangle)$ then
9                      $Q \leftarrow Q \cup \{\langle ls, rs\rangle\}$;
10                 end
11             end
12         end
13     end
14 end
15 return $Q$
```

6.4 Verifying QAC

In the verification stage, for each signed graph $\langle ls, rs\rangle$, the function $checkQAC(\langle ls, rs\rangle)$ in Algorithm 3 verifies whether the signed graph $\langle ls, rs\rangle$ is a *QAC* or not at Line 8. The function $checkQAC(\langle ls, rs\rangle)$ returns true if vertices from both sides satisfy the criteria of *QAC*. For example in Fig. 12, as none of the three signed graphs $\langle ls_1, rs_1\rangle$, $\langle ls_1, rs_2\rangle$ and $\langle ls_1, rs_3\rangle$ is a *QAC* with $min\_size = 2$ and $\epsilon = 0$, the algorithm returns an empty *QAC* set.

6.5 Complexity analysis

Enumerating a sub-community involves generating all *SCS*s in the sub-community w.r.t. positive links. Let V be the vertex set of the sub-community. The number of *SCS*s involving the clique will be of the $O(2^{|V|})$. In the worst case, the complexity of enumerating a sub-community is $O(2^{|V|})$. Fortunately, sub-communities are usually small (see distribution of graph size after Pruning Rule 4 in Fig. 17). In addition, the probability of forming a clique of large size is very small. Let $\langle L, R\rangle$ be a candidate *QAC*. The complexity of verifying $\langle L, R\rangle$ is a *QAC* is $O(|L| \times |R|)$.

Based on the complexity analysis of the enumeration stage, we can better understand the importance and effectiveness of Pruning Rule 4. Note that enumerating a sub-community involves only the positive links of a candidate graph and Pruning Rule 4 involves negative links. Pruning Rule 4 is essential because: (1) the proportion of negative links is small in a signed graph; and (2) the large proportion of positive links in a signed graph affects the number of *SCS*s generated. Further empirical results showing the effectiveness of Pruning Rule 4 will be shown in Figs. 16, 17 and 21d.

7 Mascot algorithm

In this section, we pull all steps of the Mascot framework together in Fig. 5, and propose the Mascot algorithm in Algorithm 4. We also derive other variants of Mascot algorithm which are later used in our experiments.

7.1 Outline of algorithm

The Mascot algorithm requires parameters min_size and ϵ. Note that, unless otherwise specified, we just find the all the maximal *aQACs* as the algorithm maximal *rQACs* can be obtained by a minor change to degree pruning rule in the pruning stage and the criteria to verify *rQACs* in the enumeration stage.

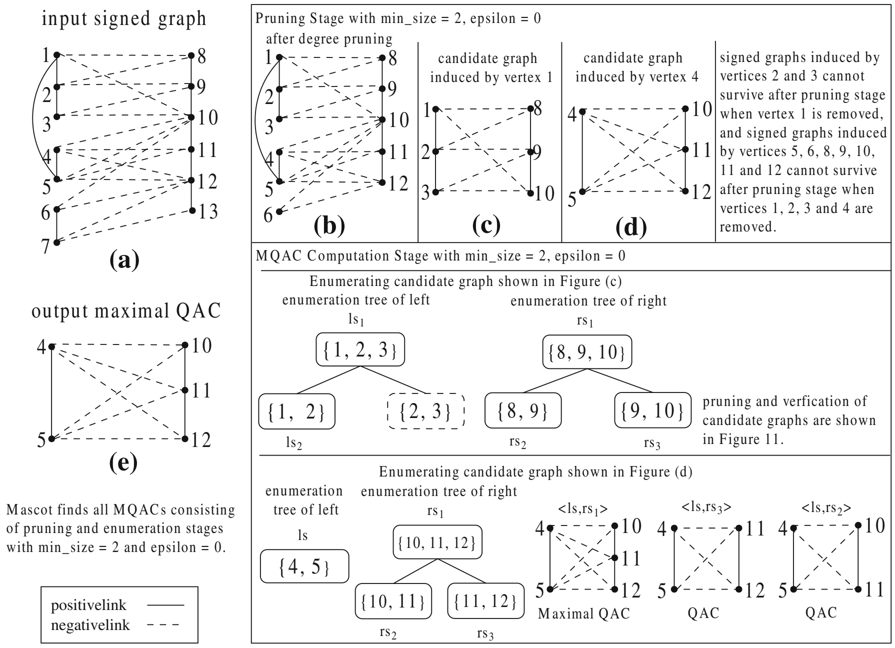
Algorithm 4 consists of two stages: pruning stage (from Lines 2 to 9, where Line 2 is degree pruning for input graph and Line 4 is the *combined pruning* for each induced graph) and enumeration stage (from Lines 10 to 18, where Line 11 enumerates all *QAC*s involving the vertex v and Lines 12 to 17 verify if a *QAC* forms a *MQAC*).

In the pruning stage, we get the induced vertex set contained in graph $G^{'}$ after applying *degree pruning* rule at Line 2. For each candidate vertex $v \in G^{'}$, we apply the four pruning rules on the induced graph of v at Line 4. To eliminate duplicates of candidate subgraph, we remove vertex v from $G^{'}$ after pruning operation on all candidate graphs which contain the vertex v, and update a graph $G^{'}$ in the Line 8.

In the enumeration stage, for each candidate graph *v_cand_graph* that is induced by vertex v, the algorithm finds all *QAC*s involving the induced vertex v, denoted as v_QAC_set, at Line 11.

For each *QAC* $q \in v\_QAC\_set$, we add it into Q at Line 14 if its supergraph does not exist in both the result set Q and v_QAC_set. Furthermore, each existing *QAC* in Q, q_1, is removed from Q at Line 15 if q_1 is also induced by vertex v and is a subgraph of q. Note that q_1 cannot become a subgraph of q if q_1 and q have different induced vertices because the induced vertex of q_1 would have been removed before we generate the candidate graph v_cand_graph at Line 8.

For example, suppose we have the input signed graph in Fig. 13a. Let $min\_size = 2$ and $\epsilon = 0$. After iterative *degree pruning* at Line 2 of Algorithm 4, Mascot computes the induced vertex set contained in graph $G^{'}$ with 11 vertices shown as in



input signed graph
(a)
output maximal QAC
(e)
Mascot finds all MQACs consisting of pruning and enumeration stages with min_size = 2 and epsilon = 0.
positivelink
negativelink
Pruning Stage with min_size = 2, epsilon = 0
after degree pruning
(b)
candidate graph induced by vertex 1
(c)
candidate graph induced by vertex 4
(d)
signed graphs induced by vertices 2 and 3 cannot survive after pruning stage when vertex 1 is removed, and signed graphs induced by vertices 5, 6, 8, 9, 10, 11 and 12 cannot survive after pruning stage when vertices 1, 2, 3 and 4 are removed.
MQAC Computation Stage with min_size = 2, epsilon = 0
Enumerating candidate graph shown in Figure (c)
enumeration tree of left
enumeration tree of right
{1, 2, 3}
{1, 2}
{2, 3}
{8, 9, 10}
{8, 9}
{9, 10}
pruning and verfication of candidate graphs are shown in Figure 11.
Enumerating candidate graph shown in Figure (d)
enumeration tree of left
enumeration tree of right
{4, 5}
{10, 11, 12}
{10, 11}
{11, 12}
Maximal QAC
QAC
QAC

Fig. 13 Example of MASCOT

Algorithm 4: Mascot Algorithm

```
   Input: A signed graph $G$, parameters $min\_size$ and $\epsilon$;
   Output: $Q$: the MQACs set (initialized to be $\emptyset$);
1  $cand\_graph\_queue \leftarrow \emptyset; Q \leftarrow \emptyset$;
2  $G' \leftarrow degreePruning(G, min\_size, \epsilon)$// $G = (V, E^+, E^-)$;
3  for each node $v \in V$ and $|V| \geq 2 * min\_size$ do
4      $v\_cand\_graph\_set \leftarrow combinedPruning(v, G', min\_size, \epsilon)$;
5      if $v\_cand\_graph\_set \neq \emptyset$ then
6          add elements of $v\_cand\_graph\_set$ into $cand\_graph\_queue$;
7      end
8      $V \leftarrow V \setminus \{v\}$ and remove edges involving $v$ from $E^+$ and $E^-$;
9  end
10 for each element $v\_cand\_graph \in cand\_graph\_queue$ do
11     $v\_QAC\_set \leftarrow findQAC(v\_cand\_graph, min\_size, \epsilon)$;
12     for each $q \in v\_QAC\_set$ do
13         if $q$ does not have supergraph in both $v\_QAC\_set$ and $Q$ then
14             $Q \leftarrow Q \cup \{q\}$;
15             for each $q_1 \in Q$ is induced by vertex $v$ and is a subgraph of $q$ then remove $q_1$ from $Q$;
16         end
17     end
18 end
19 return $Q$
```

Fig. 13b. For the first induced vertex 1, integrating four pruning rules to pruning $Ind(1, G')$ at Line 4, MASCOT gets one candidate graph shown in Fig. 13c. After getting candidate graph induced by vertex 1, MASCOT removes vertex 1 from G' and updates the signed graph G' at Line 8. It is easy to determine that $Ind(2, G')$ and $Ind(3, G')$ can be pruned by combining pruning at Line 4. After removing vertices

1, 2 and 3 at Line 8, we get $Ind(4, G^{'}) = \langle\{4, 5, 6\}, \{10, 11, 12\}\rangle$. After pruning vertices from $Ind(4, G^{'})$, we get the second candidate graph shown in Fig. 13d. There is not any induced graph that can survive after removing vertices 1, 2, 3 and 4 at Line 8.

Mascot performs enumeration stage from Lines 10 to 18. It enumerates two candidate graphs in Fig. 13c and d. For the first candidate graph, the enumeration step at Line 11 had already been illustrated in Fig. 12. There is no candidate graph that can form a *QAC* with $min\_size = 2$ and $\epsilon = 0$. For the second candidate graph shown in Fig. 13d, Mascot enumerates the left and right vertex sets by enumeration trees at Line 11, and finds three *QAC*s $\langle ls, rs_1\rangle$, $\langle ls, rs_2\rangle$ and $\langle ls, rs_3\rangle$ with $min\_size = 2$ and $\epsilon = 0$. *QAC*s $\langle ls, rs_2\rangle$ and $\langle ls, rs_3\rangle$ cannot become *MQAC*s since they are subgraph of $\langle ls, rs_1\rangle$ at Line 13. Hence, Mascot outputs $\langle ls, rs_1\rangle$ as a *MQAC* at Line 19 as shown in Fig. 13e.

The complexities of the *degree pruning rule*, the *combined pruning* and finding *QAC*s have been analyzed in previous two sections. Assume that the number of *QAC*s of an original signed graph is N and the maximum size of *QAC*s is M. Then the complexity of verifying *MQAC*s from Lines 12 to 17 in Algorithm 4 is $O(MN^2)$ in the worst case when all *QAC*s share the same induced vertex.

7.2 Variants of Mascot

To the best of our knowledge, there is not the other algorithm which finds all *MQAC*s from signed networks. To illustrate the performance of Mascot and our proposed pruning rules, we therefore propose four variants of Mascot as baselines.

From Mascot, we create four proposed variant algorithms as baselines. Each variant disables one particular pruning rule PR in Algorithm 4, and is denoted by Mascot$_{\overline{PR}}$.

In particular, we have Mascot$_{\overline{deg}}$, Mascot$_{\overline{SCC}}$, Mascot$_{\overline{dis}}$ and Mascot$_{\overline{int}}$ disabling the *degree pruning*, *SCC pruning*, *distance pruning* and *interaction graph pruning* rules, respectively. Note that algorithm Mascot$_{\overline{deg}}$ disables the *degree pruning* rule in both steps in Lines 2 and 4 of Algorithm 4.

8 Experiments on synthetic graph

In this section, we evaluate both the efficiency and accuracy of our proposed Mascot algorithm on synthetic signed graphs. Given that there are no existed algorithms detecting *MQAC*s, we compare the elapsed time of Mascot against its variants. All programs were implemented in Java, and were conducted on a dual core 64-Bit processor with 3.06 and 3.06 GHz CPUs, respectively, and 128 GB of RAM.

8.1 Graph generation

We first describe our synthetic signed graph generation algorithm as shown in Fig. 14. Based on a graph generation algorithm proposed by Palmer and Steffan (2000), our algorithm starts with vertex generation, and is followed by edge generation. From

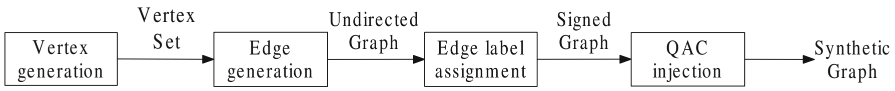


Fig. 14 Synthetic graph generation

the generated undirected graph, we derive a signed graph by assigning sign labels to edges randomly. To evaluate accuracy, we inject *QAC*s into the signed graph in the last step of the generation.

We use Syn_N to denote a synthetic signed graph with N vertices. The generator outputs a synthetic graph as shown in Algorithm 5, which consists of two parts with eight input parameters as shown in Table 2. Note that $minDegree$ is degree threshold for selecting vertices to inject *QAC*s. The first part is to generate a signed graph (at Lines 1–26). In the part, the algorithm initializes a graph with N vertices and empty adjacency list. To obtain a scale-free network, it assigns a degree k to each vertex v with probability $Pr[deg(v) = k] \approx k^{-\alpha}$. Note that the sum of vertex degree $totalDeg$ should be even. The steps from Lines 7 to 9 guarantee this. Next, we assign the neighbors of each vertex after sorting vertices by degree in decreasing order. The final step of this part is to assign positive and negative labels to edges from Lines 22 to 26. Parameter θ controls the required proportion of negative links.

The second part of graph generation is to inject $\beta \times N$ number of *QAC*s into the signed graph (at Lines 27–47). Before we inject *QAC*s into the synthetic graph, a set of vertices $topVertexSet$ with both positive and negative degrees $\geq minDegree$ (default value is 15) is selected at Lines 28–30. These vertices are selected to construct *QAC*s since vertices in $topVertexSet$ have many positive and negative neighbors. Lines 31–46 of Algorithm 5 inject *QAC*s into the synthetic graph. The injection controls: (1) the number of vertices of each sub-community; (2) the edges within and across sub-communities.

Assume that $Q_{ac} = \langle L, R\rangle$ be an injected *QAC*. $|L|$ and $|R|$ are two numbers sampled from a uniform distribution $U[min, max]$ at Line 34. L and R are selected from vertex set $\Gamma^{+}(cand)$ or $\Gamma^{-}(cand)$ randomly at Line 35, i.e., $L \subset \Gamma^{+}(cand)$ and $R \subset \Gamma^{-}(cand)$ (Note that $max < minDegree$). To ensure that L and R form a *QAC*, the algorithm removes all edges between any pair of vertices in $L \cup R$ at Line 36. A chain of positive links is formed in L and R separately at Line 37 so that each of them is now a *SCS*. Finally, $\langle L, R\rangle$ forms a (ϵ, min) absolute *QB* from Lines 39 to 46. According to the definition of (ϵ, min) absolute *QB*, $epsilonL(v)$, $epsilonR(v) \leq \epsilon$, where $epsilonL(v)$ ($epsilonR(v)$) denotes the number of disconnected vertices of v. In the algorithm, Line 40 guarantees the negative degree requirement of vertices from L, and Lines 42 to 44 guarantee the same requirement for R vertices. All negative neighbors of a vertex are assigned at Line 45.

Figure 15 shows the distributions of positive, negative and total degrees of a synthetic signed graph generated with $N = 500K$, $\beta = 0.001$, $\theta = 0.1$, $\alpha = 2.5$, $min = 3, \epsilon = 1, max = 8$ and $minDegree = 15$, denoted as Syn_500K. We observe that degree distributions follow power law as expected. Unless otherwise specified, the default parameter settings for the generator are listed highlighted as bold in Table 2.

Algorithm 5: Synthetic Undirected Signed Graph Generation

```
   Input: $N, \theta, \alpha, \beta, [min, max], \epsilon, minDegree(>= max)$;
   Output: $G$: a signed graph injected with some $QAC$s;
1  $vertexSet \leftarrow$ a vertex set with $N$ vertices;
2  $edgeSet \leftarrow \emptyset$; $totalDeg \leftarrow 0$; // initialized sum of degrees;
3  for each vertex $x \in vertexSet$ //Step 1: Vertex generation do
4      $deg(x) \leftarrow$ sample an integer value from the power law distribution with parameter $\alpha$;
5      $totalDeg \leftarrow totalDeg + deg(x)$;
6  end
7  if $numEdge$ is odd then
8      $deg(x_1) \leftarrow deg(x_1) + 1$; $totalDeg \leftarrow totalDeg + 1$;
9  end
10 sort vertices in $vertexSet$ by descending order of $deg(\cdot)$;
11 $V \leftarrow vertexSet$;
12 for each vertex $x \in vertexSet$ //Step 2: Edge generation do
13     randomly select a vertex set $Y$ from $V$ s.t. $|Y| = deg(x)$ and $x \notin Y$;
14     for each vertex $y \in Y$ do
15         $edgeSet \leftarrow edgeSet \cup \{(x, y)\}$; $deg(y) \leftarrow deg(y) - 1$;
16         if $deg(y) == 0$ then $V \leftarrow V - \{y\}$;
17         ;
18     end
19     $V \leftarrow V - \{x\}$;
20 end
21 for each edge $e \in edgeSet$ //Step 3: Label assignment do
22     $random \sim U[0, 1]$;
23     if $random \leq \theta$ then $sign(e) \leftarrow' -'$;
24     ;
25     else $sign(e) \leftarrow' +'$;
26     ;
27 end
28 $topVertexSet \leftarrow \emptyset$;
29 for each vertex $x \in vertexSet$ do
30     if $\min\{deg^{+}(x), deg^{-}(x)\} \geq minDegree$ then $topVertexSet \leftarrow topVertexSet \cup \{x\}$;
31     ;
32 end
33 $numQAC \leftarrow \beta \times N$;
34 for $i$ = 1 to $numQAC$ //Step 4: QACs injection do
35     select a vertex $cand$ from $topVertexSet$ at random;
36     construct vertex sets $L$ and $R$ s.t. $cand \in L$, $(L\backslash\{cand\}) \subset \Gamma^{+}(cand)$, $R \subset \Gamma^{-}(cand)$ and $|L|$,
       $|R|$ follow uniform distribution $U[min, max]$;
37     remove all links between any pair of vertices of $L \cup R$;
38     for each vertex set, construct a chain connected by '+' edges s.t. each vertex is randomly
       assigned its successor;
39     $R^{'} \leftarrow R$; for each vertex $r \in R$, $epsilonR(r) \leftarrow 0$;
40     for each vertex $l \in L$ do
41         $epsilonL(l) \sim U[0, \min\{\epsilon, |R^{'}|\}]$;
42         $missVertex(l) \leftarrow$ randomly select $epsilonL(l)$ vertices from $R^{'}$;
43         for each $r \in missVertex(l)$ do
44             $epsilonR(r) \leftarrow epsilonR(r) + 1$; remove $r$ from $R^{'}$ if $epsilonR(r) = \epsilon$;
45         end
46         $l$ links all vertices in $R - missVertex(l)$ by using undirected '-' edges;
47     end
48 end
49 return $G$
```

Table 2 Synthetic graph generation parameter settings

Parameter	Meaning	Value
ϵ	The absolute parameter for *aQAC*	**1**
N	# Vertices in a synthetic graph	$[100K, 200K, \ldots, \mathbf{500K}]$
β	# *QAC*s per vertex	$[\mathbf{1.0 \times 10^{-3}}, \ldots 1.5 \times 10^{-3}]$
θ	Proportion of negative links	$[\mathbf{0.1}, 0.12, \ldots, 0.2]$
α	The power law distribution parameter	[2.1,2.3,**2.5**,2.7,2.9]
$[min, max]$	Minimum and maximum vertex set size constraint	[3,8]
$minDegree$	Degree threshold for selecting vertices to inject *QAC*	**15**

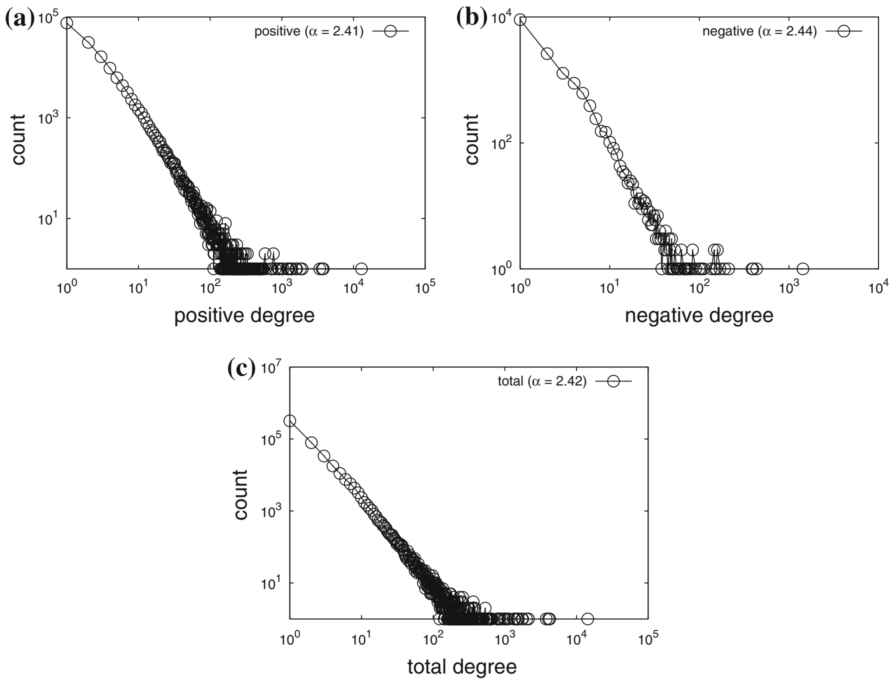


Fig. 15 Degree distributions of Syn_500K. **a** Positive degree distribution. **b** Negative degree distribution. **c** Total degree distribution

8.2 Performance results

We create two sets of synthetic graphs, one for evaluating the absolute version of Mascot finding *aQAC*s, and the other for *rQAC*s. We denote the absolute and relative versions as *aMascot* and *rMascot* respectively.

8.2.1 *Measures of performance*

We adopt two performance measures: (i) the elapsed time and; (ii) recall, that measure the efficiency and accuracy of the algorithms respectively.

We measure the elapsed time in seconds, and it consists of the elapsed time of pruning rules and the elapsed time of enumerating candidate graphs. Recall is derived from the number of *MQAC*s found by MASCOT. Every *MQAC* found by an algorithm may belong to one of three categories below.

- Type I: It is one of the injected *QAC*s.
- Type II: It is a supergraph of any injected *QAC*.
- Type III: It is not a supergraph of any injected *QAC*.

An injected *QAC* is considered found if it is among the found *QAC*s (i.e., Type I), or is a subgraph of some found *QAC*s (i.e., Type II). Therefore, we can define two recalls for Types I and II as

$$recall_I = \frac{\#\text{ injected} QAC\text{s that are found based on Type I criteria}}{\#\text{ injected} QAC\text{s}};$$
$$recall_{II} = \frac{\#\text{ injected} QAC\text{s that are found based on Type II criteria}}{\#\text{ injected} QAC\text{s}}.$$

We then define total recall to be $recall = recall_I + recall_{II}$.

8.2.2 *Performance by varying graph size N*

We evaluate MASCOT algorithm on synthetic graphs of size from 100 to 500 K. We record the elapsed time of *aMascot* on synthetic graphs injected with *aQACs*, that of *rMascot* injected with *rQACs*, and baselines. Note that we show the elapsed times of *Mascot* and baselines in Fig. 16a and b if the elapsed time is not longer than 8 h, i.e., $\text{MASCOT}_{\overline{deg}}$ takes more than 8 h to find all *MQAC*s when $N \geq 300K$. As shown in Fig. 16a and b, *aMascot* and *rMascot* outperform all baselines and requires less than 60 seconds when $N = 500K$.

We observe that without using specific pruning rules, the baselines may take much longer time. In terms of performance of baselines, interaction graph pruning also appears to play a significant part reducing the elapsed time.

From Fig. 16a and b, we also observe the elapsed time for both *aMascot* and *rMascot* increases as the input graph size increases.

To analyze the elapsed time furthermore, we show some descriptive statistics for input graphs with different graph sizes (N) in Table 3.

The statistic $QACperVertex$ is defined by

$$QACperVertex = \frac{\#\,\text{injected} QACs}{|topVertexSet|}. \tag{12}$$

A larger value for $QACperVertex$ implies higher chance for two injected *QAC*s to share the same vertex.

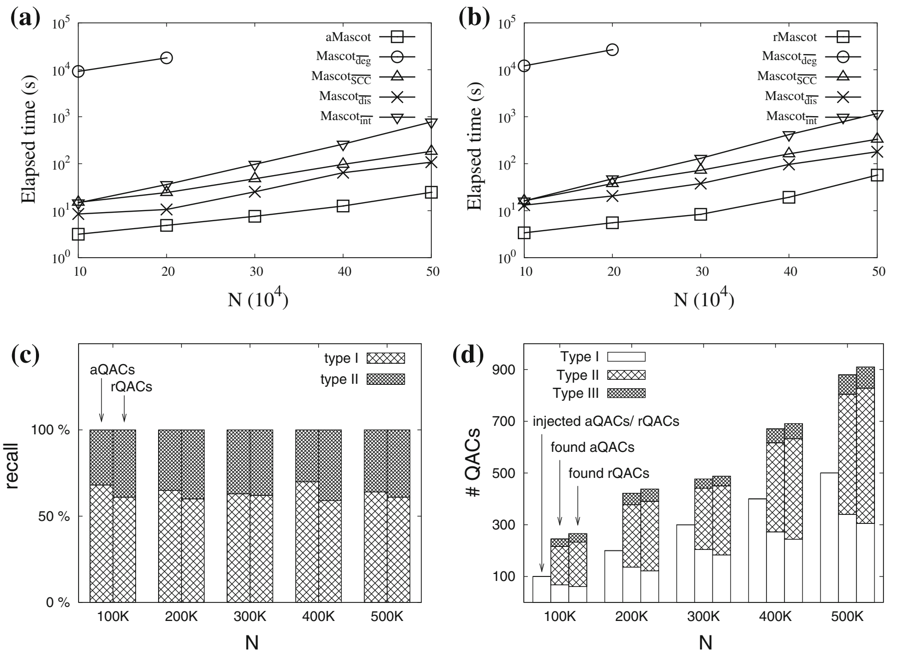


Fig. 16 Performance by varying graph size (N) ($min\_size = 3, \epsilon = 1, \delta = \frac{1}{3}, \beta = 0.001, \theta = 0.1, \alpha = 2.5$). **a** Elapsed time for absolute version, **b** elapsed time for relative version, **c** recall and **d** # QACs found

Table 3 Descriptive statistics over synthetic graphs

N	100 K	200 K	300 K	400 K	500 K
# Injected QACs	100	200	300	400	500
$\lvert topVertexSet \rvert$	161	326	495	645	823
QACperVertex	0.62	0.61	0.60	0.62	0.60
# Candidate graphs after pruning stage	274	813	1519	2644	4109

As shown in Table 3, $QACperVertex$ does not vary much with N. Table 3 also shows that MASCOT gets more candidate graphs after the pruning stage. This explains more elapsed time for larger N.

To better understand interaction graph pruning, Fig. 17 shows the distributions of graph size after repeated pruning and applying Pruning Rule 4 on Syn_500K. We observe that: (1) the graph sizes are quite small after repeated pruning; and (2) Pruning Rule 4 further reduces the large graphs to small ones. It indicates the effectiveness of Pruning Rules 1 to 4. To sum up these results, both *aMascot* and *rMascot* run fast on our synthetic graphs and their pruning rules are very effective in reducing the search space.

Figure 16c shows the recalls of *aMascot* and *rMascot*. We observe the recalls of *aMascot* and *rMascot* are 100 % for different N with more 50 % of injected QACs

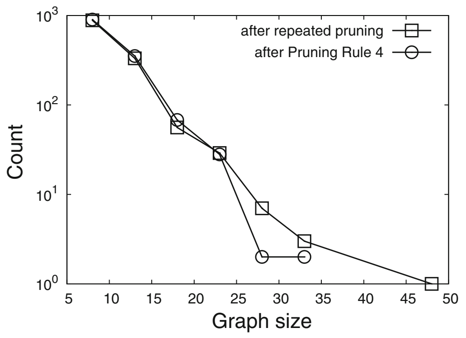


Fig. 17 Distributions of graph sizes after repeated pruning and Pruning Rule 4

recovered in their original forms. For the remaining injected *QAC*s, some of the found *MQAC*s are their supergraphs. We therefore conclude that MASCOT has high recall on the synthetic graphs.

Figure 16d shows the actual numbers of *MQACs* found by *aMascot* and *rMascot*. For each N, The left bar shows the number of injected *aQACs* or *rQACs*. The middle and right bars show the composition of *aQAC*s or *rQAC*s found matching against the injected *QAC*s on the synthetic graphs.

From Fig. 16d, we observe that only small proportion of *MQACs* found belong to Type III.

That is, most *MQAC*s are either the injected *QAC*s or their supergraphs.

8.2.3 Performance by varying the proportion of injected QACs β

Figure 18a and b show the elapsed times of MASCOT and baselines by varying β. Please note that MASCOT$_{\overline{deg}}$ takes more than 8 h to find all *MQAC*s.

It is easy to find that MASCOT outperforms all baselines. From Fig. 18a and b, we also observe that the elapsed time increases with β. This is due to more candidate graphs remain after pruning rules, and more injected *QAC*s share common vertices as β increases.

Figure 18c shows that the total recall is still 100 % suggesting that MASCOT returns all injected *QAC*s in original or supergraph form. As shown in Fig. 18d, smaller proportions of Type I *MQAC*s are found since the injected *QAC*s share more vertices.

8.2.4 Performance by varying proportion of negative links θ

Figure 19a and b show the elapsed times of MASCOT and baselines when increasing the proportion of negative links in a synthetic graph.

Note that MASCOT$_{\overline{deg}}$ takes more than 8 h to find all *MQAC*s with different values of θ.

MASCOT also outperforms all baselines. The figures show the elapsed time increases with θ.

As the proportion of negative links becomes larger, MASCOT has larger induced graphs, thus requiring more elapsed time.

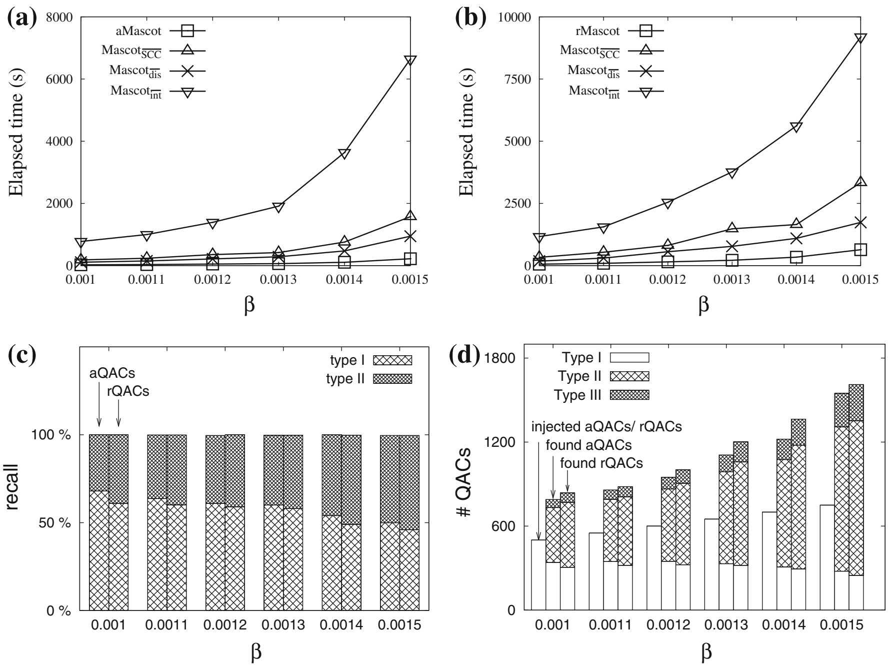


Fig. 18 Performance by varying proportion of injected *QAC*s $(\beta)(N = 500K, min\_size = 3, \epsilon = 1, \delta = \frac{1}{3}, \theta = 0.1, \alpha = 2.5)$. **a** Elapsed time for absolute version, **b** elapsed time for relative version, **c** recall and **d** # *QAC*s found

Despite that, Fig. 19c shows that Mascot still achieves very high recall. From Fig. 19d, we observe smaller proportions of *MQAC*s found belong to Type I since the injected *QAC*s are likely to larger when more negative links are in the signed graph.

8.2.5 *Performance by varying min_size*

Figure 20a and b illustrate the performance of *aMascot* on *Syn*_500*K* injected with *aQACs* by varying *min_size* from 3 to 8.

We can observe that the elapsed time for *aMascot* decreases as *min_size* increases, so is the number of *aMQACs*.

This shows that Mascot is able to reduce the elapsed time and search space using *min_size*.

As we use larger ϵ, the elapsed time increases. A larger ϵ increases the number of *QAC*s to be found, hence increasing the time required.

In Fig. 21c and d, we observe the similar performance trends for *rMascot*.

In addition, we also compare *Mascot* with the algorithm for detecting *DAC*s (Lo et al. 2013, 2011) when $\epsilon = 0$ or $\delta = 0$. Under these conditions, the inter-community negative links of a *QAC* form a biclique, which is fully connected by negative links; thus, the *QAC* will also be a *DAC*. We show the results of our experiments in Fig. 20a–d. From Fig. 20a and c, when *min_size* is large, both *aMascot* and *rMascot* are more

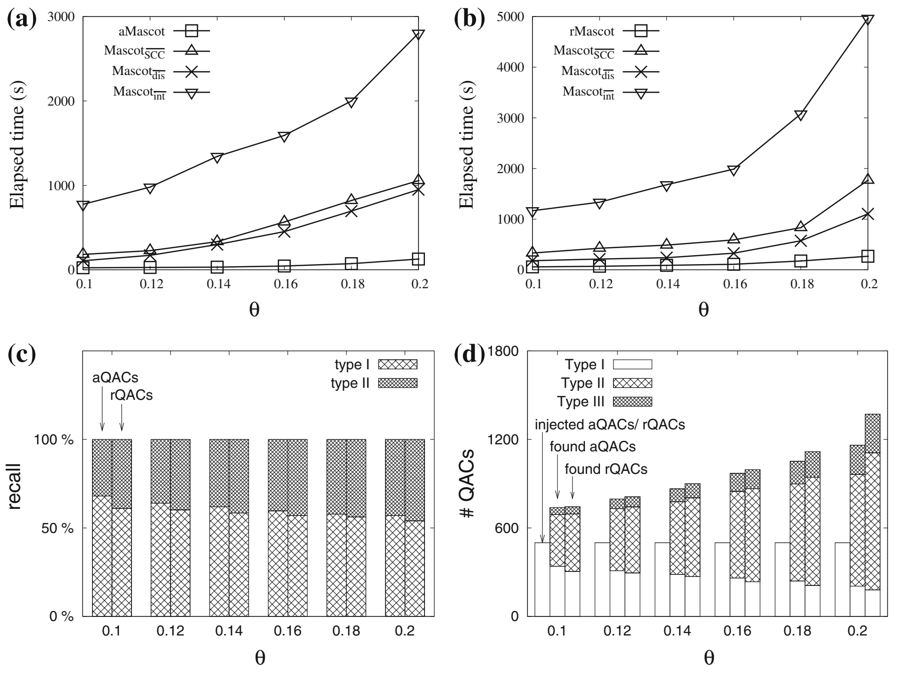


Fig. 19 Performance by varying proportion of negative links (θ) ($N = 500K, min\_size = 3, \epsilon = 1, \delta = \frac{1}{3}, \beta = 0.001, \alpha = 2.5$). **a** Elapsed time for absolute version, **b** elapsed time for relative version, **c** recall and **d** # *QAC*s found

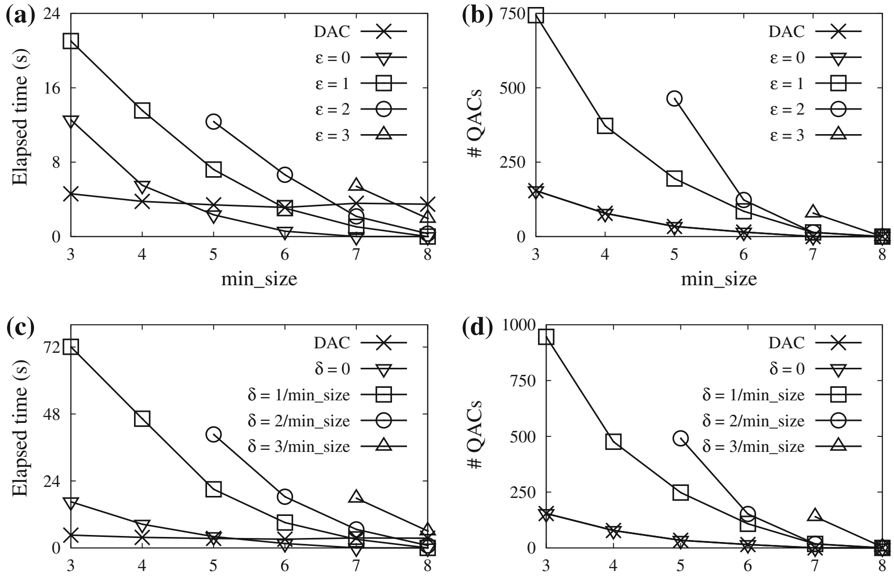


Fig. 20 Performance by varying min_size ($N = 500K, \beta = 0.001, \theta = 0.1, \alpha = 2.5$). **a** Elapsed time (*aMascot*), **b** # *QAC*s found (*aMascot*), **c** elapsed time (*rMascot*) and **d** # *QAC*s found (*rMascot*)

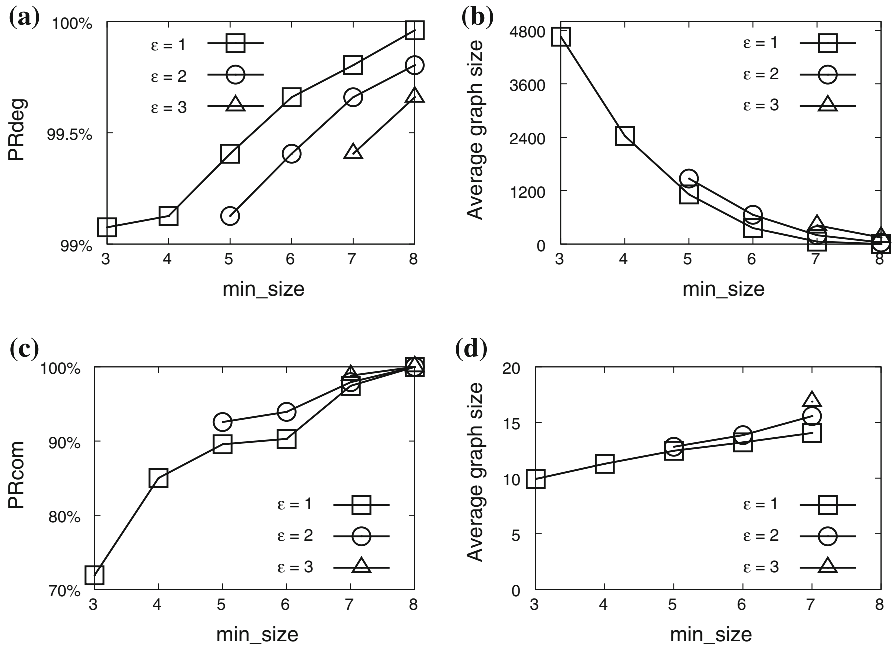


Fig. 21 Pruning efficiency by varying min_size ($N = 500K, \beta = 0.001, \theta = 0.1, \alpha = 2.5$). **a** Pruning ratio of *degree pruning* (*aMascot*). **b** Average size of induced graphs (*aMascot*). **c** Pruning ratio of *combined pruning* (*aMascot*). **d** Average size of candidate graphs (*aMascot*)

efficient than the algorithm that mines *DAC*s. From Fig. 20b and d, we observe that *aQAC* and *rQAC* help us to find more antagonistic communities due to their relaxed conditions. Note that the elapsed time for detecting *DAC*s is not affected by min_size since the algorithm for detecting *DAC*s generates all maximal bi-cliques from some subgraphs.

To understand the effectiveness of pruning, we measure the pruning efficiency of our pruning rules. Our pruning aims to reduce the number of candidate graphs and their sizes. We therefore introduce two pruning efficiency measures: (i) average graph size after pruning and; (ii) pruning ratio of each pruning rule. Average graph size is defined by summing the number of vertices of induced or candidate graphs divided by the number of such graphs after applying the pruning rule. The pruning ratio of *degree pruning* rule and *combined pruning* (i.e., degree, SCC and distance pruning) are defined as

$$PRdeg = 1 - \frac{\text{\# induced vertices}}{\text{\# vertices in an input graph}};$$
$$PRcom = 1 - \frac{\text{\# candidate graphs}}{\text{\# induced graphs}}.$$

$PRdeg$ and $PRcom$ range from 0 to 1. The larger the pruning ratio, the more effective is pruning.

Figure 21 shows the pruning efficiency of *aMascot* by varying min_size and ϵ on Syn_500K. The derived pruning ratio $PRdeg$ exceeds 99 % which suggests many vertices are pruned away. The average induced graph size after *degree pruning* rule has around 4K vertices when $min\_size = 3$ and $\epsilon = 1$. While this number is small compared with $500K$ vertices in Syn_500K, it is still computationally expensive to enumerate a subgraph with thousands of vertices.

Figure 21c shows that combined pruning further reduces the number of candidate graphs with pruning ratio $PRcom$ above 70 %. Figure 21d further shows that the average candidate graph size is smaller than 20. We therefore conclude that our pruning techniques achieve good pruning efficiency. The same conclusion can be made for *rMascot*.

9 Experiments on real networks

In this section, we report the results and analysis of Mascot for finding all *MQAC*s in two real social networks: myGamma and Epinions.

9.1 Description of datasets

myGamma: myGamma is an online social networking site, which can be accessed through mobile phone with an Internet connection. On that network, friendships are considered positive links, and a negative link is defined between two users when one of them blocks the others. In our experiment, we select myGamma users from eight countries in five continents. Some descriptive statistics of this network is given in Table 4. The statistic $ratio^{-}$ denotes the proportion of negative links. We remove both positive and negative self-loop edges and directed edges between two vertices that have conflicting polarities. We ignore directions of both positive and negative links. Finally, we combine users from these eight countries into an undirected signed graph, denoted as myGamma.

Epinions: Epinions is a product review web site. Users can write subjective reviews about many different types of items, such as software, movies and music videos, etc. A

Table 4 Descriptive statistics on myGamma networks grouped by country

CountryID	Country	# Users	# Edges	$ratio^{-}$	Density
au	Australia	2325	7895	0.0610	2.9e−3
cn	China	8285	10,184	0.0212	2.9e−4
fr	France	211	167	0.0538	7.5e−3
gh	Ghana	5648	26,697	0.0124	1.7e−3
ir	Iran	2100	6615	0.1339	3.0e−3
py	Paraguay	2091	8900	0.0368	4.1e−3
sg	Singapore	25,523	131,298	0.0474	4.0e−4
us	United States	22,392	75,794	0.0805	3.0e−4

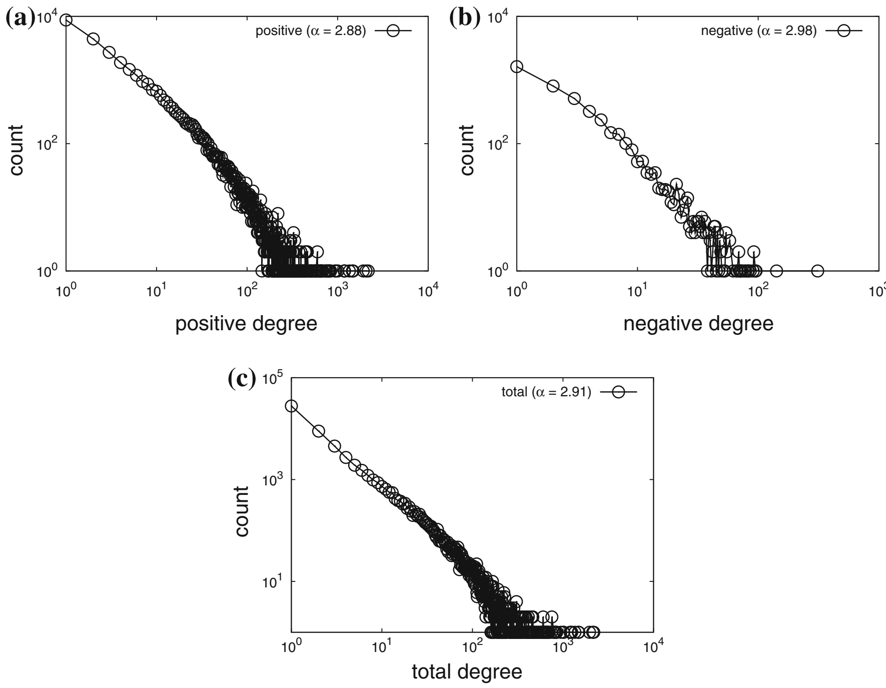


Fig. 22 Degree distribution of myGamma network. **a** Positive degree distribution, **b** negative degree distribution and **c** total degree distribution

Table 5 Descriptive statistics for myGamma and Epinions networks

Data	# Users	# Edges	$ratio^-$	Density
myGamma	68,575	293,328	0.0552	1.2e−4
Epinions	131,828	253,772	0.0185	2.9e−5

trust network is defined among these Epinion users (Leskovec et al. 2010). We download the network from the Stanford Large Network Dataset Collection.[4] We perform the same pre-processing similar to myGamma and we obtain another undirected trust network, denoted as Epinions.

The descriptive statistics of the two real networks are shown in Table 5. myGamma consists of 68,575 vertices and 293,327 links of which 5.5 % are negative. Epinions consists of 131,828 vertices and 253,772 links of which about 1.8 % are negative. To save space, Fig. 22 only shows various power-law degree distributions of myGamma network. The Epinions network follow similar property. Unless otherwise specified, the default parameter settings are $min\_size = 3$, $\epsilon = 1$ and $\delta = \frac{1}{3}$.

4 http://snap.stanford.edu/data/soc-sign-epinions.html

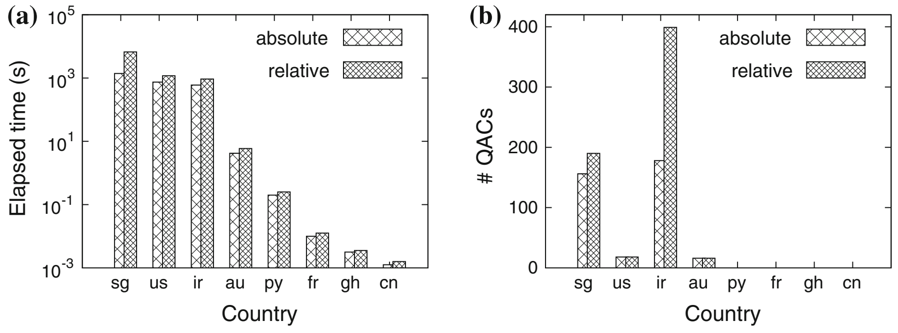


Fig. 23 Performance by varying countries. **a** Elapsed time and **b** # *QAC*s found

9.2 Performance results

9.2.1 Performance by varying countries

We apply both *aMascot* and *rMascot* on the country specific networks to determine the elapsed time and the numbers of *MQAC*s for each network.

Figure 23a illustrates the elapsed time for each country specific network. Among them, Singapore network requires the longest elapsed time, 1000 and 6000 s for *aQAC*s and *rQAC*s, respectively. This can be attributed to the large number of edges in the Singapore network (see Table 4). This results in more candidate graphs generated by the pruning stage. In the pruning stage, MASCOT therefore finds more candidate graphs. It takes longer time to process these candidate graphs in the enumeration stage. The Iran network has few edges but a high proportion of negative links. It therefore requires more elapsed time.

Figure 23b shows the number of *MQAC*s found. Iran has the largest numbers of *aQAC*s and *rQAC*s. Again, this is due to the high proportion of negative links. This result is consistent with the earlier result shown in Fig. 19.

9.2.2 Performance by varying min_size on myGamma

We apply both *aMascot* and *rMascot* on myGamma network by varying the parameters min_size, ϵ and δ. Figure 24a and c show the elapsed time of them. The elapsed time generally decreases with larger min_size and small ϵ (or δ). Nevertheless, the elapsed time is much longer for these larger real networks. This is consistent with our earlier results on the synthetic graphs. Figure 24b and d show the number of *MQAC*s found. In summary, *aMascot* and *rMascot* take several hours to finish for $min\_size = 3$, but much less time when $min\_size > 3$.

In addition, we also compare *Mascot* with the algorithm for detecting *DAC*s (Lo et al. 2013, 2011) on myGamma when $\epsilon = 0$ or $\delta = 0$. Under these conditions, the *QAC* will also be a *DAC*. We show the results of our experiments in Fig. 24a–d. From Fig. 24a and c, when min_size is large, both *aMascot* and *rMascot* are more efficient than the algorithm that mines *DAC*s. Figure 24b and d indicate that *aQAC* and *rQAC* help us to find more antagonistic communities due to their relaxed conditions. The results are consistent to our earlier results on the synthetic graph.

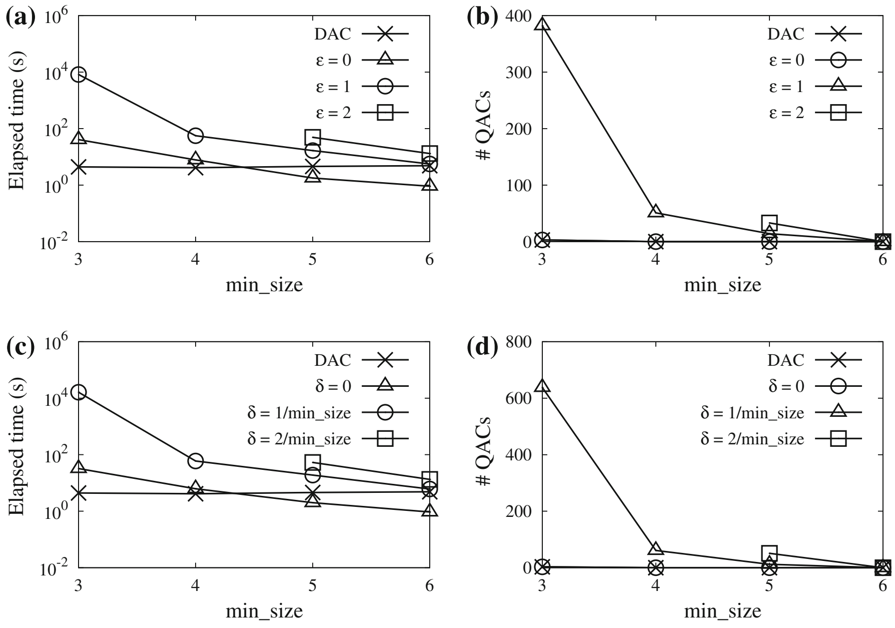


Fig. 24 Performance by varying *min_size* on myGamma. **a** Elapsed time (absolute), **b** # *QAC*s found (absolute), **c** elapsed time (relative) and **d** # *QAC*s found (relative)

9.2.3 Performance by varying min_size on Epinions

Figure 25a and c show the elapsed time by varying the parameters *min_size*, ϵ and δ on Epinions. Figure 25b and d show the number of *MQAC*s found. The observation is consistent with our earlier results on both the synthetic graphs and myGamma network. In summary, *aMascot* and *rMascot* take about a hour to finish for $min\_size = 3$, but much less time when $min\_size > 3$.

Meanwhile, we compare *Mascot* with the algorithm for detecting *DAC*s (Lo et al. 2011, 2013) on Epinions when $\epsilon = 0$ or $\delta = 0$. We also show the results of our experiments in Fig. 25a–d. From Fig. 25a and c, when *min_size* is large, *aMascot* and *rMascot* are more efficient than the algorithm that detects *DAC*s. From Fig. 25b and d, we find that *aQAC* and *rQAC* help us to find more antagonistic communities. The results are also consistent to our earlier results on the synthetic and myGamma networks.

9.3 Example cases

To gain a better understanding of the effectiveness of MASCOT. We now examine the actual *QAC*s found in myGamma networks.

9.3.1 Difference between aQAC and rQAC

Figure 26a shows one of the largest *aQAC* found with $min\_size = 3$ and $\epsilon = 1$. We apply *rMascot* with the same *min_size* and $\delta = \frac{1}{3}$. Figure 26b shows the corre-

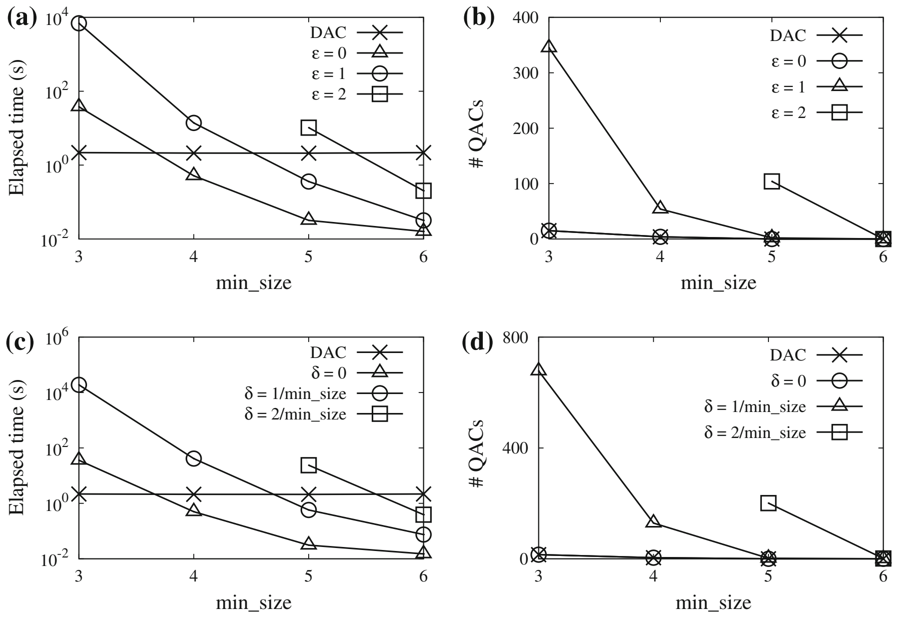


Fig. 25 Performance by varying min_size on Epinions. **a** Elapsed time (absolute), **b** # *QAC*s found (absolute), **c** elapsed time (relative) and **d** # *QAC*s found (relative)

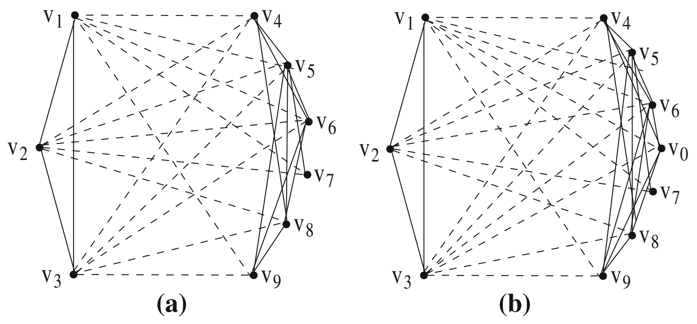


Fig. 26 Absolute and relative *QAC*s. **a** Absolute definition and **b** relative definition

sponding *rQAC*, which shares the most number of vertices with the above *aQAC*. The two *QAC*s are very similar except that v_0 is missing in the *aQAC*. This shows that the absolute and relation versions of MASCOT produce similar antagonistic communities.

9.3.2 *Cross-countries community*

Most users link to the other users from the same country. We however would like to examine if any *QAC*s exist between users from different countries. So as to obtain cross country network for finding *MQAC*s, we therefore combine users from Singapore, USA and Iran in various ways. As shown in Table 6, we cannot find any new absolute

Table 6 Statistics for detecting cross-countries *MQAC*s

Countries	Absolute		Relative	
	# *MQAC*s	Time (s)	# *MQAC*s	Time (s)
sg	156	1387.7	190	6681.1
us	18	742.9	18	1182.6
ir	178	597.9	399	930.5
sg+ir	334	1594	589	7089.3
sg+ir+us	355	2318.3	610	8271.8

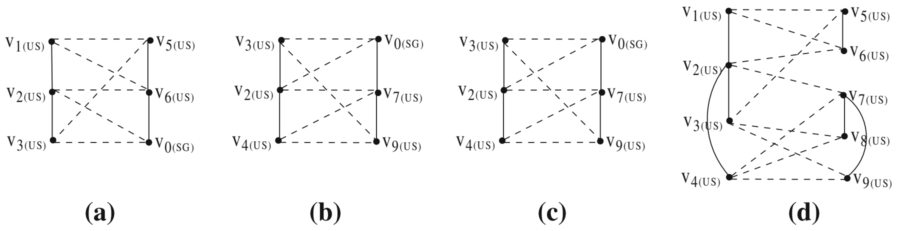


Fig. 27 *QAC*s formed by users from different countries

*MQAC*s by combining Singapore users with Iran users. When combining the users from the 3 countries, we found three additional absolute *MQAC*s (355 − 156 − 178 − 18 = 3) whose members are from different countries. These three absolute *MQAC*s are induced by the same Singapore user v_0 as shown in Fig. 27. The same observation can be found for relative *MQAC*s.

Based on the empirical results in Table 6, we can hypothesize that the elapsed time of a network consisting of users from different countries does not necessarily increase exponentially with the number of users. This is because the interactions between users from different countries are rare comparing to users from the same country. This also helps to keep MASCOT scalable.

9.4 Predicting polarity of links

We now want to determine if detecting *QAC* is helpful to predict the sign of an individual edge. We employ supervised machine learning approach to classify if an individual edge inside a detected *QAC* is positive or negative.

9.4.1 Features

To build the classifier, we group a collection of features of edges from a *QAC* into three categories. Suppose that we are interested in predicting the sign of the edge from u to v. The first group of features is related to the degrees of u and v. Specifically, we use $deg^+(u)$, $deg^+(v)$, $deg^-(u)$, $deg^-(v)$, $deg(u)$ and $deg(v)$ to denote the number of positive neighbors of u to v, the number of negative neighbors of u to v, and the total number of neighbors of u to v. The second group is the triad feature which is the set

Table 7 Accuracy for predicting signs of edges on myGamma and Epinions

Data	Approach	Precision			Recall			F-Measure		
		C_Q	C_M	N-C	C_Q	C_M	N-C	C_Q	C_M	N-C
myGamma	NaiveBayes	**.978**	.816	.653	**.979**	.901	.823	**.978**	.853	.728
	ADTree	**.992**	.927	.861	**.986**	.916	.845	**.989**	.921	.853
	J48	**.987**	.926	.864	**.994**	.936	.878	**.990**	.931	.871
	SMO	**.996**	.922	.848	**.995**	.940	.885	**.995**	.931	.866
	Logistic	**.998**	.918	.838	**.997**	.933	.868	**.997**	.925	.853
Epinions	NaiveBayes	**.962**	.838	.713	**.971**	.862	.753	**.966**	.849	.732
	ADTree	**.971**	.898	.824	**.976**	.894	.812	**.973**	.896	.818
	J48	**.979**	.920	.861	**.977**	.910	.843	**.978**	.915	.852
	SMO	**.982**	.909	.836	**.980**	.913	.846	**.981**	.911	.841
	Logistic	**.991**	.924	.856	**.987**	.918	.849	**.989**	.921	.852

Best results are highlighted in bold

of count of triads involving edge (u, v). We use $tri^{+}(u, v)$, $tri^{-}(u, v)$ and $tri^{\pm}(u, v)$ to denote the number of common positive neighbors, the number of common negative neighbors and the number of users who have different kinds of relationship to u to v. The final type of feature is a community feature which is a binary feature. It is 1 if u to v come from the same sub-community of a *QAC* or the same community detected by maximizing modularity (following the algorithm described in Mucha et al. 2010), otherwise 0.

9.4.2 *Learning methodology and results*

We build three sets of classifiers. One set is built based on degree feature and triad feature, except for the community feature, denoted as **N-C**. The second one is built based on degree feature, triad feature and *QAC*-based community feature, denoted as **C_Q**. The final one is built based on degree feature, triad feature and modularity-based community feature, denoted as **C_M**. Each set of classifiers consists of five classifiers: Naive Bayes, AD-tree, J48, Weka SVM and logistic regression.

We randomly select 600 edges from our found *MQAC*s, which consist of 317 positive edges and 283 negative edges, on both myGamma and Epinions networks. We employ 10-fold cross validation to evaluate the results. Table 7 illustrates precision, recall, and F-measure of the learning results on both myGamma and Epinions networks, where F-measure is computed as $\frac{2\times precision\times recall}{precision+recall}$. We observe that the predicted results with the community feature outperform those without the community feature. Also, our *QAC*-based community feature outperforms the modularity-based community feature in predicting the polarities of links.

9.5 Discussion: coverage and applicability

The goal of our work is to identify strong local communities that fight among one another. In a typical network, nodes involved in these antagonistic communities are expected to be small in number—since otherwise there would be too much fights and the network might even crumble into multiple networks. Thus, by design, our antagonistic communities will not have high coverage. Most of the nodes in a typical network would not be part of any antagonistic community.

However, despite low coverage, antagonistic community mining still have a number of applications. Detecting antagonistic communities would help a network administrator to detect unwanted antagonism early and prevent it to spread too much in the network. By detecting antagonistic communities, one can also study factors that lead to antagonism in a large network which will enrich existing studies in social science that have typically only analyzed small datasets. Antagonistic communities can also be helpful to predict link polarity, user preferences, product adoption, etc. Of course, the low coverage of antagonistic communities might limit their utility in these applications; however, for nodes that are covered by these antagonistic communities (which could still be a large number, e.g., thousands of nodes), the detected communities can help in these tasks—as shown in Sect. 9.4.

10 Conclusion

The importance of local structures in a graph has been recognized in many application areas, such as understanding user behavior and predicting links between users.

In this paper, we present a comprehensive study of finding maximal quasi antagonistic communities (*MQAC*s). We define both the absolute and relative versions of *QAC* and propose two-stage MASCOT algorithm to find all *MQAC*s: pruning and enumeration stages.

In the pruning stage, we introduce four pruning rules to reduce the number and size of candidate graphs. The experimental results tell us that our pruning rules have high pruning efficiency.

In the enumeration stage, we propose an enumeration tree in a top–down manner to enumerate all *SCCs* of each vertex set of a candidate graph.

We conduct an extensive set of experiments on both synthetic graphs and two real networks.

Our results show that MASCOT can efficiently handle an input signed graph with hundreds of thousands of vertices and millions of edges.

We also demonstrate that MASCOT achieves good recall, returning the *QAC*s injected into synthetic graphs.

And in the myGamma social network, a real world network, we observe that most *QAC*s are among users from the same country.

Finally, we find that detecting *QAC*s is helpful to predict the sign of an individual edge from both myGamma and Epinions networks.

Acknowledgments This work is supported by the National Basic Research Program (973) of China (No. 2012CB316203) and NSFC under Grant Nos. 61033007, 61103039, 61402177 and 61232002. This work is also supported by the National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, and National Research Foundation (NRF) (NRF2008IDM-IDM004-036). We are grateful to the anonymous reviewers for their valuable and insightful comments. We would also like to thank BuzzCity Pte Ltd for sharing their data set with us.

References

Abello J, Resende MGC, Sudarsky S (2002) Massive quasi-clique detection. In: Latin American theoretical informatics symposium, pp 598–612

Alba RD (1973) A graph-theoretic definition of a socimetric clique. J Math Sociol 3:113–126

Alvarez-Hamelin I, DallÁsta L, Barrat A, Vespignani A (2008) K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. Netw Heterog Media 3(2):371–393

Anchuri P, Magdon-Ismail M (2012) Communities and balance in signed networks: a spectral approach. In: ASONAM, pp 235–242

Ball B, Karrer B, Newman MEJ (2011) An efficient and principled method for detecting communities in networks. Phys Rev E 84:36103

Bansal N, Blum A, Chawla S (2004) Correlation clustering. Mach Learn 56(1–3):89–113

Beyene Y, Faloutsos M, Chau P, Faloutsos C (2008) The ebay graph: How do online auction users interact? In: Computer communications workshops, pp 13–18

Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech 10:1–12

Cartwright D, Harary F (1956) Structure balance: a generalization of Heider's theory. Psychol Rev 63(5):277–293

Coen B, Joep K (1973) Algorithm 457: finding all cliques of an undirected graph. Commun ACM 16(9):575–577

Dandekar P (2010) Analysis and generative model for trust networks. Technique Report, pp 1–5

Donetti L, Muñoz MA (2004) Detecting network communities: a new systematic and efficient algorithm. J Stat Mech: P10012(cond-mat/0404652)

Doreian P, Mrvar A (1996) A partitioning approach to structural balance. Soc Netw 18(2):149–168

Everett M (1982) Graph theoretic blockings, k-plexes and k-cutpoints. J Math Sociol 9:75–84

Giatsidis C, Thilikos DM, Vazirgiannis M (2011) Evaluating cooperation in communities with the k-core structure. In: 2011 International conference on advances in social networks analysis and mining, pp 87–93

Girvan M, Newman MEJ (2004) Finding and evaluating community structure in networks. Phys Rev E 69:026113

Groshaus M, Szwarcfiter JL (2010) Biclique graphs and biclique matrices. J Graph Theory 63(1):1–16

Heider F (1946) Attitudes and cognitive organization. J Psychol 21:107–112

Jamali M, Abolhassani H (2006) Different aspects of social network analysis. In: Web intelligence, pp 66–72

Johnson DS, Yanakakis M, Papadimitriou CH (1988) On generating all maximal independent sets. Inf Process Lett 27(3):119–123

Karrer B, Newman MEJ (2011) Stochastic blockmodels and community structure in networks. Phys Rev E 83:016107

Leicht EA, Girvan M, Newman MEJ (2006) Vertex similarity in networks. Phys Rev E 73:026120

Leskovec J, Huttenlocher DP, Kleinberg JM (2010) Signed networks in social media. In: CHI, pp 1361–1370

Li J, Sim K, Liu G, Wong L (2008) Maximal quasi-bicliques with balanced noise tolerance: concepts and co-clustering applications. In: SIAM international conference on data mining, pp 72–83

Liu G, Wong L (2008) Effective pruning techniques for mining quasi-cliques. In: European conference on machine learning and knowledge discovery in databases, pp 33–49

Liu X, Li J, Wang L (2008) Quasi-bicliques: complexity and binding pairs. In: The 14th annual international computing and combinatorics conference, pp 255–264

Lo D, Surian D, Prasetyo PK, Zhang K, Lim EP (2013) Mining direct antagonistic communities in signed social networks. Inf Process Manag 49(4):773–791

Lo D, Surian D, Zhang K, Lim EP (2011) Mining direct antagonistic communities in explicit trust networks. In: ACM conference on information and knowledge management, pp 1043–1054

Luce RD (1950) Connectivity and generalized cliques in sociometric group structure. Psychometrika 15:169–190

Luce RD, Perry AD (1949) A method of matrix analysis of group structure. Psychometrika 14(2):95–116

Mokken RJ (1979) Cliques, clubs and clans. Qual Quant 13:161–173

Moon JW, Moser L (1965) On cliques in graphs. Isr J Math 3:23–28

Mucha PJ, Porter MA (2010) Communities in multislice voting networks. Chaos 20:041108

Mucha PJ, Richardson T, Macon K, Porter MA, Onnela JP (2010) Community structure in time-dependent, multiscale, and multiplex networks. Science 328:876–878

Palla G, Derényi I, Farkas I, Vicsek T (2005) Uncovering the overlapping community structure of complex networks in nature and society. Nature 435:814–818

Palmer CR, Steffan JG (2000) Generating network topologies that obey power laws. In: IEEE globecom 2000, pp 33–37

Ronhovde P, Nussinov Z (2009) Multiresolution community detection for megascale networks by information-based replica correlations. Phys Rev E Stat Nonlinear Soft Matter Phys 80:016,109–19658776

Sim K, Li J, Gopalkrishnan V, Liu G (2006) Mining maximal quasi-bicliques to co-clustering stocks and financial ratios for value investment. In: IEEE international conference on data mining, pp 1059–1063

Tarjan RE (1972) Depth-first search and linear graph algorithms. SIAM J Comput 1(2):146–160

Traag VA, Bruggeman J (2009) Community detection in networks with positive and negative links. Phys Rev E 80:036115

Wasserman S, Faust K (1994) Social network analysis: methods and applications. Cambridge University Press, Cambridge

Zhang K, Lo D, Lim EP (2010) Mining antagonistic communities from social networks. In: The 14th Pacific-Asia conference on knowledge discovery and data, pp 68–80

Zhang K, Lo D, Lim EP, Prasetyo PK (2013) Mining indirect antagonistic communities from social interactions. Knowl Inf Syst 35(3):553–583