Signed Clique Search in Signed Networks: Concepts and Algorithms

Rong-Hua Li, Qiangqiang Dai, Lu Qin, Guoren Wang, Xiaokui Xiao, Jeffrey Xu Yu, and Shaojie Qiao

Abstract—Mining cohesive subgraphs from a network is a fundamental problem in network analysis. Most existing cohesive subgraph models are mainly tailored to unsigned networks. In this paper, we study the problem of seeking cohesive subgraphs in a signed network, in which each edge can be positive or negative, denoting friendship or conflict, respectively. We propose a novel model, called maximal (α, k)-clique, that represents a cohesive subgraph in signed networks. Specifically, a maximal (α, k)-clique is a clique in which every node has at most k negative neighbors and at least $\lceil \alpha k \rceil$ positive neighbors ($\alpha \geq 1$). We show that the problem of enumerating all maximal (α, k)-cliques in a signed network is NP-hard. To enumerate all maximal (α, k)-cliques efficiently, we first develop an elegant signed network reduction technique to significantly prune the signed network. Then, we present an efficient branch and bound enumeration algorithm with several carefully-designed pruning rules to enumerate all maximal (α, k)-cliques in the reduced signed network. In addition, we also propose an efficient algorithm with three novel upper-bounding techniques to find the maximum (α, k)-clique in a signed network. The results of extensive experiments on five large real-life datasets demonstrate the efficiency, scalability, and effectiveness of our algorithms.

Index Terms—Signed clique, signed network, maximal clique enumeration, branch and bound algorithm

1 Introduction

Real-life networks, such as social networks and web graphs, typically involve cohesive subgraph structures. Discovering cohesive subgraphs in a network is a fundamental problem in network analysis, and is useful in numerous applications including community discovery [1], [2], protein complex mining [3], spam detection [4], and so on.

In applications such as trust networks analysis [5], opinion networks mining [6], online social networks analysis [6], as well as protein-protein interaction (PPI) networks analysis [3], the edges in these networks can be either positive representing friendship, or negative representing antagonism. Finding cohesive subgraphs in these signed networks can be used to detect community structures [7], study trust dynamics [5], or identify protein complexes [4], etc. Unfortunately, most existing cohesive subgraph models, such as maximal clique [8], k-core [9], and k-truss [10], ignore the signed edge information that might be inappropriate for characterizing the cohesive subgraphs in a signed network.

Recently, Giatsidis et al. [5] proposed a signed core model to capture the signed edge information in a cohesive subgraph. The signed core is a maximal subgraph C such that each node in C has at least β positive neighbors and also has more than γ negative neighbors, where β and γ are two integer parameters. The main deficiencies of the signed core model are twofold. First, a signed core could contain too many negative edges. Second, the signed core may be not very compact when β and γ are small.

Intuitively, a cohesive subgraph in the signed network should be densely-connected. It should involve many positive edges, but not too many negative edges. For example, in applications related to community detection [7] or community search [1], we may wish to find a community such that most links have positive edges and few negative edges. Based on this intuition, we have developed a novel cohesive subgraph model for signed networks, called maximal (α, k)-clique. A maximal (α, k)-clique satisfies three properties: (i) it is a clique in which every pair of nodes has a connection; (ii) every node in a maximal (α, k)-clique has at most k negative neighbors (foes) and at least $\lceil \alpha k \rceil$ ($\alpha \geq 1$) positive neighbors (friends); and (iii) it is a maximal subgraph that meets (i) and (ii). Clearly, the maximal (α, k)-clique can limit the number of negative edges and it is also compact in terms of the clique property. In the experiments, we show that the maximal (α, k)-clique model is able to identify interesting cohesive subgraphs in many signed network analysis applications. This type of cohesive subgraph could be very useful for discovering trust communities in a

- R.-H. Li, Q. Dai, and G. Wang are with the Beijing Institute of Technology, Beijing 100081, China. E-mail: lironghuascut@gmail.com, qiang56734@163.com, wanggrbit@126.com.
- L. Qin is with the University of Technology Sydney, Ultimo, NSW 2007, Australia. E-mail: Lu.Qin@uts.edu.au.
- X. Xiao is with the National University of Singapore, Singapore 119077. E-mail: xkxiao@nus.edu.sg.
- J.X. Yu is with the Chinese University of Hong Kong, Shatin, NT, Hong Kong. E-mail: yu@se.cuhk.edu.hk.
- S. Qiao is with the Chengdu University of Information Technology, Chengdu, Sichuan Sheng 610225, China. E-mail: sjqiao@cuit.edu.cn.

Manuscript received 9 Sept. 2018, accepted 4 Mar. 2019, Date of publication 0 . 0000; date of current version 0 . 0000.
(Corresponding author: Guoren Wang.)
Recommended for acceptance by P. K. Chrysanthis, B. C. Ooi, and J. Dittrich.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2019.2904569

trust network, revealing interesting protein complexes in signed PPI networks, and for detecting strongly-cooperative research groups in collaboration networks.

Trust Community Mining. In a trust network, such as Epinions (www.epinions.com), users can express trust or distrust of other users. By finding the maximal (α, k)-cliques, the trust communities with the most users who have rated each other positively could be identified. After discovering those trust communities, a company could perform powerful viral marketing to promote their products by influencing just a small portion of its users because most of those users trust each other.

Protein Complex Discovery. In a signed PPI network, a protein complex can be represented as a densely-connected subgraph, in which most protein-protein interactions exhibit a *positive* relationship (e.g., a common function relationship) and few interactions show a *negative* relationship (e.g., inhibition relationships) [3]. By identifying the maximal (α, k)-cliques, the protein complexes can be discovered in the signed PPI network, as the model clearly represents a cohesive subgraph containing many positive edges and few negative edges.

Finding Strongly Cooperative Research Groups. To identify strongly cooperative research groups in a co-authorship network (e.g., DBLP), the network could be modeled as a signed network, where the positive and negative edges represent strong and weak cooperative relationships. For example, if two researchers co-author many/few papers, the cooperative relationship between them can be modeled as a positive/negative edge. By seeking the maximal (α, k)-cliques, strongly cooperative groups can be discovered as the model consists of many strong ties and only few weak links.

Contributions. In this paper, we formulate and provide efficient solutions for two fundamental problems of seeking cohesive subgraphs in a signed network: (i) enumerating all maximal (α, k)-cliques, and (ii) finding the maximum (α, k)-clique. The main contributions of this paper are summarized as follows.

New model. We propose a novel maximal (α, k)-clique model that represents a cohesive subgraph in signed networks. We show that the classic maximal clique is a special case of the maximal (α, k)-clique. Since the classic maximal clique enumeration problem is NP-hard, our problems are also NP-hard.

Novel algorithms. To compute the maximal (α, k)-cliques, we develop an elegant signed graph reduction technique to substantially prune the signed network. We show that our signed graph reduction algorithm takes $O(\delta m)$ and uses $O(m+n)$ space, where δ denotes the arboricity, m is the number of edges, and n denotes the number of nodes of the graph. Note that the arboricity δ is bounded by $O(\sqrt{m})$ [11], and it is often much smaller than such a worst-case bound in real-life graphs [12]. In the reduced signed network, we propose a new branch and bound enumeration algorithm with several carefully-designed pruning strategies to efficiently enumerate all maximal (α, k)-cliques. We also develop an efficient algorithm with three novel upper-bounding techniques to identify the maximum (α, k)-clique in a signed network.

Extensive experimental results. We conduct comprehensive experimental studies to evaluate the proposed algorithms using five large real-world datasets. The results show that our algorithm takes less than 1,000 seconds to enumerate all maximal (α, k)-cliques under most parameter settings in a signed network with more than 1.6 million nodes and 30.6 million edges (in the same dataset, our algorithm takes less than 100 seconds to find the maximum (α, k)-clique). Based on the traditional conductance [13] metric, we introduce a new and intuitive metric, called *signed conductance*, to measure the quality of a cohesive subgraph. We show that the proposed model consistently outperforms the baselines in terms of the signed conductance metric. We also examine several case studies to evaluate the effectiveness of our model. The results indicate that our model is able to identify intuitive and compact communities in signed networks that cannot be found by the baseline models.

Organization. Section 2 introduces the maximal (α, k)-clique model. The signed graph reduction technique is proposed in Section 3. Section 4 presents the branch and bound enumeration algorithm. The maximum (α, k)-clique search algorithm is shown in Section 5. The experimental results are reported in Section 6. We review the related work in Section 7, and conclude this work in Section 8.

2 Problem Statement

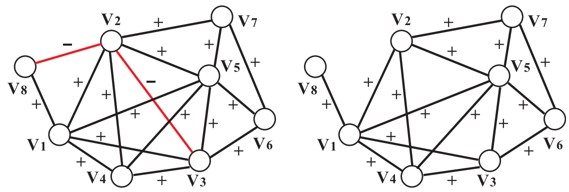
Let $G = (V, E)$ be an undirected signed network, where V ($|V| = n$) and E ($|E| = m$) denote the set of nodes and edges respectively. In G, each edge $e \in E$ is associated with a label either "+" or "−". An edge with label "+" denotes a positive edge, while an edge with label "−" denotes a negative edge. Let $N_u \triangleq \{v | (u, v) \in E\}$ be the set of neighbor nodes of u, $N_u^+ \triangleq \{v | (u, v) \in E, \textit{ and } (u, v) \textit{ is a positive edge}\}$ be the set of positive neighbors, and $N_u^- \triangleq \{v | (u, v) \in E, \textit{ and } (u, v) \textit{ is a negative edge}\}$ be the set of negative neighbors. Let $d_u(G) = |N_u|$, $d_u^+(G) = |N_u^+|$, $d_u^-(G) = |N_u^-|$, be the degree, the positive degree, and the negative degree of u in G respectively. A subgraph $H = (V_H, E_H)$ is called an induced subgraph of G if $V_H \subseteq V$ and $E_H = \{(u, v) | (u, v) \in E, u \in V_H, v \in V_H\}$. An induced subgraph H of G is a clique if every pair of nodes in H has an edge, i.e., $(u, v) \in E$ for any $u \in H$ and $v \in H$. Given a signed network G and an integer k, a k-core, denoted by C_k, is an induced subgraph of G such that every node in C_k has a degree no less than k, i.e., $d_u(C_k) \geq k$ for every $u \in C_k$ [9]. A maximal k-core C_k is a k-core such that there is no k-core C_k' in G that contains C_k [9].

Intuitively, an interesting cohesive subgraph in signed networks should be densely connected. It should consist of many positive edges and not contain too many negative edges. Based on this intuition, we propose a new model, called maximal (α, k)-clique, to describe the cohesive subgraphs in a signed network.

Definition 1. ((α, k)-clique) *Given a signed graph G, a positive real value α ($\alpha \geq 1$), and an integer k, an (α, k)-clique is an induced subgraph C that satisfies the following constraints.*

- Clique constraint*: C is a clique in G;*
- Negative-edge constraint*: for each $u \in C$, $d_u^-(C) \leq k$;*
- Positive-edge constraint*: for each $u \in C$, $d_u^+(C) \geq \alpha k$.*

In Definition 1, the clique constraint ensures that the subgraph is densely-connected. The negative-edge constraint



(a) Signed graph (b) Positive-edge graph

Fig. 1. Running example (red edges denote negative edges).

imposes a limit that every node cannot have too many negative neighbors in the subgraph, and the positive-edge constraint guarantees that every node has a sufficient number of positive neighbors in the subgraph. Based on Definition 1, we define the maximal (α, k)-clique below.

Definition 2. (Maximal (α, k)-clique) *An induced subgraph C is a maximal (α, k)-clique if C is an (α, k)-clique and there is no (α, k)-clique C' in G containing C.*

Example 1. Consider a signed network shown in Fig. 1a. Suppose that $\alpha = 3$ and $k = 1$. We can easily derive that $\{v_1, v_2, v_3, v_4, v_5\}$ is a $(3, 1)$-clique. Moreover, it is a maximal $(3, 1)$-clique, because there is no super clique that can contain it. Similarly, if $\alpha = 3$ and $k = 0$, we have two maximal $(3, 0)$-cliques which are $\{v_1, v_2, v_4, v_5\}$ and $\{v_1, v_3, v_4, v_5\}$. Note that in this case, $\{v_1, v_2, v_3, v_4, v_5\}$ is no longer a $(3, 0)$-clique, as the node v_2 violates the negative-edge constraint.

Let $\mathcal{C}$ be the set of all (α, k)-cliques in the signed network G. The (α, k)-clique in $\mathcal{C}$ with the largest size is referred to as the maximum (α, k)-clique. In this paper, we aim to find all maximal (α, k)-cliques and the maximum (α, k)-clique in a signed network. Specifically, we formulate our problem as follows.

Problem Statement. Given a signed network G and the parameters α, k and r, our goal is to develop efficient algorithms to settle the following two fundamental problems: 1) enumerate all maximal (α, k)-cliques in G; and 2) identify the maximum (α, k)-clique in G.

Note that the maximum (α, k)-clique search problem can be solved easily if we can enumerate all maximal (α, k)-cliques. Below, we focus mainly on analyzing the hardness and challenges of the maximal (α, k)-clique enumeration problem.

Hardness and Challenges. First, we show that the traditional maximal clique enumeration problem [8], [14], [15], [16] is a special case of the maximal (α, k)-cliques enumeration problem. Suppose that $\alpha = 0$ and $k = d^-_{\max}$, where $d^-_{\max}$ is the largest negative degree in G. Given this parameter setting, a maximal (α, k)-clique degrades to a traditional maximal clique. This is because both the negative-edge and positive-edge constraints in Definition 1 always hold when $\alpha = 0$ and $k = d^-_{\max}$. As a result, enumerating all maximal (α, k)-cliques is equivalent to enumerating all traditional maximal cliques if $\alpha = 0$ and $k = d^-_{\max}$. Therefore, the classic maximal clique enumeration problem is a special case of our problem when the parameters $\alpha = 0$ and $k = d^-_{\max}$. Since the traditional maximal clique enumeration problem is NP-hard, our problem is also NP-hard.

Although there is a close connection between our problem and the maximal clique problem, the existing maximal clique enumeration algorithms cannot be immediately applied to solve our problem. This is because the traditional clique enumeration algorithms, such as the classic Bron-Kerbosch algorithm and its variants [14], [15], [16], can only enumerate all maximal cliques, but they cannot guarantee that all sub-cliques contained in the maximal cliques will be explored. Since a maximal (α, k)-clique can be a sub-clique of any maximal clique in the signed network, the traditional clique enumeration algorithms cannot be directly used for our problem. To solve our problem, a straightforward method is to find all the traditional maximal cliques first, and then and then enumerate all the maximal (α, k)-cliques in C for each traditional maximal clique C. However, this method is intractable for large signed graphs because the number of traditional maximal cliques in a signed graph may be very large and many maximal (α, k)-cliques contained in C may exist for each traditional maximal clique C. Moreover, this straightforward method may generate numerous redundant maximal (α, k)-cliques because the same maximal (α, k)-clique could be contained in many overlapped traditional maximal cliques. Therefore, the main challenge of our problem is how to efficiently enumerate every maximal (α, k)-clique only once. Several powerful pruning techniques and a novel branch and bound algorithm to tackle this challenge are presented below.

3 Signed Graph Reduction

In this section, we propose several effective rules to prune the unpromising nodes that are definitely not contained in any maximal (α, k)-clique. Let $G^+ = (V, E^+)$ be the subgraph of $G = (V, E)$ that contains all the positive edges in G, in which $E^+ \triangleq \{(u, v) | (u, v) \in E$, *and* (u, v) *is a positive edge*$\}$. For convenience, we refer to G^+ as the positive-edge graph of G. For example, Fig. 1b depicts a positive-edge graph of the signed graph shown in Fig. 1a.

Based on the k-core concept in [9], the maximal positive-edge $\lceil \alpha k \rceil$-core is defined as the maximal induced subgraph of G such that every node in this subgraph has a positive degree no less than $\lceil \alpha k \rceil$. Clearly, by this definition, the node set of the maximal positive-edge $\lceil \alpha k \rceil$-core in G is the same as the node set of the maximal $\lceil \alpha k \rceil$-core in G^+. Below, we show that all maximal (α, k)-cliques are contained in the maximal positive-edge $\lceil \alpha k \rceil$-core of G.

Lemma 1. *Any maximal (α, k)-clique is contained in a connected component of the maximal positive-edge $\lceil \alpha k \rceil$-core of G.*

Proof. Clearly, each node in the maximal (α, k)-clique has $\lceil \alpha k \rceil$ positive neighbors (see Definition 1). Thus, the maximal (α, k)-clique forms an $\lceil \alpha k \rceil$-core. Since any maximal (α, k)-clique is connected, it must be contained in a connected component of the maximal positive-edge $\lceil \alpha k \rceil$-core of G. □

To compute maximal (α, k)-cliques, we are able to reduce the signed graph based on Lemma 1. Specifically, we can first compute the maximal $\lceil \alpha k \rceil$-core in G^+, because its node set is the same as that of the maximal positive-edge $\lceil \alpha k \rceil$-core in G. Then, we prune all the nodes in G that are not contained in the maximal $\lceil \alpha k \rceil$-core of G^+.

Example 2. Reconsider the signed graph in Fig. 1a. Suppose that $\alpha = 3$ and $k = 1$. We can easily figure out that there is a maximal $\lceil \alpha k \rceil$-core $\{v_1, \ldots, v_7\}$ in the positive-edge graph G^+ (see Fig. 1b). Obviously, $\{v_1, \ldots, v_7\}$ is also a maximal positive-edge $\lceil \alpha k \rceil$-core in G. Based on the maximal positive-edge $\lceil \alpha k \rceil$-core, we can safely prune the node v_8 to compute maximal (α, k)-cliques, as v_8 is definitely excluded in any maximal (α, k)-clique.

Although the maximal positive-edge $\lceil \alpha k \rceil$-core excludes many unpromising nodes, it may still be not very powerful for pruning. For example, in Fig. 1b, the nodes v_6 and v_7 are clearly not contained in any maximal (α, k)-clique when $\alpha = 3$ and $k = 1$, but the maximal positive-edge $\lceil \alpha k \rceil$-core fails to prune these two nodes. Below, we propose a more effective approach to further prune unpromising nodes.

3.1 The MCCore Pruning Rule

Here, we present a new cohesive subgraph model, called maximal constrained $\lceil \alpha k \rceil$-core, to further prune unpromising nodes for the maximal (α, k)-clique enumeration problem. We abbreviate the maximal constrained $\lceil \alpha k \rceil$-core as MCCore, when it is clear from the context. The key idea of the MCCore model is based on the following result.

Lemma 2. *Let C be an (α, k)-clique. Then, for each node $u \in C$, the subgraph induced by $N_u^+(G)$ must contain an $(\lceil \alpha k \rceil - 1)$-core.*

Proof. By Definition 1, the neighbors of node u in C ($N_u^+(C)$) form a $d_u^+(C)$-clique. Thus, the subgraph induced by $N_u^+(G)$ must contain a $d_u^+(C)$-clique. Since $u \in C$, we have $d_u^+(C) \geq \lceil \alpha k \rceil$. As a result, the subgraph induced by $N_u^+(G)$ must contain an $(\lceil \alpha k \rceil - 1)$-core. □

From Lemma 2, we can immediately obtain the following corollary.

Corollary 1. *For each node $u \in V$, if the subgraph induced by $N_u^+(G)$ does not contain an $(\lceil \alpha k \rceil - 1)$-core, u cannot be involved in any (α, k)-clique.*

Armed with Corollary 1, we can prune the node from G if the subgraph induced by its positive neighbors cannot include an $(\lceil \alpha k \rceil - 1)$-core. Note that after removing all these unpromising nodes, some of the remaining nodes in G may become unpromising. Thus, this pruning procedure can iterate until no further nodes can be pruned. We will show that the remaining nodes form a maximal constrained $\lceil \alpha k \rceil$-core when this iterative pruning procedure terminates. The maximal constrained $\lceil \alpha k \rceil$-core is formally defined as follows.

Definition 3. (Maximal constrained $\lceil \alpha k \rceil$-core) *Given a signed graph G, a positive real value α, and an integer k, a maximal constrained $\lceil \alpha k \rceil$-core R is an induced subgraph of G that meets the following constraints.*

- Neighbor-core constraint: *for each $u \in R$, the subgraph induced by $N_u^+(R)$ contains an $(\lceil \alpha k \rceil - 1)$-core;*
- Maximal constraint: *there does not exist an induced subgraph in G that contains R and also satisfies the neighbor-core constraints.*

Below, we show that all maximal (α, k)-cliques are contained in the maximal constrained $\lceil \alpha k \rceil$-core.

Lemma 3. *Any maximal (α, k)-clique must be contained in a connected component of the maximal constrained $\lceil \alpha k \rceil$-core of G.*

Proof. For each node u in a maximal (α, k)-clique C, the positive neighbors of u in C must be a $d_u^+(C)$-clique. Thus, every node in C satisfies the neighbor-core constraint. Since C is connected, it must be included in a connected component of the maximal constrained $\lceil \alpha k \rceil$-core of G. □

According to Lemma 3, we can prune all the nodes that are not contained in the maximal constrained $\lceil \alpha k \rceil$-core. Note that the maximal constrained $\lceil \alpha k \rceil$-core is more effective than the maximal positive-edge $\lceil \alpha k \rceil$-core to prune unpromising nodes. The reason is as follows. By Definition 3, we can easily obtain that $d_u^+(R) \geq \lceil \alpha k \rceil$ for every node u in a maximal constrained $\lceil \alpha k \rceil$-core R on the basis of the neighbor-core constraint. As a result, the maximal constrained $\lceil \alpha k \rceil$-core of G must be contained in the maximal positive-edge $\lceil \alpha k \rceil$-core of G. That is to say, the maximal constrained $\lceil \alpha k \rceil$-core can prune more unpromising nodes than the maximal positive-edge $\lceil \alpha k \rceil$-core.

Example 3. Reconsider the signed graph in Fig. 1a. Assume that $\alpha = 3$ and $k = 1$. We can see that the node v_7 violates the neighbor-core constraint, because the subgraph induced by its positive neighbors $\{v_2, v_5, v_6\}$ cannot consist of a 2-core. Thus, v_7 cannot be contained in the maximal constrained $\lceil \alpha k \rceil$-core. Likewise, v_6 and v_8 can also be pruned. It is easy to verify that $\{v_1, \ldots, v_5\}$ is a maximal constrained $\lceil \alpha k \rceil$-core. Clearly, compared to the maximal positive-edge $\lceil \alpha k \rceil$-core, the maximal constrained $\lceil \alpha k \rceil$-core can prune more nodes (v_7 and v_8) in this example.

3.2 The MCBasic Algorithm

To compute the MCCore, we can first compute the maximal positive-edge $\lceil \alpha k \rceil$-core denoted by S, as S contains the MCCore. Then, we check whether or not u satisfies the neighbor-core constraint for each node $u \in S$. Specifically, we create a subgraph S_u^+ induced by u's positive neighbors in S ($N_u^+(S)$), and calculate the $(\lceil \alpha k \rceil - 1)$-core in S_u^+. If S_u^+ does not contain an $(\lceil \alpha k \rceil - 1)$-core, we delete u from S. Since the deletion of u may result in u's neighbors no longer meeting the neighbor-core constraint, we need to iteratively process u's neighbors. The processing terminates if no node can be deleted. The details are provided in Algorithm 2.

Algorithm 2 first invokes Algorithm 1 to compute the maximal $\lceil \alpha k \rceil$-core in G^+. Note that Algorithm 1 admits three input parameters $\{H, I, \tau\}$, where H is a graph, I is a set of fixed nodes, and τ is an integer. Algorithm 1 aims at computing the maximal τ-core in H such that it must contain all nodes in I. If no such a τ-core exists, the algorithm returns a Boolean constant 0 and an empty set. To compute a traditional maximal τ-core in H, we can invoke Algorithm 1 with an empty fixed nodes set, i.e., $I = \emptyset$.

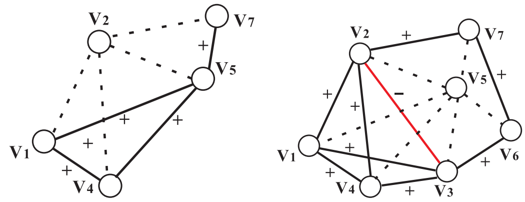
Algorithm 1. ICore ($H = (V_H, E_H)$, I, τ)

Input: Graph $H = (V_H, E_H)$, fixed nodes I, and an integer τ
Output: An boolean constant and the node set of the τ-core
1: $D \leftarrow \emptyset$; $\mathcal{Q} \leftarrow \emptyset$;
2: **for** each $v \in V_H$ **do**
2:   **if** $d_v(H) < \tau$ **then**
4:     **if** $v \in I$ **then return** $(0, \emptyset)$;
5:     $\mathcal{Q}.push(v)$;
6: **while** $\mathcal{Q} \neq \emptyset$ **do**
7:   $u \leftarrow \mathcal{Q}.pop()$; $D \leftarrow D \cup \{u\}$;
8:   **for** each $v \in N_u(H)$ s.t. $d_v(H) \geq \tau$ **do**
9:     $d_v(H) \leftarrow d_v(H) - 1$;
10:     **if** $d_v(H) < \tau$ **then**
11:       **if** $v \in I$ **then return** $(0, \emptyset)$;
12:       $\mathcal{Q}.push(v)$;
13: $V_H \leftarrow V_H \backslash D$;
14: **if** $V_H = \emptyset$ **then return** $(0, \emptyset)$;
15: **return** $(1, V_H)$;

Algorithm 2. MCBasic (G, α, k)

Input: $G = (V, E)$, α, and k
Output: The node set of the maximal constrained $\lceil \alpha k \rceil$-core
1: $(flag, V_R) \leftarrow$ ICore $(G^+, \emptyset, \lceil \alpha k \rceil)$; /* compute the $\lceil \alpha k \rceil$-core in G^+ */
2: Let R be the subgraph induced by V_R;
3: Let $d_v^+(R)$ be the positive degree of v in the subgraph R;
4: $f_u \leftarrow 1$ for all $u \in V_R$;
5: $X \leftarrow \emptyset$; $\mathcal{Q} \leftarrow \emptyset$; /* $\mathcal{Q}$ is a queue */
6: **for** each $u \in V_R$ **do**
7:   Let R_u^+ be the subgraph induced by $N_u^+(R)$; /* ego network of u in R */
8:   $(flag, S_u) \leftarrow$ ICore $(R_u^+, \emptyset, \lceil \alpha k \rceil - 1)$;
9:   **if** $flag = 0$ **then** $\mathcal{Q}.push(u)$; $f_u \leftarrow 0$;
10: **while** $\mathcal{Q} \neq \emptyset$ **do**
11:   $u \leftarrow \mathcal{Q}.pop()$; $X \leftarrow X \cup \{u\}$;
12:   **for** each $v \in N_u^+(R)$ s.t. $f_v = 1$ **do**
13:     $d_v^+(R) \leftarrow d_v^+(R) - 1$;
14:     **if** $d_v^+(R) < \lceil \alpha k \rceil$ **then**
15:       $\mathcal{Q}.push(v)$; $f_v \leftarrow 0$; /* degree pruning */
16:     **else**
17:       Let $\tilde{R}_v^+$ be the subgraph induced by $N_v^+(R) \setminus \{u\}$;
18:       $(flag, S_v) \leftarrow$ ICore $(\tilde{R}_v^+, \emptyset, \lceil \alpha k \rceil - 1)$;
19:       **if** $flag = 0$ **then** $\mathcal{Q}.push(v)$; $f_v \leftarrow 0$;
20: $V_R \leftarrow V_R \setminus X$;
21: **return** V_R;

Algorithm 2 makes use of a queue $\mathcal{Q}$ to maintain all nodes that need to be deleted (line 5). The iterative node-pruning procedure is shown in lines 10-19. Note that Algorithm 2 also applies a degree pruning rule to optimize efficiency (lines 14-15). Specifically, when the algorithm processes a node u, it first computes its positive degree. If the positive degree is smaller than $\lceil \alpha k \rceil$, the subgraph induced by its positive neighbors cannot contain an $(\lceil \alpha k \rceil - 1)$-core, and thus u can be directly deleted without invoking Algorithm 1 to compute the $(\lceil \alpha k \rceil - 1)$-core (lines 14-15). The following theorem shows the correctness of Algorithm 2.



(a) ego network of v_2 (b) ego network of v_5

Fig. 2. Illustration of the definition of ego network (solid lines).

Theorem 1. *Algorithm 2 correctly computes the maximal constrained $\lceil \alpha k \rceil$-core.*

Proof. Let R be the results obtained by Algorithm 2. First, we claim that if the MCCore does not exist, Algorithm 2 outputs $R = \emptyset$. This can be proven by contradiction. Suppose, to the contrary, that $R \neq \emptyset$. Since no MCCore exists in G, there does not exist an induced subgraph of G such that every node in this subgraph meets the neighbor-core constraint. However, by Algorithm 2, all the remaining nodes in R must satisfy the neighbor-core constraint, which is a contradiction. Second, we show that Algorithm 2 correctly outputs the MCCore, if it exists. Obviously, every node in R meets the neighbor-core constraint. To prove the theorem, it remains to show that R satisfies the maximal constraint (see Definition 3). This can also be proven by contradiction. Suppose that there is an MCCore R' such that it strictly contains R, i.e., R is a subgraph of R' and $R \neq R'$. Since R' is an MCCore, every node in R' meets the neighbor-core constraint. Clearly, all nodes in R' cannot be deleted by Algorithm 2. Therefore, all the nodes in R' must be contained in the results obtained by Algorithm 2, i.e., R, which is a contradiction. □

Example 4. Consider the signed graph in Fig. 1a. Let $\alpha = 3$ and $k = 1$. Clearly, the maximal positive-edge $\lceil \alpha k \rceil$-core is the subgraph induced by $\{v_1, \ldots, v_7\}$. We can see that the nodes $\{v_1, \ldots, v_5\}$ satisfy the neighbor-core constraint, while the nodes $\{v_6, v_7\}$ violate this constraint. Thus, in lines 6-9, the algorithm pushes $\{v_6, v_7\}$ into the queue $\mathcal{Q}$. After deleting $\{v_6, v_7\}$ from $\mathcal{Q}$, the nodes $\{v_1, \ldots, v_5\}$ still meet the neighbor-core constraint. Thus, we have $\mathcal{Q} = \emptyset$ after deleting v_6, and v_7. Since $\mathcal{Q} = \emptyset$, the algorithm terminates and returns $\{v_1, \ldots, v_5\}$ as the MCCore as desired.

Below, we introduce a useful concept, called ego network, which will be used to analyze the time complexity of Algorithm 2.

Definition 4. (ego network) *Given a signed graph G and a node u, the* ego network *of u is a subgraph of G induced by u's positive neighbors, i.e., $N_u^+(G)$.*

Example 5. Consider the signed network in Fig. 1a. By Definition 4, the ego network of v_2 is the subgraph induced by its positive neighbors $\{v_1, v_4, v_5, v_7\}$ shown in Fig. 2a. Similarly, Fig. 2b depicts an ego network of v_5 which is a subgraph induced by $\{v_1, v_2, v_4, v_4, v_6, v_7\}$.

It should be noted that an ego network may contain negative edges (see Fig. 2b). Let $H_{\max}$ be the maximum ego network in G among all the nodes' ego networks. Based on $H_{\max}$, we analyze the time and space complexity of MCBasic in Theorem 2.

Theorem 2. *The time and space complexity of Algorithm 2 is $O(m|H_{\max}|)$ and $O(m+n)$ respectively.*

Proof. The time spent to compute the maximal positive-edge $\lceil \alpha k \rceil$-core is $O(m+n)$. In lines 6-19, the algorithm traverses each edge in R at most 2 times in the main loops (except for invoking Algorithm 1 to compute the $(\lceil \alpha k \rceil - 1)$-core). When the algorithm traverses an edge (u,v) (line 12), it may visit v's ego network to compute the $(\lceil \alpha k \rceil - 1)$-core, which takes $O(|H_{\max}|)$ time. Therefore, the total cost in lines 6-19 is bounded by $O(m|H_{\max}|)$. Since the algorithm only needs to maintain several linear-sized structures, the space usage of Algorithm 2 is $O(m+n)$. □

Note that in real-world signed graphs, the running time of Algorithm 2 could be much less than the worst-case time complexity shown in Theorem 2. This is because the size of most ego networks is much smaller than $|H_{\max}|$, due to the power-law degree distribution of real-world graphs. Moreover, Algorithm 2 makes use of the degree pruning rule (line 15) to further reduce the time costs. In our experiments, we will show that Algorithm 2 is very efficient in practice.

3.3 The MCNew Algorithm

To further improve the efficiency of MCBasic, we propose a novel algorithm, called MCNew, based on a dramatically different idea. The striking feature of MCNew is that its worst-case time complexity is $O(\sigma m)$, where σ is the arboricity of the signed graph G [17]. The arboricity is shown to be bounded by $O(\sqrt{m})$ [11], and it is typically much smaller than the worst-case bound in most real-world graphs [12].

Before devising the MCNew algorithm, we first introduce a new concept called ego triangle as follows.

Definition 5. (ego triangle) *For any node u, a triangle (u,v,w) in the signed graph G is called an* ego triangle *of u if and only if both (u,v) and (u,w) are positive edges.*

It is important to note that the ego triangle is defined for a specified node. The same triangle (u,v,w) may be an ego triangle for u, but it may not be an ego triangle for v and w. For example, in Fig. 1a, the triangle (v_1,v_2,v_3) is an ego triangle of v_1, because both (v_1,v_2) and (v_1,v_3) are positive edges. This triangle, however, is not an ego triangle of v_2 (or v_3), as (v_2,v_3) is a negative edge.

Based on Definition 5, we can obtain a useful result, as shown in Lemma 4.

Lemma 4. *For any positive edge (u,v) in a signed graph G, the degree of v in u's* ego network *is equal to the number of* ego triangles *of u containing (u,v).*

Proof. Recall that the ego network of u is a subgraph induced by $N_u^+(G)$. For any ego triangle of u containing (u,v), denoted by (u,v,w), there exists an edge (v,w) in u's ego network. This is because both (u,v) and (u,w) are positive edges, and thereby both v and w are contained in u's ego network by definition. As a result, the number of ego triangles containing (u,v) equals the number of neighbors of v in u's ego network. □

Let Δ_u^v be the degree of v in u's ego network. Notice that Δ_u^v is not necessarily equal to Δ_v^u. The following example illustrates the definition of Δ_u^v.

Example 6. Consider an edge (v_2,v_5) in Fig. 1a. We have $\Delta_{v_2}^{v_5}=3$, because v_5 has three neighbors in v_2's ego network as shown in Fig. 2a. On the other hand, we can see that there are three ego triangles of v_2 containing (v_2,v_5), including (v_2,v_1,v_5), (v_2,v_4,v_5), and (v_2,v_5,v_7). This result confirms that Δ_u^v equals the number of ego triangles of u including (u,v), as shown in Lemma 4. We can also determine that $\Delta_{v_5}^{v_2}=4$ because v_2 has four neighbors in v_5's ego network as illustrated in Fig. 2b. Clearly, $\Delta_{v_5}^{v_2} \neq \Delta_{v_2}^{v_5}$ in this example.

Recall that to compute the MCCore, it is crucial to determine whether a node's ego network involves an $(\lceil \alpha k \rceil - 1)$-core. The key step to calculating the $(\lceil \alpha k \rceil - 1)$-core in u's ego network is to compute the degree of each node in u's ego network. In terms of Lemma 4, we are capable of computing the degree of every node in u's ego network by counting the ego triangles of u. Specifically, for each positive edge (u,v), we can compute Δ_u^v by counting the ego triangles of u including (u,v). We are also able to calculate Δ_v^u by counting the ego triangles of v including (v,u). Consequently, for each positive edge in G, we can compute Δ_u^v and Δ_v^u following two various directions, respectively. Thus, in our computation, each undirected positive edge (u,v) can be transformed into two *directed* positive edges (u,v) and (v,u).

If $\Delta_u^v < \lceil \alpha k \rceil - 1$, we can safely remove v from u's ego network. As indicated in Lemma 4, removing v from u's ego network is equivalent to deleting a *directed* positive edge (u,v) in G. For instance, in Fig. 2a, removing a node v_1 in v_2's ego network is equivalent to deleting a *directed* edge (v_2,v_1), because the number of ego triangles of v_2 containing (v_2,v_1) is 0 after removing (v_2,v_1). The deletion of (u,v) may cause the other *directed* positive edges need to be removed. For example, consider the ego network of v_2 in Fig. 2a. Assume that $\alpha=3$ and $k=1$. After deleting (v_2,v_1), we have also to remove (v_2,v_4) (and (v_2,v_5)), because the number of ego triangles of v_2 containing (v_2,v_4) (and (v_2,v_5)) decreases to 1 which is smaller than $\lceil \alpha k \rceil - 1$. Moreover, delete a *directed* positive edge (u,v), which will decrease the positive degree of u by 1, denoted by d_u^+. If d_u^+ is smaller than $\lceil \alpha k \rceil$, u can be deleted from G because u's ego network cannot contain an $(\lceil \alpha k \rceil - 1)$-core. Note that the deletion of a node u can be implemented by removing a set of edges associated with u, thus the same edge-deletion method can be used to handle a node deletion. This edge-deletion procedure is iteratively performed until no edge (and also no node) can be removed. It can be shown that each remaining node must satisfy the neighbor-core constraint when the algorithm completes, and thus all remaining nodes are comprised in the MCCore. The MCNew algorithm is outlined in Algorithm 3.

Implementation Details. Algorithm 3 first calls Algorithm 1 to compute the maximal $\lceil \alpha k \rceil$-core $R=(V_R,E_R)$ in the positive-edge graph, because the maximal constrained $\lceil \alpha k \rceil$-core is contained in the maximal $\lceil \alpha k \rceil$-core (line 1). Then, the algorithm doubles the directions for each positive edge in

E_R, and maintains all *directed* positive edges in S^+ (lines 5-6). Subsequently, for each *directed* positive edge $(u, v) \in S^+$, the algorithm computes Δ_u^v by counting the ego triangles that contains (u, v) (lines 7-8). If $\Delta_u^v < \lceil \alpha k \rceil - 1$, the algorithm pushes the *directed* positive edge (u, v) into a queue $\mathcal{Q}$ (line 9). Then, the algorithm iteratively removes the unqualified *directed* positive edges from the queue $\mathcal{Q}$ (line 10-24). When deleting a *directed* positive edge (u, v) from S^+, the algorithm needs to update Δ_u^w for each $(u, w) \in S^+$ and $(v, w) \in E_R$ (lines 12-13). This is because the removal of (u, v) may break an ego triangle of u containing (u, w), and therefore the algorithm may need to update Δ_u^w in terms of Lemma 4. If the updated Δ_u^w is smaller than $\lceil \alpha k \rceil - 1$, the algorithm pushes it into $\mathcal{Q}$ for iterative edge deletion (line 14). If the positive degree of a node u is smaller than τ after deleting (u, v), the algorithm removes u from G, and applies a similar edge-deletion method to handle the node deletion case (lines 16-24). The algorithm terminates when no further edges can be deleted. Finally, the algorithm outputs a subgraph induced by all the remaining nodes (line 25). The following theorem shows the correctness of Algorithm 3.

Algorithm 3. MCNew (G, α, k)

Input: $G = (V, E)$, α, and k
Output: The node set of the maximal constrained $\lceil \alpha k \rceil$-core
1: $(flag, V_R) \leftarrow$ ICore $(G^+, \emptyset, \lceil \alpha k \rceil)$; /* compute the $\lfloor \alpha k \rfloor$-core in G^+ */
2: $R \leftarrow$ the subgraph induced by V_R; /* $R = (V_R, E_R)$ */
3: $\mathcal{Q} \leftarrow \emptyset$; $S^+ \leftarrow \emptyset$; $\tau \leftarrow \lceil \alpha k \rceil - 1$;
4: $d_u^+ \leftarrow |\{w | (u, w) \in E_R^+\}|$; /* d_u^+ is the positive degree of u in R */
5: **for** each $(u, v) \in E_R^+$ **do**
6: $\quad S^+ \leftarrow S^+ \cup \{(u, v), (v, u)\}$;
7: **for** each $(u, v) \in S^+$ **do**
8: $\quad \Delta_u^v \leftarrow |\{w | (u, w) \in E_R^+, (v, w) \in E_R\}|$;
9: $\quad$**if** $\Delta_u^v < \tau$ **then** $\mathcal{Q}.push((u, v))$;
10: **while** $\mathcal{Q} \neq \emptyset$ **do**
11: $\quad (u, v) \leftarrow \mathcal{Q}.pop()$; Remove (u, v) from S^+;
12: $\quad$**for** each w s.t. $(u, w) \in S^+$ and $(v, w) \in E_R$ **do**
13: $\quad\quad \Delta_u^w \leftarrow \Delta_u^w - 1$;
14: $\quad\quad$**if** $\Delta_u^w < \tau$ and $(u, w) \notin \mathcal{Q}$ **then** $\mathcal{Q}.push((u, w))$;
15: $\quad d_u^+ \leftarrow d_u^+ - 1$;
16: $\quad$**if** $d_u^+ \leq \tau$ **then**
17: $\quad\quad$**for** each w s.t. $(u, w) \in S^+$ **do**
18: $\quad\quad\quad$Remove (u, w) from S^+ and $\mathcal{Q}$;
19: $\quad\quad$**for** each w s.t. $(w, u) \in S^+$ **do**
20: $\quad\quad\quad$Remove (w, u) from S^+ and $\mathcal{Q}$; $d_w^+ \leftarrow d_w^+ - 1$;
21: $\quad\quad\quad$**for** each x s.t. $(w, x) \in S^+$ and $(u, x) \in E_R$ **do**
22: $\quad\quad\quad\quad \Delta_w^x \leftarrow \Delta_w^x - 1$;
23: $\quad\quad\quad\quad$**if** $\Delta_w^x < \tau$ and $(w, x) \notin \mathcal{Q}$ **then** $\mathcal{Q}.push((w, x))$;
24: $\quad\quad$Remove u from R;
25: **return** the subgraph induced by nodes in $E^+(R)$;

Theorem 3. *Algorithm 3 correctly calculates the maximal constrained $\lceil \alpha k \rceil$-core.*

Proof. Let $R = (V_R, E_R)$ be the subgraph output of Algorithm 3. First, we claim that every node in V_R meets the neighbor-core constraint. Clearly, after the algorithm terminates, each node u in V_R has at least $\lceil \alpha k \rceil$ positive neighbors ($d_u^+ \geq \lceil \alpha k \rceil$). When the algorithm completes, we have $\Delta_u^v \geq \lceil \alpha k \rceil - 1$ for each directed positive edge (u, v), indicating every node in u's ego network has a degree at least $\lceil \alpha k \rceil - 1$. As a consequence, the ego network of u contains an $(\lceil \alpha k \rceil - 1)$-core. Second, we show that R also satisfies the maximal constraint. Suppose, to the contrary, that there is a subgraph R' such that it contains R and also every node in R' meets the neighbor-core constraint. Let u be a node in $R' \setminus R$. By the neighbor-core constraint, $d_u^+ \geq \lceil \alpha k \rceil$, u has at least $\lceil \alpha k \rceil$ positive neighbors whose degrees in u's ego network are no smaller than $\lceil \alpha k \rceil - 1$ (i.e., $\Delta_u^v \geq \lceil \alpha k \rceil - 1$). As a result, such a node u cannot be deleted by Algorithm 3. Thus, u must be included in R, which is a contradiction. □

Example 7. Reconsider the signed graph in Fig. 1a. Let $\alpha = 3$ and $k = 1$. First, the algorithm obtains a maximal $\lceil \alpha k \rceil$-core which is the subgraph induced by $\{v_1, \ldots, v_7\}$. We can easily derive that $\Delta_{v_7}^{v_2} = 1$, $\Delta_{v_7}^{v_6} = 1$, $\Delta_{v_6}^{v_7} = 1$, $\Delta_{v_6}^{v_3} = 1$, $\Delta_{v_2}^{v_7} = 1$, and $\Delta_{v_3}^{v_6} = 1$. Thus, the algorithm pushes six directed positive edges into $\mathcal{Q}$. After deleting (v_7, v_2), $\Delta_{v_7}^{v_5}$ is updated by 1. Thus, (v_7, v_5) will be pushed into $\mathcal{Q}$. Since $d_{v_7}^+ < 3$ after deleting (v_7, v_2), the algorithm removes v_7 (lines 15-24). As a consequence, the edges (v_7, v_6), (v_7, v_5), (v_6, v_7), and (v_2, v_7) are removed from $\mathcal{Q}$ (lines 17-20). For node v_6, $d_{v_6}^+$ decreases to 2. In the next iteration, the algorithm pops (v_6, v_3) from $\mathcal{Q}$, and v_6 will be deleted as $d_{v_6}^+ < 3$. Finally, the algorithm will obtain the MCCore $\{v_1, \ldots, v_5\}$ as desired.

Complexity Analysis. The time and space complexity of Algorithm 3 is analyzed in the following theorem.

Theorem 4. *The time and space complexity of Algorithm 3 is $O(\sigma m)$ and $O(m + n)$ respectively, where σ denotes the arboricity of the signed graph G.*

Proof. First, in line 1, the algorithm takes at most $O(m)$ time to compute the maximal $\lceil \alpha k \rceil$-core R. Second, in lines 7-9, the algorithm has to enumerate all ego triangles for all nodes in R, which can be implemented in $O(\sigma m)$ time by using triangle enumeration algorithm [18]. Third, in lines 10-24, every *directed* positive edge (u, v) is pushed into $\mathcal{Q}$ at most once. Thus, at most $O(m)$ edges can be popped from $\mathcal{Q}$. When deleting an edge (u, v) from $\mathcal{Q}$, the algorithm may explore all common neighbors between u and v (line 12), which can be done in $O(\min\{d_u, d_v\})$ using a hashing structure. Since at most $O(m)$ edges can be deleted, the total cost in lines 10-24 is $O(\sum_{(u,v) \in E_R} \min\{d_u, d_v\})$. Note that this total cost includes the cost of deleting nodes in lines 16-24, because the node deletion in our algorithm is processed as a set of edge deletions. As a result, the total time overhead of our algorithm is $O(\sum_{(u,v) \in E_R} \min \{d_u, d_v\} + \sigma m)$, which can be bounded by $O(\sigma m)$. Since our algorithm only maintains several linear-sized data structure, the space complexity of the algorithm is $O(m + n)$. □

Remark. It is worth remarking that the MCCore model is fundamentally different from the k-truss model [19]. In the k-truss model, each edge is contained in at least $k - 2$ triangles. The MCCore model contains both positive and negative edges, and each positive edge has two *implicit* directions as shown in Algorithm 3. The k-truss model only has one type of edge, and it does not consider the direction of the edges. Owing to these differences, the MCCore computation algorithm is much more complicated than the k-truss computation algorithm. Algorithm 3 not only needs to delete the unqualified edges, but it also needs to delete nodes. The traditional k-truss computation algorithm [19] only needs to iteratively remove unpromising edges.

4 Enumerating All Maximal (α, k)-Cliques

Recall that the maximal (α, k)-clique enumeration problem is NP-hard. Thus, a polynomial-time algorithm does not exist to solve our problem unless P=NP. In this section, we propose a branch and bound algorithm, called MSCE, to compute all maximal (α, k)-cliques in large signed networks. The MSCE algorithm first invokes the MCNew algorithm to prune the unpromising nodes, and then performs an efficient branch and bound enumeration (BBE) procedure on the reduced signed graph to find all maximal (α, k)-cliques. Below, we detail the branch and bound enumeration (BBE) procedure.

The Key Idea of BBE. Let $\mathcal{C}$ be the set of all maximal connected component of MCCore obtained by Algorithm 3. For each maximal connected component $R \in \mathcal{C}$, we carry out the following BBE procedure. First, if R is not a valid (α, k)-clique, BBE picks a node u from R to divide the search space into two subspaces: 1) the subspace of including u, and 2) the subspace of excluding u. Then, BBE recursively performs the same procedure in these two subspaces. Obviously, any maximal (α, k)-clique must be contained in one of these subspaces. The BBE algorithm makes use of a pair (R, I) to represent a search space, in which R is the set of candidate nodes, and I denotes the set of included nodes. Initially, R is set to be a maximal connected component of MCCore, and $I = \emptyset$. In each recursion, BBE may select a node $v \in R$ to split the search space (R, I) into two subspaces $(R, I \cup \{u\})$ and $(R \setminus \{u\}, I)$. Note that a search space (R, I) comprises all the maximal (α, k)-cliques containing I.

Second, if R is an (α, k)-clique, BBE can terminate the search early, and then verifies whether R is a maximal (α, k)-clique. Note that for each (α, k)-clique C, we can apply the following approach to show whether it is a maximal (α, k)-clique. First, we compute the common neighbors of all nodes in C. Then, for each common neighbor v, we determine whether $C \cup \{v\}$ is a valid (α, k)-clique or not. If this the case, C is not a maximal (α, k)-clique, as it can be expanded by a node v. Otherwise, C is a maximal (α, k)-clique. Below, we propose several effective pruning techniques to further improve the efficiency of the BBE algorithm.

4.1 The Pruning Rules in BBE

The $\lceil \alpha k \rceil$-Core Pruning Rule. In the search subspace (R, I), let G_R be the subgraph induced by R, and G_R^+ be the positive-edge graph of G_R. Then, we compute the maximal $\lceil \alpha k \rceil$-core on G_R^+, denoted by C. If C contains all nodes in I, we are able to reduce the candidate nodes set R. In particular, we can set $R = C$, because all nodes in $R \setminus C$ can be pruned (see Lemma 1). Otherwise, we can prune the entire search space, because it cannot contain any maximal (α, k)-clique including all nodes in I. Similarly, we are also capable of using MCCore for pruning. However, in BBE, we only adopt $\lceil \alpha k \rceil$-core pruning. This is because the algorithm needs to perform the pruning rule in each recursion (each search subspace). Thus, we choose $\lceil \alpha k \rceil$-core pruning, as it is much more computationally efficient than MCCore pruning.

The Clique-Constraint Pruning Rule. Let u be the picked node in the search space (R, I). Consider the subspace of including u, i.e., $(R, I \cup \{u\})$. Clearly, $I \cup \{u\}$ must be a clique, because all the included nodes in an (α, k)-clique form a clique. Otherwise, u cannot be added into I. For each $v \in R \setminus \{I \cup \{u\}\}$, if v is not a common neighbor of the nodes in $I \cup \{u\}$, we can safely prune v. This is because, v cannot be involved in a maximal (α, k)-clique that contains $I \cup \{u\}$. Therefore, we can prune v in the search space $(R, I \cup \{u\})$. Using this pruning rule, we can further reduce the candidate nodes set R.

The Negative-Edge Constraint Pruning Rule. Except for the clique-constraint pruning, we are also able to leverage the negative-edge constraint to further prune the subspace of including u. Specifically, for each $v \in R \setminus \{I \cup \{u\}\}$, if every node in the subgraph induced by $\{I \cup \{u, v\}\}$ violates the negative-edge constraint, v can be pruned. The reason is as follows. If some of nodes in $\{I \cup \{u, v\}\}$ do not meet the negative-edge constraint, $\{I \cup \{u, v\}\}$ cannot be contained in any maximal (α, k)-clique. That is to say, v cannot be included in any maximal (α, k)-clique that already contains $\{I \cup \{u\}\}$. As a result, we can prune v in the subspace $(R, I \cup \{u\})$.

4.2 The MSCE Algorithm

The MSCE algorithm is detailed in Algorithm 4. In lines 1-5, MSCE first invokes MCNew to compute the MCCore (line 1). Then, for each maximal connected component, MSCE calls BBE to enumerate all maximal (α, k)-cliques (line 2-5). Lines 6-25 outlines the BBE procedure. The $\lceil \alpha k \rceil$-core pruning rule is implemented in lines 8-10. Specifically, the algorithm invokes Algorithm 1 with the fixed nodes set I to compute whether there is an $\lceil \alpha k \rceil$-core in the positive-edge graph G_R^+ containing I (line 9). If no such an $\lceil \alpha k \rceil$-core exists, the algorithm prunes the current search space in terms of the $\lceil \alpha k \rceil$-core pruning rule (line 10). Otherwise, if the resulting $\lceil \alpha k \rceil$-core is also an (α, k)-clique, the algorithm performs a maximal property testing to verify whether it is a maximal (α, k)-clique (lines 11-12 and lines 21-25), and terminates early (line 13). The recursion in the subspace of including u is implemented in lines 15-19, while line 20 describes the recursion performed in the subspace of excluding u. Note that both the clique-constraint and negative-edge constraint pruning rules are implemented in lines 16-18. Since Algorithm 4 explores all search subspaces, the correctness of our algorithm is easily guaranteed. Below, we analyze the time and space complexity of our algorithm.

Complexity Analysis. The worst-case time complexity of the MSCE algorithm is exponential, due to the NP-hardness of our problem. Clearly, the enumeration tree of the MSCE algorithm is a binary tree because the algorithm partitions the search space into two subspaces in each recursion. Let n' and m' be the number of nodes and edges in the MCCore C, respectively. There are at most $2^{n'}$ subspaces explored by

MSCE. In each search subspace (R, I), MSCE takes $O(|G_R|)$ time to compute the $\lceil \alpha k \rceil$-core (line 9 in Algorithm 4), which is dominated by $O(m')$. To compute the clique-constraint pruning and the negative-edge constraint pruning, the algorithm consumes $O(|R| + |I|)$ time, which is bounded by $O(n')$. To check the maximal property for an (α, k)-clique, MSCE takes at most $O(\sum_{u \in R} d_u(C))$ time (lines 21-25), which is bounded by $O(m')$. Therefore, the total cost of MSCE spent in each recursion is at most $O(m')$. As a result, the time complexity of MSCE is $O(2^{n'}(m'))$. Since the size of the MCCore is typically not very large and the proposed pruning rules are very effective, MSCE is tractable for handling large-scale signed graphs. In the experiments, we show that our algorithm is scalable to the signed graph with more than one million nodes and ten millions edges. For the space complexity, the algorithm uses at most $O(m')$ space in each recursion. Since our algorithm works in a depth-first search (DFS) manner, the total space overhead of MSCE is $O(m + n)$, which is linear with respect to (w.r.t.) the graph size.

Algorithm 4. MSCE (G, α, k)

Input: $G = (V, E)$, α, and k
Output: All maximal (α, k)-cliques
1: $\mathcal{R} \leftarrow \emptyset$; $V_R \leftarrow$ MCNew (G, α, k);
2: $\mathcal{C} \leftarrow$ the set of maximal connected components of the subgraph induced by V_R;
3: **for** each $C \in \mathcal{C}$ **do**
4:   BBE $(V_C, \emptyset, \alpha, k)$;
5: **return** $\mathcal{R}$;
6: **Procedure** BBE (R, I, α, k)
7: Let $G_R = (R, E_R)$ be the subgraph induced by R;
8: Let $G_R^+ = (R, E_R^+)$ be the positive-edge subgraph of G_R;
9: $(flag, R) \leftarrow$ ICore $(G_R^+, I, \lceil \alpha k \rceil)$;
10: **if** $flag = 0$ **then return**;
11: **if** R is an (α, k)-clique **then**
12:   **if** MaxTest (R, α, k)=1 **then** $\mathcal{R} \leftarrow \mathcal{R} \cup \{R\}$;
13:   **return**; /* early termination */
14: Pick a node u from $R \setminus I$;
15: $D \leftarrow \emptyset$; $I_u \leftarrow I \cup \{u\}$; /* include u */
16: **for** $v \in R \setminus I_u$ **do**
17:   **if** $(v \notin N_u(G_R))$ or $(\exists w \in I_u \cup \{v\}$ s.t. $d_w^-(I_u \cup \{v\}) > k)$ **then**
18:     $D \leftarrow D \cup \{v\}$;
19: BBE $(R \setminus D, I_u, \alpha, k)$
20: BBE $(R \setminus \{u\}, I, \alpha, k)$; /* exclude u */
21: **Procedure** MaxTest (R, α, k)
22: Let CN_R be the set of common neighbors of all nodes in R;
23: **for** each $v \in CN_R$ **do**
24:   **if** $d_w^-(R \cup \{v\}) \leq k$ for all $w \in R \cup \{v\}$ **then return** 0;
25: **return** 1;

Heuristic Node Selection Strategy. Recall that the MSCE algorithm needs to select a node to split the search space in each recursion (line 14). A naive method is to randomly select a node u from $R \setminus I$. However, such a method may be inefficient. This is because this naive approach may pick a node that has many neighbors which may degrade the performance of the clique-constraint pruning and the negative-edge constraint pruning (lines 16-18). To enhance the pruning performance, we propose a heuristic node selection strategy. Specifically, we choose the node u from $R \setminus I$ with the minimum positive degree, i.e., $u = \arg\min_{v \in R \setminus I} \{d_v^+(G_R)\}$. The rationale behind our approach is as follows. The node u with minimum positive degree results in many other nodes in $R \setminus I$ that are either negative neighbors or non-neighbor nodes of u. The negative neighbors are likely to be pruned by the negative-edge constraint pruning rule, and the non-neighbor nodes can be pruned by the clique-constraint pruning rule. In our experiments, we show that this heuristic node selection strategy significantly outperforms a random node selection strategy.

5 Finding the Maximum (α, k)-Clique

We can slightly modify Algorithm 4 to find the maximum (α, k)-clique. Specifically, in line 11, when obtaining an (α, k)-clique, we maintain the size of the largest (α, k)-clique C^* found so far, and then use the size of C^*, denoted by $\rho = |C^*|$, to prune the search space. After computing the $\lceil \alpha k \rceil$-core R (line 9), the algorithm can terminate early if $|R| < \rho$. This is because in this case, the results obtained in the current search subspace cannot contain an (α, k)-clique that is larger than ρ. Such a size-based pruning rule, however, may not be very effective, because the number of nodes in the $\lceil \alpha k \rceil$-core R is often larger than the size of C^*. In this section, we propose several more effective pruning rules to further speed up the algorithm for maximum (α, k)-clique search.

Similar to Algorithm 4, we refer to (R, I) as a search subspace of the maximum (α, k)-clique search problem, where R is the $\lceil \alpha k \rceil$-core computed in line 9 in Algorithm 4 and $I \subseteq R$ is a clique which will be expanded to an (α, k)-clique. Note that for each (α, k)-clique C identified in the search subspace (R, I), C must include I and also C is contained in R.

For any search subspace (R, I), the key step of our pruning rules is to derive some tight upper bounds for the size of the largest (α, k)-clique that is contained in R. If the upper bound of the largest (α, k)-clique contained in R is no larger than the size of the largest (α, k)-clique found so far (i.e., ρ), we can safely prune the current search subspace. Below, we propose three new upper bounds.

Color-Based Bound. Since the (α, k)-clique must satisfy the clique constraint, we can make use of the classic color-based approach to bound the size of the largest (α, k)-clique contained in R. Specifically, we assign a color to each node in the signed graph G using a degree-ordering based greedy coloring algorithm [20], [21] so that no two adjacent nodes have the same color. Clearly, the nodes in an (α, k)-clique must have different colors. As a result, the number of colors of the nodes in R, denoted by $\mathsf{color}(R)$, is an upper bound of the size of the maximum (α, k)-clique contained in R. Note that since $\mathsf{color}(R)$ is typically much smaller than $|R|$, the color-based pruning rule is more effective than the size-based pruning rule.

In Algorithm 4, we can use $|I| + \mathsf{color}(R \setminus I)$ as an upper bound for pruning. Specifically, we compute $|I| + \mathsf{color}(R \setminus I)$ after obtaining the $\lceil \alpha k \rceil$-core R in line 9 of Algorithm 4. If $|I| + \mathsf{color}(R \setminus I)$ is no larger than ρ, the algorithm can prune the current search subspace (R, I). Note that for an efficient implementation, we only invoke the degree-ordering based greedy coloring algorithm once, and compute $\mathsf{color}(R \setminus I)$ in each recursion based on the same coloring result. Since the degree-ordering based greedy coloring algorithm can be implemented in linear time w.r.t. the size of the signed graph [20] and computing $\mathsf{color}(R \setminus I)$ can be done in $O(|\mathsf{color}(R \setminus I)|)$ time, the color-based pruning rule is very efficient in practice.

Negative-Degree Based Bound. The color-based bound only considers the clique constraint which ignores the negative-edge constraint. Here we develop a tighter upper bound, termed as a negative-degree based bound, on the basis of both the clique and negative-edge constraints. Let $d_u^-(I) \triangleq |\{v|v \in I, and\ v \in N_u^-\}|$ be the negative degree of a node u w.r.t. the nodes set I. For any node $u \in I$, if it is contained in an (α, k)-clique, u cannot have $k - d_u^-(I)$ negative neighbors in $R \setminus I$ by the negative-edge constraint. Therefore, if I is contained in an (α, k)-clique, the total number of negative links from a node in I to a node in $R \setminus I$ must be no larger than $\mathsf{sum}(I) \triangleq k|I| - \sum_{u \in I}\{d_u^-(I)\}$.

To derive the upper bound, we first assign a color to each node in the signed graph G. Let t be the number of colors in the nodes set $R \setminus I$, i.e., $t = \mathsf{color}(R \setminus I)$. Clearly, we can classify the nodes in $R \setminus I$ into t color groups, denoted by $\mathcal{X}_1, \ldots, \mathcal{X}_t$, such that the nodes in a color group $\mathcal{X}_i$ ($1 \leq i \leq t$) have the same color. Let $v_i^* \triangleq \arg\min_{v \in \mathcal{X}_i}\{d_v^-(I)\}$ be the node in $\mathcal{X}_i$ that has the smallest negative degree w.r.t. I. Then, we sort the nodes $\{v_1^*, \ldots, v_t^*\}$ in a non-decreasing order based on $d_{v_i^*}^-(I)$. Suppose without loss of generality that $d_{v_1^*}^-(I) \leq d_{v_2^*}^-(I) \leq \cdots \leq d_{v_t^*}^-(I)$. Let $\mathsf{psum}(i) \triangleq \sum_{j=1}^{i} d_{v_j^*}^-(I)$ be the *prefix sum* of the negative-degree array $[d_{v_1^*}^-(I), d_{v_2^*}^-(I), \ldots, d_{v_t^*}^-(I)]$. Then, we define $\bar{c}$ as

$$\bar{c} \triangleq \begin{cases} \arg\max_i\{\mathsf{psum}(i) \leq \mathsf{sum}(I)\}, if\ \mathsf{psum}(t) > \mathsf{sum}(I) \\ t, \quad otherwise. \end{cases} \tag{1}$$

Based on Eq. (1), we have the following result.

Lemma 5. *For any search subspace (R, I), $|I| + \bar{c}$ is an upper bound of the size of the maximum (α, k)-clique in (R, I).*

Proof. We can prove this lemma by contradiction. Let C be an (α, k)-clique in the search subspace (R, I). Suppose to the contrary that $|C| > |I| + \bar{c}$. Then, since C contains I, we have $|C \setminus I| > \bar{c}$. Since C is a clique, the nodes in $C \setminus I$ must have different colors. As a result, we have $\sum_{v \in C \setminus I}\{d_u^-(I)\} \geq \mathsf{psum}(|C \setminus I|) > \mathsf{psum}(\bar{c})$ by definition. If $\mathsf{psum}(t) > \mathsf{sum}(I)$, we have $\sum_{v \in C \setminus I}\{d_u^-(I)\} > \mathsf{sum}(I)$ by Eq. (1), violating the negative-edge constraint. Hence, C is not a valid (α, k)-clique in this case, which is a contradiction. If $\mathsf{psum}(t) \leq \mathsf{sum}(I)$, we have $\bar{c} = t$. Since the size of the largest (α, k)-clique in (R, I) must be no larger than $|I| + t$, we have $|C| \leq |I| + \bar{c}$, which contradicts our assumption. □

Note that the negative-degree based bound is tighter than the color-based bound, because $\bar{c} \leq \mathsf{color}(R \setminus I)$. Let $R_e = \sum_{u \in R} d_u$. Then, we can easily show that $\bar{c}$ (Eq. (1)) can be computed in $O(R_e)$ time after obtaining the colors of the nodes. Hence, the negative-degree based bound can also be efficiently obtained.

A Novel Supply-Demand Bound. Here we develop a novel supply-demand upper bound which is tighter than the negative-degree based bound. Recall that for a node $v \in R \setminus I$, $d_v^-(I)$ is the number of negative neighbors of v in I. That is to say, v can *supply* at most $d_v^-(I)$ negative neighbors for the nodes in I. Likewise, for a node $u \in I$, $k - d_u^-(I)$ is the maximum number of negative neighbors in $R \setminus I$ that can link to u by the negative-degree constraint. In other words, the *demand* of negative neighbors of u is at most $k - d_u^-(I)$. For convenience, we define $\mathsf{sup}(v) \triangleq d_v^-(I)$ as the *supply* of negative neighbors of v for each $v \in R \setminus I$, and $\mathsf{dem}(u) \triangleq k - d_u^-(I)$ as the demand of negative neighbors of u for each $u \in I$. Based on the *supply-demand* relationship of the negative degrees, we derive a novel upper bound as follows.

Algorithm 5 details the pseudo-code for computing the *supply-demand* upper bound. First, we assume that there are t color groups for all nodes in $R \setminus I$, denoted by $\mathcal{X}_1, \ldots, \mathcal{X}_t$ (line 2). Similar to the negative-degree based bound, we let $v_i^* \triangleq \arg\min_{v \in \mathcal{X}_i}\{\mathsf{sup}(v)\}$, and then sort the nodes $\{v_1^*, \ldots, v_t^*\}$ in a non-decreasing order based on $\mathsf{sup}(v_i^*)$ (lines 3-4). Suppose without loss of generality that $\mathsf{sup}(v_1^*) \leq \mathsf{sup}(v_2^*) \leq \cdots \leq \mathsf{sup}(v_t^*)$. Let I_h be the set of top-h nodes in I with the largest $\mathsf{dem}(u)$ for $u \in I$. Then, we perform the following iterative *supply-demand* procedure to derive an upper bound (lines 5-9). Specifically, for each $i = 1, \ldots, t$, we iteratively use a node v_i^* to *supply* a negative neighbor for each node $u \in I_{\mathsf{sup}(v_i^*)}$ (i.e., adds a negative link from v_i^* to u for each $u \in I_{\mathsf{sup}(v_i^*)}$). After that, the demand of negative neighbors for each $u \in I_{\mathsf{sup}(v_i^*)}$ decreases by 1. To implement this procedure, we can first calculate $I_{\mathsf{sup}(v_i^*)}$ (line 6), and then decrease $\mathsf{dem}(u)$ by 1 for each $u \in I_{\mathsf{sup}(v_i^*)}$ (line 7). If there exists a node u such that $\mathsf{dem}(u) < 0$, the iterative *supply-demand* procedure can early terminate (line 8). Assume that the iterative *supply-demand* procedure completes at the $\hat{c} + 1$th iteration. Then, we have the following result.

Lemma 6. *For any search subspace (R, I), $|I| + \hat{c}$ is an upper bound of the size of the maximum (α, k)-clique in (R, I).*

Proof. We prove this lemma by contradiction. Let C be an (α, k)-clique in (R, I). Suppose that $|C| > |I| + \hat{c}$. Then, we have $|C \setminus I| > \hat{c}$, as C contains I. We sort the nodes in $C \setminus I$ in a non-decreasing order based on $\mathsf{sup}(v)$ (for $v \in C \setminus I$). Let $l = |C \setminus I| > \hat{c}$ and $C \setminus I = \{\tilde{v}_1, \ldots, \tilde{v}_l\}$ with $\mathsf{sup}(\tilde{v}_1) \leq \ldots, \leq \mathsf{sup}(\tilde{v}_l)$. Clearly, we have $\mathsf{sup}(\tilde{v}_i) \geq \mathsf{sup}(v_i^*)$ for each $i = 1, \ldots, t$ by definition. Since C is a (α, k)-clique, there exists a *supply-demand* relationship between the nodes in $C \setminus I$ and the nodes in I, such that each node $\tilde{v}_i \in C \setminus I$ must *supply* $\mathsf{sup}(\tilde{v}_i)$ negative neighbors for the nodes in I. Recall that in the ith iteration of Algorithm 5, we greedily supplies a negative neighbor for the top-($\mathsf{sup}(\tilde{v}_i)$) nodes with the largest $\mathsf{dem}(u)$. Based on such a greedy strategy, we can also establish a *supply-demand* relationship between $C \setminus I$ and I, such that each node $\tilde{v}_i \in C \setminus I$ *supplies* $\mathsf{sup}(\tilde{v}_i)$ negative neighbors for the nodes in I. This result indicates that the number of iteration of Algorithm 5 must be no less than $|C \setminus I|$. Thus, we have $|C \setminus I| \leq \hat{c}$, which is a contradiction. □

We analyze the time complexity of Algorithm 5 as follows. First, the algorithm takes $O(R_e)$ ($R_e = \sum_{u \in R} d_u$) to

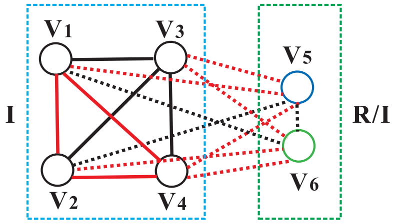


Fig. 3. Illustration of the tightness of the supply-demand bound.

TABLE 1
Datasets

Dataset	$n = \|V\|$	$m = \|E\|$	$\|E^+\|$	$\|E^-\|$	$k_{\max}$
Slashdot	82,144	500,481	382,882	117,599	54
Wiki	138,592	715,883	631,546	84,337	55
DBLP	1,314,050	5,362,414	1,245,522	4,116,892	118
Youtube	1,157,827	2,987,624	2,090,338	897,286	51
Pokec	1,632,803	30,622,564	21,355,492	9,267,072	47

compute $\mathsf{sup}(v)$ for each $v \in R \setminus I$ and $\mathsf{dem}(u)$ for each $u \in I$ after obtaining the colors of the nodes. Second, in each iteration, the algorithm takes $O(|I|)$ time to compute $I_{\mathsf{sup}(v_i^*)}$ using a linear-time integer sort algorithm (because $\mathsf{dem}(u)$ is an integer for each $u \in I$). Thus, the time complexity of Algorithm 5 is $O(R_e + \hat{c}|I|)$, except the time cost for coloring the nodes. Since both $\hat{c}$ and $|I|$ are often very small, the time cost for computing the supply-demand bound is comparable to the time cost for deriving the negative-degree based bound. In addition, it is easy to verify that $\hat{c} \leq \bar{c}$, indicating that the supply-demand bound is tighter than the negative-degree based bound. Moreover, $\hat{c}$ can be strictly smaller than $\bar{c}$ as illustrated in the following example.

Algorithm 5. sup-dem-bound (R, I)

Compute $\mathsf{sup}(v)$ for each $v \in R \setminus I$, and $\mathsf{dem}(u)$ for each $u \in I$;
Let $\mathcal{X}_1, \ldots, \mathcal{X}_t$ be the t color groups for all nodes in $R \setminus I$;
Compute $v_i^* \triangleq \arg\min_{v \in \mathcal{X}_i}\{\mathsf{sup}(v)\}$; $\hat{c} \leftarrow 0$;
$\{v_1^*, \ldots, v_t^*\} \leftarrow$ sort the t nodes v_i^* in a non-decreasing order by $\mathsf{sup}(v_i^*)$.
for $i = 1$ **to** t **do**
  Compute the top-($\mathsf{sup}(v_i^*)$) nodes set $I_{\mathsf{sup}(v_i^*)}$;
  $\mathsf{dem}(u) \leftarrow \mathsf{dem}(u) - 1$ for each $u \in I_{\mathsf{sup}(v_i^*)}$
  if there exists u such that $\mathsf{dem}(u) < 0$ **then** break;
  $\hat{c} \leftarrow \hat{c} + 1$;
return $|I| + \hat{c}$;

Example 8. Consider a search subspace (R, I) shown in Fig. 3, where $R = \{v_1, \ldots, v_6\}$ and $I = \{v_1, \ldots, v_4\}$. The black and red edges represent the positive and negative edges respectively. Suppose that $k = 3$. There are two color groups in $R \setminus I$ which contains v_5 and v_6 respectively. In this example, we can derive that $\mathsf{sup}(v_5) = 3$, $\mathsf{sup}(v_6) = 3$, $\mathsf{dem}(v_1) = 1$, $\mathsf{dem}(v_2) = 1$, $\mathsf{dem}(v_3) = 3$, and $\mathsf{dem}(v_4) = 1$. Clearly, we have $\mathsf{sum}(I) = 6$ and $\mathsf{psum}(2) = 6$, thus $\bar{c} = 2$ and the negative-degree based bound is $|I| + \bar{c} = 5$. However, by Algorithm 5, we can derive that $\hat{c} = 1$, thus the supply-demand bound is $|I| + \hat{c} = 4$, which is strictly tighter than the negative-degree based bound.

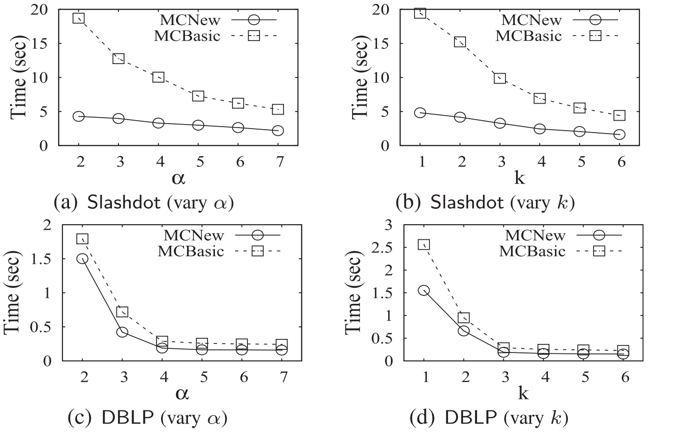
Finding the Top-r Maximal (α, k)-Cliques. The proposed upper bounds can also be applied to speed up the algorithm for finding the top-r maximal (α, k)-cliques. Specifically, in line 12 of Algorithm 4, when obtaining r maximal (α, k)-cliques, the algorithm maintains the minimum size over all r results. Suppose that the minimum size is ρ. Then, the algorithm computes the three proposed upper bounds in the search subspace (R, I). If one of an upper bound is smaller than ρ, the current search subspace (R, I) can be pruned. This is because in this case, the results obtained in the current search subspace cannot contain a maximal (α, k)-clique that is larger than the top-r results. The experimental results show that our algorithm is much faster at finding the top-r maximal (α, k)-cliques compared to enumerating all the results.

6 Experiments

In this section, we conduct extensive experiments to evaluate the efficiency and effectiveness of our algorithms. We implement two algorithms MCBasic and MCNew to compute maximal constrained $\lceil \alpha k \rceil$-cores. We also implement two algorithms MSCE-R and MSCE-G to compute all maximal (α, k)-cliques. MSCE-R is essentially Algorithm 4 with a random node-selection strategy, while MSCE-G is Algorithm 4 with a greedy node-selection strategy (see Section 4 for details). Since no existing algorithm can be applied to enumerate signed cliques, we use MSCE-R as the baseline for efficiency testing. We also implement two algorithms MSC *and* MSC+ for computing the maximum (α, k)-clique (or the top-r maximal (α, k)-cliques). MSC is the MSCE-G algorithm with the size-based pruning technique, while MSC+ is the MSCE-G algorithm with three upper bounding techniques developed in Section 5. Section 6.2 compares the effectiveness of our signed clique model with three other community models. All algorithms are implemented in C++. We conduct all experiments on a PC with a 2.4 GHz Xeon CPU and 16 GB memory running Red Hat Linux 6.4.

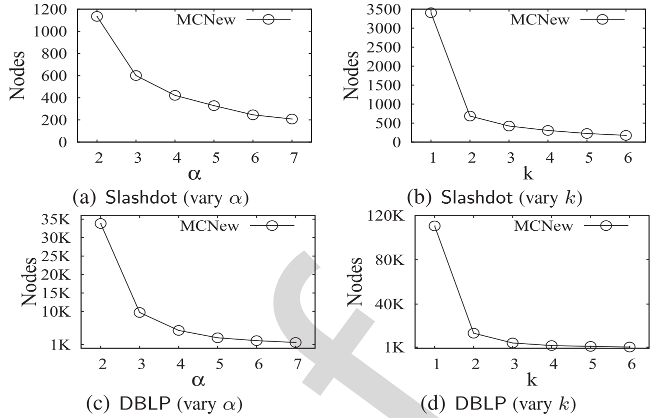
Datasets. We make use of five real-world datasets in our experiments. Table 1 provides the statistics, where the last column denotes the maximum k-core number of the network. Slashdot and Wiki are signed networks. DBLP is a co-authorship network, where each node denotes an author and an edge (u, v) means that u and v co-authored at least one paper. To create a signed network for DBLP, we assign "+" to an edge (u, v) if the number of papers co-authored by u and v is no less than the threshold τ, otherwise we assign "-" to (u, v). In all experiments, we set τ as the average number of papers co-authored by two researchers ($\tau = 1.427$ in our dataset). Both Youtube and Pokec are social networks. We generate a signed network for each by randomly picking 30 percent of the edges as the negative edges and the remaining edges as positive edges. Slashdot, DBLP, Youtube, and Pokec are downloaded from the Stanford network dataset collection (http://snap.stanford.edu). Wiki is downloaded from the Koblenz network collection (http://konect.uni-koblenz.de/).

Parameters. There are two parameters in our algorithms: α and k. The parameter α is selected from the interval $[2, 7]$ with a default value of $\alpha = 4$; k is chosen from the interval $[1, 6]$ with a default value of $k = 3$. Unless otherwise



(a) Slashdot (vary α) (b) Slashdot (vary k)
(c) DBLP (vary α) (d) DBLP (vary k)

Fig. 4. Efficiency of MCBasic and MCNew.



(a) Slashdot (vary α) (b) Slashdot (vary k)
(c) DBLP (vary α) (d) DBLP (vary k)

Fig. 5. The total number of nodes of maximal constrained $\lceil \alpha k \rceil$-cores.

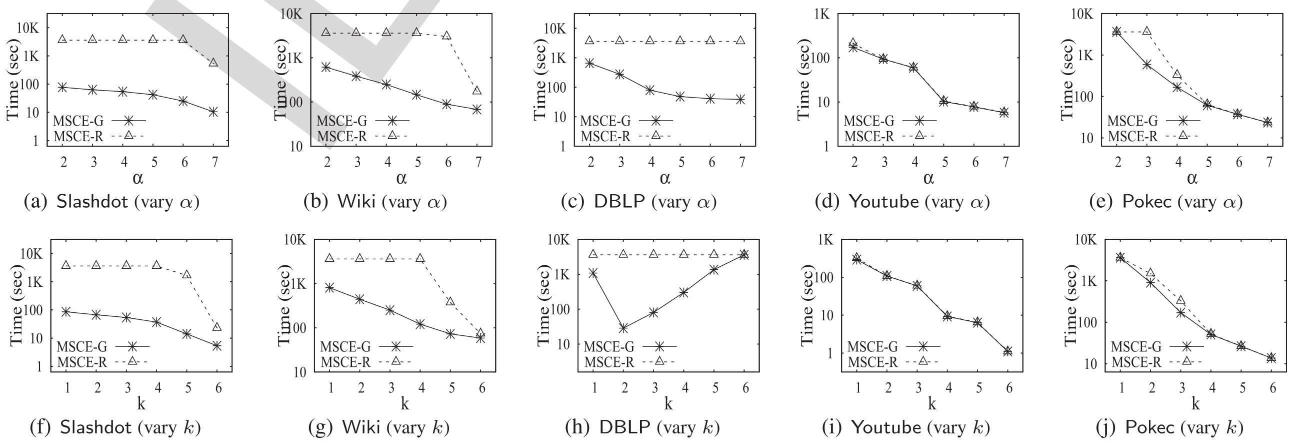
specified, the value of the other parameter is set to its default value when varying a parameter.

6.1 Efficiency Testing

Exp-1: Comparison Between MCBasic *and* MCNew. Fig. 4 shows the efficiency of MCBasic and MCNew on Slashdot and DBLP datasets. Similar results can also be observed for the other datasets. Both MCBasic and MCNew are very efficient. MCNew consistently outperforms MCBasic with all parameter settings. For example, on Slashdot, MCNew is four times faster than MCBasic when $\alpha = 2$ and $k = 3$. In general, the running time of both MCBasic and MCNew decrease with an increasing α and k. This is because the neighbor-core constraint of the maximal constrained $\lceil \alpha k \rceil$-core (Definition 3) grows stronger when α and k are large, which gives rise to strong pruning performance in both MCBasic and MCNew. It is worth noting that our algorithms are fairly fast in DBLP because the positive-edge network in DBLP is very sparse. These results confirm our theoretical analysis in Section 3.

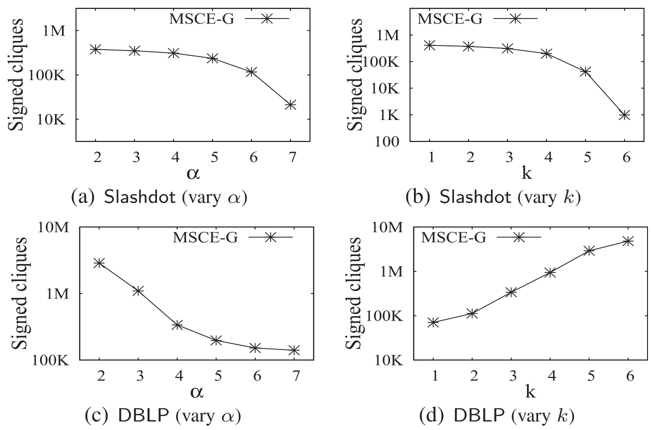
Exp-2: The Size of Maximal Constrained $\lceil \alpha k \rceil$-Cores. In this experiment, we study the total number of nodes of the maximal constrained $\lceil \alpha k \rceil$-cores. To this end, we use MCNew to compute the maximal constrained $\lceil \alpha k \rceil$-cores, as it is more efficient than MCBasic. Fig. 5 shows the results for the Slashdot and DBLP datasets. Similar results can be also observed for the other datasets. As desired, the number of nodes of the maximal constrained $\lceil \alpha k \rceil$-cores decreases with an increasing α and k. Moreover, we observe that the total number of nodes of the maximal constrained $\lceil \alpha k \rceil$-cores is much smaller than the number of nodes of the graph. For instance, in Fig. 5a, when $\alpha = 4$ and $k = 3$, the total number of nodes of maximal constrained $\lceil \alpha k \rceil$-cores is only 422, but the entire graph size is 82,144. These results indicate that the proposed graph reduction technique can drastically prune unpromising nodes to identify the signed cliques.

Exp-3: Results of Enumerating all Signed Cliques. In this experiment, we study the efficiency of MSCE-R and MSCE-G for enumerating all signed cliques. We limit the maximal running time to 3,600 seconds for both MSCE-R and MSCE-G, because MSCE-R may be intractable with some parameter settings due to the NP-hardness of our problem. Fig. 6 reports the efficiency of these algorithms with varying values for α and k. From Fig. 6, we can see that MSCE-G is at least one order of magnitude faster than MSCE-R on the Slashdot, Wiki, and DBLP datasets with most parameter settings. For example, when $\alpha = 4$ and $k = 3$, MSCE-G takes 54 seconds to enumerate all signed cliques on Slashdot, while MSCE-R does not terminate within 3,600 seconds. On Youtube and Pokec, MSCE-G consistently outperforms MSCE-R. We can also clearly observe that MSCE-G is tractable on all datasets with almost all parameter settings. MSCE-R, however, is only tractable on the Youtube dataset. These results confirm that the greedy



(a) Slashdot (vary α) (b) Wiki (vary α) (c) DBLP (vary α) (d) Youtube (vary α) (e) Pokec (vary α)
(f) Slashdot (vary k) (g) Wiki (vary k) (h) DBLP (vary k) (i) Youtube (vary k) (j) Pokec (vary k)

Fig. 6. Efficiency of our algorithms for enumerating all signed cliques.



(a) Slashdot (vary α) (b) Slashdot (vary k)
(c) DBLP (vary α) (d) DBLP (vary k)

Fig. 7. The number of maximal (α, k)-cliques.

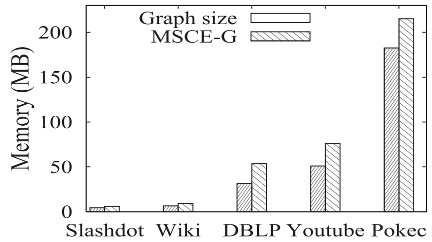
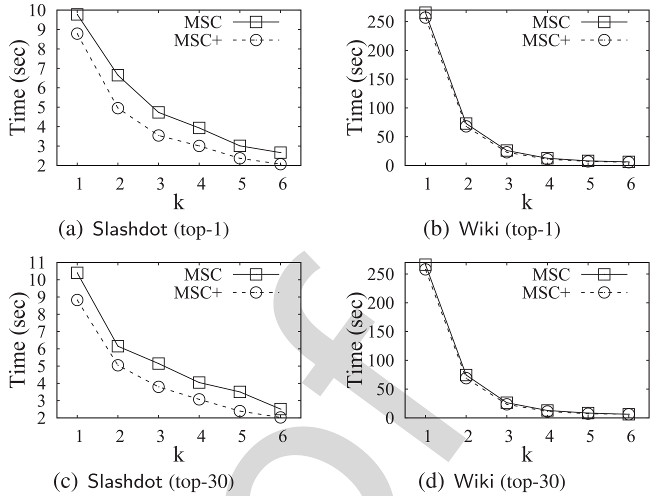


Fig. 8. Memory overhead of MSCE-G.

node-selection strategy in Algorithm 4 is significantly better than the random node-selection strategy.

Generally, the running time of our algorithms drops with an increasing α and k. This is because the positive-edge constraint of maximal (α, k)-clique is strong for large values of α and k, thus enhancing the pruning power of our algorithms. Interestingly, in some cases, the running time of MSCE-G does not necessarily decrease when k increases. For example, in Fig. 6h, when $k \geq 2$, MSCE-G's running time increases as k increases on DBLP. This could be because MSCE-G's pruning power may be dominated by negative-edge pruning when $k \geq 2$. Since (i) the negative-edge constraint of maximal (α, k)-clique is relatively weak for a large k and (ii) DBLP has a relatively large $k_{\max}$ value (see Table 1), the number of signed cliques can be very large. Therefore, in this case, the pruning power of MSCE-G decreases when k increases. However, on the other datasets, the $k_{\max}$ values are relatively small and the pruning power of our algorithm may be dominated by the positive-edge constraint, thus the running time of MSCE-G decreases as k increases.

Exp-4: The Number of Maximal (α, k)-Cliques. Fig. 7 shows the number of maximal (α, k)-cliques on the Slashdot and DBLP datasets. Similar results can also be derived on the other datasets. On Slashdot, the number of signed cliques decreases as both α and k increases, because the positive-edge constraint (see Definition 1) is strong if k is large. On DBLP, however, the number of signed cliques increases with an increasing k. The reason could be that on DBLP, the negative-edge constraint of the maximal (α, k)-clique may dominate its positive-edge constraint. With a large k, the negative-edge constraint is relatively weak. Thus, the number of signed cliques increases with increasing k. These results are consistent with the results observed in Exp-3.



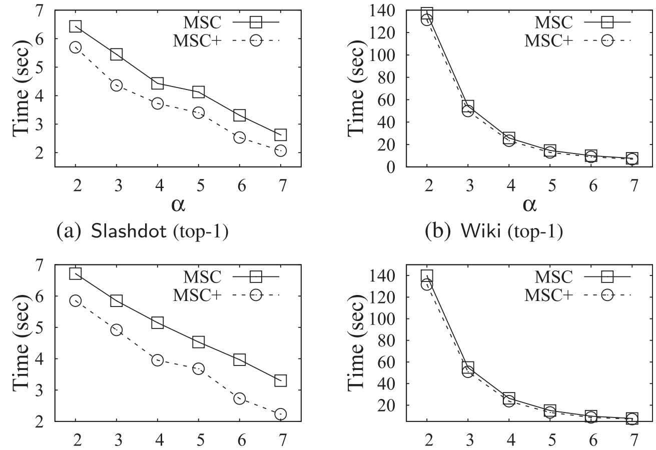
(a) Slashdot (top-1) (b) Wiki (top-1)
(c) Slashdot (top-30) (d) Wiki (top-30)

Fig. 9. Comparison between MSC and MSC+ with varying k.

Exp-5: Memory Overhead. Fig. 8 reports the memory overhead of MSCE-G for all datasets. The results demonstrate that the memory usage of MSCE-G is slightly higher than the graph size but clearly lower than twice the size of the graph. These results confirm the linear space complexity of MSCE-G.

Exp-6: Comparison Between MSC *and* MSC+. Here we evaluate the efficiency of MSC and MSC+ for finding the maximum (α, k)-clique and identifying the top-r maximal (α, k)-cliques, respectively. In this experiment, r is selected from an interval $[1, 50]$. When varying α and k, r is set to a default value 30. Fig. 9 shows the efficiency of MSC and MSC+ with varying k on the Slashdot and Wiki datasets. Similar results can also be observed on the other datasets. As can be seen, MSC+ consistently outperforms MSC with varying k on both Slashdot and Wiki, due to the effective upper bounds developed in Section 5. For example, MSC+ only takes 3.5 seconds to identify the maximum (α, k)-clique on Slashdot given that $k = 3$, while MSC consumes near 5 seconds under the same parameter setting. We can also observe that the upper bounds used in MSC+ do not offer significant benefits on the Wiki dataset. This is because Wiki contains few negative edges, resulting in that both the negative-degree based bound and the supply-demand bound are not very effective. As desired, the running time of both MSC and MSC+ decreases as k increases, because the negative-edge constraint is strong for a large k. Fig. 10 depicts the results with varying α on Slashdot and Wiki. The results on the other datasets are consistent. Similarly, we can see that MSC+ is significantly faster than MSC on Slashdot, and it slightly outperforms MSC on Wiki when varying α. These results indicate that our newly-developed upper bounds are indeed more effective than the size-based upper bound for pruning unpromising search subspaces in finding the maximum (α, k)-clique or the top-r maximal (α, k)-cliques (especially for signed graphs with many negative edges).

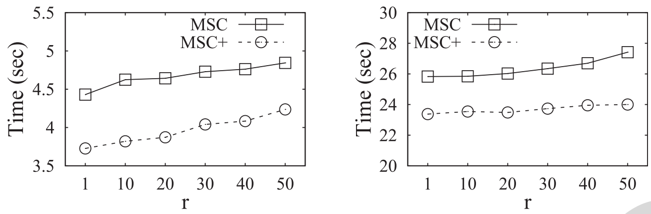
We also evaluate the efficiency of MSC and MSC+ with varying r. The results on Slashdot and Wiki are reported in Fig. 11. As desired, the running time of both MSC and MSC+ increases as r increases. We can also see that MSC+ consistently outperforms MSC for all r on both Slashdot and



(a) Slashdot (top-1) (b) Wiki (top-1)

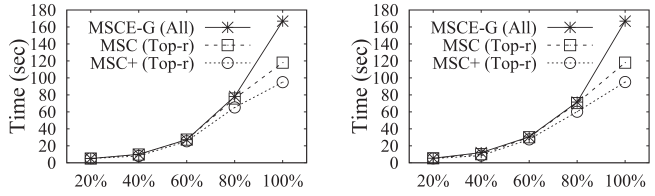
(c) Slashdot (top-30) (d) Wiki (top-30)

Fig. 10. Comparison between MSC and MSC+ with varying α.



(a) Slashdot ($\alpha = 4, k = 3$) (b) Wiki ($\alpha = 4, k = 3$)

Fig. 11. Comparison between MSC and MSC+ with varying r.



(a) Vary $|V|$ (b) Vary $|E|$

Fig. 12. Scalability testing (Pokec, $\alpha = 4$, $k = 3$, $r = 30$).

Wiki. Compared to MSCE-G for enumerating all maximal (α, k)-cliques (Fig. 6), both MSC and MSC+ take substantially less time to compute top-r signed cliques. For example, when $\alpha = 4$ and $k = 3$, MSC and MSC+ takes 4.7 and 4 seconds on Slashdot to find the top-30 results, respectively. However, it takes 54 seconds to enumerate all the signed cliques on Slashdot. These results demonstrate the high efficiency of our algorithms for identifying the top-r results.

Exp-7: Scalability Testing. We make use of the largest dataset Pokec to test the scalability of MSCE-G, MSC , and MSC+. Specifically, we generate four subgraphs by randomly sampling 20-80 percent of the edges from Pokec and test the time costs of our algorithms on these subgraphs. Fig. 12 depicts the scalability results using the default parameter setting. The time costs of all algorithms increase smoothly with a varying $|V|$ or $|E|$ in both tests. We also find that both MSC and MSC+ show near-linear scalability in identifying the top-r results. These results suggest that our algorithms are scalable when handling large real-world signed networks.

6.2 Effectiveness Testing

To measure the quality of a cohesive subgraph in signed networks, we propose an intuitive metric called signed conductance, based on the classic conductance in graph theory [13]. Let S be a set of nodes. The signed conductance of S is defined below:

$$\phi(S) \triangleq \frac{\sum_{u\in S, v\in V\setminus S} A^+_{uv}}{\min\{\sum_{u\in S} d^+_u, \sum_{u\in V\setminus S} d^+_u\}} - \frac{\sum_{u\in S, v\in V\setminus S} A^-_{uv}}{\min\{\sum_{u\in S} d^-_u, \sum_{u\in V\setminus S} d^-_u\}}. \tag{2}$$

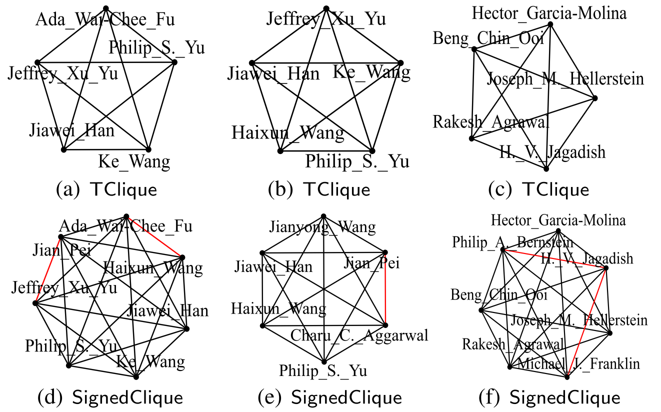
The first (second) part in Eq. (2) is the classic conductance of S [13] defined on the signed network without considering negative (positive) edges. For convenience, we refer to the first (second) part as the positive-edge conductance (negative-edge conductance). Intuitively, an *interesting* cohesive subgraph (e.g., a trust community) in a signed network should have many positive intra-edges and few negative intra-edges. It should also have many negative inter-edges and few positive inter-edges. In other words, an *interesting* cohesive subgraph in the signed network should have a low positive-edge conductance and a high negative-edge conductance. Clearly, the definition of signed conductance in Eq. (2) captures this intuition. Note that the signed conductance $\phi(S)$ falls into a range $[-1, 1]$. An *interesting* cohesive subgraph in a signed network should has a small signed conductance.

We compare our signed clique model, denoted by SignedClique, with three intuitive baselines: Core [9], SignedCore [5], and TClique [22]. Core is a method that computes the $\lceil \alpha k \rceil$-core in the signed network after removing all negative edges. SignedCore is an existing signed community model proposed in [5], which has been successfully applied to analyze trust dynamics in signed networks. SignedCore, as defined in [5], has two parameters β and γ. It requires that every node in the SignedCore has at least β positive neighbors and also has at least γ negative neighbors. Thus, to match the parameters between SignedCore and SignedClique, we set $\beta = \lceil \alpha k \rceil$ and $\gamma = k$ in our experiments. TClique is the state-of-the-art signed community model proposed in [22] which aims to identify maximal cliques in the signed network without considering negative edges. In [22], the TClique model is considered to be a trusted clique, and its size is limited to k. For a fair comparison, we drop this size constraint in TClique with the aim of finding all maximal *trusted* cliques.

Exp-8: Signed Conductance of Various Models. We compute the average signed conductance of the top-r communities returned by each method. Table 2 reports the results obtained with the default parameter settings (i.e., $\alpha = 4$, $k = 3$, and $r = 30$). Similar results can also be obtained with

TABLE 2
Signed Conductance of Various Models

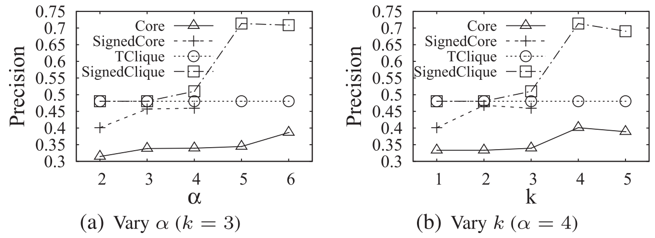
Datasets	Core	SignedCore	TClique	SignedClique
Slashdot	−0.0252	−0.0764	−0.0838	**−0.0863**
Wiki	0.0835	0.0252	−0.0124	**−0.0218**
DBLP	−0.4485	−0.4946	−0.4856	**−0.5154**
Youtube	−0.0201	−0.0158	−0.0233	**−0.0237**
Pokec	−0.0235	−0.0149	−0.1345	**−0.2262**



(a) TClique (b) TClique (c) TClique

(d) SignedClique (e) SignedClique (f) SignedClique

Fig. 13. Comparison of various models ($\alpha = 2$ and $k = 2$, black edges are positive edges and red edges denote negative edges).



(a) Vary α ($k = 3$) (b) Vary k ($\alpha = 4$)

Fig. 14. Precision of different community models (FlySign).

other parameter settings. From Table 2, we can see that SignedClique consistently outperforms all the baselines. The results for SignedCore and TClique are comparable, with both performing slightly better than Core. The reasons are as follows. Compared to other models, SignedClique not only requires every node that has $\lceil \alpha k \rceil$ positive intra-neighbors, but it also limits the number of negative intra-neighbors to be smaller than k. Therefore, there may be many positive edges in the community, with a few positive edges that can span different communities, resulting in a small positive-edge conductance. On the other hand, there are not too many negative edges in our community (due to the negative-edge constraint). Hence, there may be many negative edges spanning different communities, which gives rise to a large negative-edge conductance. As a consequence, the signed conductance of our model should be small. These results indicate that the proposed approach is indeed effective for modeling cohesive subgraphs in signed networks.

Exp-9: Case Study on DBLP. We conduct a case study using the DBLP dataset to compare the effectiveness of various models. Recall that, in DBLP, a negative (positive) edge implies that two researchers have co-authored at least τ papers, where $\tau = 1.427$ is the average number of papers co-authored by the researchers. A negative (positive) edge in DBLP can be considered to be a weak (strong) connection between two authors. Fig. 13 shows the communities of Professors Jiawei Han and H. V. Jagadish derived by TClique and SignedClique with the parameters $\alpha = 2$ and $k = 2$. Note that we test both Core and SignedCore using many parameter settings, but the community size (including Jiawei Han or H. V. Jagadish) is either very large (more than 10,000 nodes), or no community is found, so those results have not been included. The reason could be that the k-core constraint in both Core and SignedCore is relatively loose; therefore, these models fail to discover compact communities. As shown in Fig. 13, our model is able to find strongly-cooperative and compact communities with a tolerance to a few negative edges, whereas the TClique model may miss some important members of the community. For example, in Figs. 13a and 13b, TClique misses Professors Pei Jian and Charu C. Aggarwal. However, with a few negative edges, the communities in Figs. 13d and 13e obtained by SignedClique consist of Professors Pei Jian and Charu C. Aggarwal. Similar results can also be observed in Figs. 13c and 13f. These results indicate that our model is more effective than the baselines in identifying intuitive and compact communities in signed networks.

Exp-10: Protein Complex Discovery. In signed protein-protein interaction networks, a protein complex typically denotes a densely-connected signed subgraph [3]. In this experiment, we compare the effectiveness of SignedClique with those of the other baseline models for protein complex discovery. We collect a real-world signed PPI network, called FlySign, from [23]. The FlySign network consists of 3,352 nodes and 6,094 signed edges (4,112 positive edges and 1982 negative edges). The ground-truth complexes in FlySign can be obtained by using the complex enrichment analysis tool [3], [24]. Based on the ground-truth complexes, we are able to compute the precision for different models. Specifically, for each complex obtained by various models, the precision is computed by TP/(TP+FP), where TP denotes the number of true-positive nodes and FP denotes the number of false-positive nodes. We compute the average precision of the top-30 complexes identified by different models. The results are shown in Fig. 14. We can see that SignedClique significantly outperforms the other baselines under all parameter settings. In general, the clique-based models (SignedClique and TClique) perform much better than the core-based models (SignedCore and Core). The reason could be that the results of the clique-based models are much more compact than those of the core-based models. For example, when $\alpha = 5$ and $k = 3$, the precision of SignedClique and TClique is 0.71 and 0.48 respectively, while the precision of SignedCore and Core is 0 and 0.34 respectively. Note that SignedCore may return an empty subgraph when k is large, because the SignedCore model imposes a strong negative-edge constraint which requires the number of negative edges no less than k [5]. As a result, the precision of SignedCore can be 0 when k is large. These results further confirm the effectiveness of SignedClique.

Exp-11: Structural Balance of SignedClique. We analyze the communities obtained by SignedClique using the structural balance theory in signed network [6]. Specifically, by the strong structural balance theory [6], the triangles in a signed network are classified into balanced triangles and unbalanced triangles. The balanced triangles contain 1 or 3 positive edges, while the unbalance triangles consist of 0 or 2 positive edges. The weak structural balance theory further assumes that only triangles with 2 positive edges are called unbalanced triangles, and all other types of triangles are balanced triangles. The structural balance theory indicates that a *good* community in a signed network should include many balanced triangles and a few unbalanced triangles. Fig. 15 shows the average ratio of balanced triangles contained in the top-30 communities of SignedClique using the

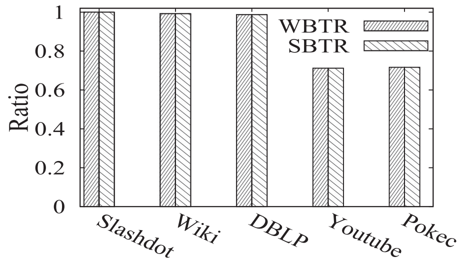


Fig. 15. The ratio of balanced triangles of SignedClique. WBTR (SBTR) denotes the ratio of weak (strong) structurally balanced triangles.

default parameters $\alpha = 4$ and $k = 3$. As can be seen, the ratio of balanced triangles obtained by SignedClique is no less than 0.7 on all datasets. More importantly, on the real-world signed networks Slashdot and Wiki, the ratios are even close to 1. These results indicate that SignedClique can be used to detect structurally balanced communities in signed networks.

7 Related Work

Community Modeling. Communities in a graph are often represented by densely-connected subgraphs. Many community models exist in the literature. Notable examples include the maximal clique model [8], [14], k-core [9], [25], k-truss [2], [10], [19], maximal k-edge connected subgraph [26], [27], quasi-clique [28], locally densest subgraph [29], and so on. More recently, many different community models have been proposed for attributed graphs. For example, Fang et al. [30] proposed an attributed community model based on k-core. Huang and Lakshmanan [31] presented an attributed truss model to find the community with highest attribute relevance score w.r.t. query nodes. Beyond attributed communities, Li et al. [32] introduced an influential community model to capture the influence of a community. More recently, Li et al. [33] proposed a skyline community model for seeking communities in multi-valued networks. The same group also proposed a persistent community model to detect persistent communities in temporal graphs [34]. All the above-mentioned community models are tailored to unsigned networks. To define a cohesive subgraph model in signed networks, Giatsidis et al. [5] introduced a signed core model, which was originally proposed to study the trust dynamic in signed networks. However, this model is not able to intuitively reveal a community in a signed network because it requires the number of negative edges to be larger than a given threshold, which may result in the nodes in the community having many negative neighbors. Hao et al. [22] proposed a trusted clique model, which completely ignores the negative edges in the signed network. Unlike previous models, the proposed signed clique model limits the number of negative neighbors for each node in the community. Thus, it is better to reflect a community in signed networks, as confirmed in our experiments. [35] contains a short version of this work in which we focus mainly on enumerating all maximal (α, k)-cliques. In this work, we also investigate the problem of finding the maximum (α, k)-clique, and develop an efficient algorithm to identify the maximum (α, k)-clique based on three carefully-designed upper bounds.

Signed Network Analysis. After a seminal work [6], signed network analysis has attracted much attention in recent years. Notable applications include link prediction [36], [37], [38], recommendation systems [39], [40], clustering and community detection [3], [7], [41], [42], and antagonistic community analysis [43], [44]. An excellent survey on signed network analysis can be found in [45]. Our work is closely related to clustering and community detection. The aim in solving this problem is to partition the signed network into several densely-connected components [7], [41]. Most existing solutions involve a complicated optimization procedure (e.g., [3], [42]), and therefore they cannot handle million-sized signed networks. Moreover, they also lack a clear and cohesive subgraph model to characterize the resulting communities. Unlike these studies, our work provides a cohesive subgraph model that could be useful for community discovery and search related applications in signed networks [1]. Further, the proposed algorithm is scalable to million-sized signed networks.

8 Conclusion

In this paper, we introduce a novel model, called maximal (α, k)-clique, to characterize a cohesive subgraph in signed networks. To enumerate all maximal (α, k)-cliques, we first propose an efficient signed network reduction algorithm to substantially prune the signed network. The time complexity of our technique is $O(\delta m)$, where δ denotes the arboricity of the signed network. Then, we develop a new branch and bound enumeration algorithm with several powerful pruning techniques to efficiently enumerate all maximal (α, k)-cliques. We also devise an efficient maximum (α, k)-clique search algorithm with three novel upper-bounding techniques. Comprehensive experiments on five large real-life networks demonstrate the efficiency, scalability, and effectiveness of our algorithms.

Acknowledgments

This work was partially supported by (i) NSFC Grants 61772346, 61732003, U1809206, 61772091, 61802035; (ii) National Key R&D Program of China 2018YFB1004402; (iii) Beijing Institute of Technology Research Fund Program for Young Scholars; (iv) Research Grants Council of the Hong Kong SAR, China No. 14221716 and 14203618; (v) ARC Discovery Project Grant DP160101513; and (vi) MOE, Singapore under grant MOE2015-T2-2-069 and NUS, Singapore under an SUG.

References

[1] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 939–948.
[2] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1311–1322.
[3] L. Ou-Yang, D. Dai, and X. Zhang, "Detecting protein complexes from signed protein-protein interaction networks," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 12, no. 6, pp. 1333–1344, Nov./Dec. 2015.
[4] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proc. Int. Conf. Very Large Data Bases*, 2005, pp. 721–732.

[5] C. Giatsidis, B. Cautis, S. Maniu, D. M. Thilikos, and M. Vazirgiannis, "Quantifying trust dynamics in signed graphs, the S-cores approach," in *Proc. SIAM Int. Conf. Data Mining*, 2014, pp. 668–676.

[6] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg, "Signed networks in social media," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2010, pp. 1361–1370.

[7] B. Yang, W. K. Cheung, and J. Liu, "Community mining from signed social networks," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 10, pp. 1333–1348, Oct. 2007.

[8] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Trans. Database Syst.*, vol. 36, no. 4, pp. 21:1–21:34, 2011.

[9] S. B. Seidman, "Network structure and minimum degree," *Social Netw.*, vol. 5, no. 3, pp. 269–287, 1983.

[10] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," Fort Meade, MD, USA, National Security Agency, Tech. Rep., 2005.

[11] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM J. Comput.*, vol. 14, no. 1, pp. 210–223, 1985.

[12] M. C. Lin, F. J. Soulignac, and J. L. Szwarcfiter, "Arboricity, h-index, and dynamic algorithms," *Theoretical Comput. Sci.*, vol. 426, pp. 75–90, 2012.

[13] S. Galhotra, A. Bagchi, S. Bedathur, M. Ramanath, and V. Jain, "Tracking the conductance of rapidly evolving topic-subgraphs," *Proc. VLDB Endowment*, vol. 8, no. 13, pp. 2170–2181, 2015.

[14] C. Bron and J. Kerbosch, "Finding all cliques of an undirected graph (algorithm 457)," *Commun. ACM*, vol. 16, no. 9, pp. 575–576, 1973.

[15] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Comput. Sci.*, vol. 363, no. 1, pp. 28–42, 2006.

[16] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *ACM J. Exp. Algorithmics*, vol. 18, 2013, Art. no. 3.1.

[17] C. S. J. A. Nash-Williams, "Decomposition of finite graphs into forests," *J. London Math. Soc.*, vol. 39, no. 1, pp. 12–12, 1964.

[18] X. Hu, Y. Tao, and C. Chung, "I/O-Efficient algorithms on triangle listing and counting," *ACM Trans. Database Syst.*, vol. 39, no. 4, pp. 27:1–27:30, 2014.

[19] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proc. VLDB Endowment*, vol. 5, no. 9, pp. 812–823, 2012.

[20] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Ordering heuristics for parallel graph coloring," in *Proc. Annu. ACM Symp. Parallelism Algorithms Archit.*, 2014, pp. 166–177.

[21] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Effective and efficient dynamic graph coloring," *Proc. VLDB Endowment*, vol. 11, no. 3, pp. 338–351, 2017.

[22] F. Hao, S. S. Yau, G. Min, and L. T. Yang, "Detecting k-balanced trusted cliques in signed social networks," *IEEE Internet Comput.*, vol. 18, no. 2, pp. 24–31, Mar./Apr. 2014.

[23] A. Vinayagam, J. Zirin, C. Roesel, Y. Hu, B. Yilmazel, A. A. Samsonova, R. A. Neumuller, S. E. Mohr, and N. Perrimon, "Integrating protein-protein interaction networks with phenotypes reveals signs of interactions," *Nature Methods*, vol. 11, pp. 94–99, 2014.

[24] A. Vinayagam, Y. Hu, M. Kulkarni, C. Roesel, R. Sopko, S. E. Mohr, and N. Perrimon, "Protein complex–based analysis framework for high-throughput data sets," *Sci. Signaling*, vol. 6, no. 264, 2013, Art. no. rs5.

[25] R. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2453–2465, Oct. 2014.

[26] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li, "Finding maximal k-edge-connected subgraphs from a large graph," in *Proc. Int. Conf. Extending Database Technol.*, 2012, pp. 480–491.

[27] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 205–216.

[28] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 277–288.

[29] L. Qin, R. Li, L. Chang, and C. Zhang, "Locally densest subgraph discovery," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 965–974.

[30] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1233–1244, 2016.

[31] X. Huang and L. V. S. Lakshmanan, "Attribute-driven community search," *Proc. VLDB Endowment*, vol. 10, no. 9, pp. 949–960, 2017.

[32] R. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *Proc. VLDB Endowment*, vol. 8, no. 5, pp. 509–520, 2015.

[33] R. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng, "Skyline community search in multi-valued networks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2018, pp. 457–472.

[34] R. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2018, pp. 797–808.

[35] R. Li, Q. Dai, L. Qin, G. Wang, X. Xiao, J. X. Yu, and S. Qiao, "Efficient signed clique search in signed networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2018, pp. 245–256.

[36] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg, "Predicting positive and negative links in online social networks," in *Proc. Int. Conf. World Wide Web*, 2010, pp. 641–650.

[37] J. Ye, H. Cheng, Z. Zhu, and M. Chen, "Predicting positive and negative links in signed social networks by transfer learning," in *Proc. Int. Conf. World Wide Web*, 2013, pp. 1477–1488.

[38] D. Song and D. A. Meyer, "Recommending positive links in signed social networks by optimizing a generalized AUC," in *Proc. AAAI Conf. Artif. Intell.*, 2015, pp. 290–296.

[39] D. Song, D. A. Meyer, and D. Tao, "Efficient latent link recommendation in signed networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1105–1114.

[40] J. Tang, C. C. Aggarwal, and H. Liu, "Recommendations in signed social networks," in *Proc. Int. Conf. World Wide Web*, 2016, pp. 31–40.

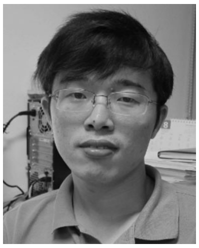
[41] P. Doreian and A. Mrvar, "Partitioning signed social networks," *Social Netw.*, vol. 31, no. 1, pp. 1–11, 2009.

[42] J. Cadena, A. K. S. Vullikanti, and C. C. Aggarwal, "On dense subgraphs in signed network streams," in *Proc. IEEE Int. Conf. Data Mining*, 2016, pp. 51–60.

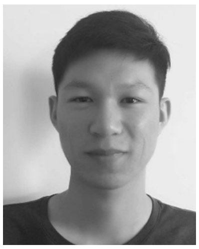
[43] M. Gao, E. Lim, D. Lo, and P. K. Prasetyo, "On detecting maximal quasi antagonistic communities in signed graphs," *Data Mining Knowl. Discovery*, vol. 30, no. 1, pp. 99–146, 2016.

[44] L. Chu, Z. Wang, J. Pei, J. Wang, Z. Zhao, and E. Chen, "Finding gangs in war from signed networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1505–1514.

[45] J. Tang, Y. Chang, C. C. Aggarwal, and H. Liu, "A survey of signed network mining in social media," *ACM Comput. Surveys*, vol. 49, no. 3, pp. 42:1–42:37, 2016.



Rong-Hua Li received the PhD degree from the Chinese University of Hong Kong, in 2013. He is currently an associate professor with the Beijing Institute of Technology (BIT), Beijing, China. Before joining BIT in 2018, he was an assistant professor with Shenzhen University. His research interests include graph data management and mining, social network analysis, graph computation systems, and graph-based machine learning.

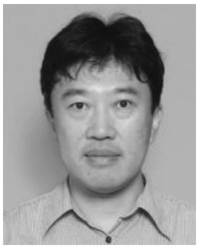


Qiangqiang Dai is working toward the master's degree at Shenzhen University, Shenzhen, China. Currently, he is also a research assistant with the Beijing Institute of Technology, Beijing, China. His research interests include graph data management and mining, social network analysis, and graph computation systems.

Lu Qin received the bachelor's degree from the Department of Computer Science and Technology, Renmin University of China, in 2006, and the PhD degree from the Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong, in 2010. He is now a senior lecturer in the Centre of Quantum Computation and Intelligent Systems (QCIS), University of Technology, Sydney (UTS). His research interests include parallel big graph processing, I/O efficient algorithms on massive graphs, and keyword search in graph data.

Guoren Wang received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, China, in 1988, 1991, and 1996, respectively. Currently, he is a professor with the Department of Computer Science, Northeastern University, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management. He has published more than 100 research papers.

Xiaokui Xiao received the PhD degree in computer science from the Chinese University of Hong Kong, in 2008. He is currently an associate professor with the National University of Singapore (NUS), Singapore. Before joining NUS in 2018, he was an associate professor with Nanyang Technological University (NTU). His research interests include data privacy, spatial databases, graph databases, and parallel computing.



Jeffery Xu Yu received the BE, ME, and PhD degrees in computer science from the University of Tsukuba, Japan, in 1985, 1987, and 1990, respectively. He has held teaching positions with the Institute of Information Sciences and Electronics, University of Tsukuba, and with the Department of Computer Science, Australian National University, Australia. Currently, he is a professor with the Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong, Hong Kong. His current research interests include graph database, graph mining, keyword search in relational databases, and social network analysis.

Shaojie Qiao received the BS and PhD degrees from Sichuan University, Chengdu, China, in 2004 and 2009, respectively. From 2007 to 2008, he was a visiting scholar with the School of Computing, National University of Singapore. He is currently a professor with the School of Cybersecurity, Chengdu University of Information Technology, Chengdu. He has led several research projects in moving objects databases and trajectory data mining. He authored more than 40 high-quality papers and co-authored more than 90 papers. His research interests include trajectory prediction and intelligent transportation systems.