Finding Gangs in War from Signed Networks [*]

Lingyang Chu[†], Zhefeng Wang[‡], Jian Pei[†], Jiannan Wang[†],
Zijin Zhao[†] and Enhong Chen[‡]
[†]Simon Fraser University, Burnaby, Canada
[‡]University of Science and Technology of China,Hefei, China
lca117@sfu.ca, zhefwang@mail.ustc.edu.cn, jpei@cs.sfu.ca, jnwang@sfu.ca,
zijinz@sfu.ca, cheneh@ustc.edu.cn

ABSTRACT

Given a signed network where edges are weighted in real number, and positive weights indicate cohesion between vertices and negative weights indicate opposition, we are interested in finding k-Oppositive Cohesive Groups (k-OCG). Each k-OCG is a group of k subgraphs such that (1) the edges within each subgraph are dense and cohesive; and (2) the edges crossing different subgraphs are dense and oppositive. Finding k-OCGs is challenging since the subgraphs are often small, there are multiple k-OCGs in a large signed network, and many existing dense subgraph extraction methods cannot handle edges of two signs. We model k-OCG finding task as a quadratic optimization problem. However, the classical Proximal Gradient method is very costly since it has to use the entire adjacency matrix, which is huge on large networks. Thus, we develop FOCG, an algorithm that is two orders of magnitudes faster than the Proximal Gradient method. The main idea is to only search in small subgraphs and thus avoids using a major portion of the adjacency matrix. Our experimental results on synthetic and real data sets as well as a case study clearly demonstrate the effectiveness and efficiency of our method.

1. INTRODUCTION

The US presidential primaries and then the US presidential election of 2016 are foreordained a hot topic in social media this year. In the process of social media analysis, to zoom in, it is interesting to find several groups of individuals in an active social network, such as Facebook and Twitter, so that within each group the individuals are of the same mind while different groups have very different opinions. We call such a data mining problem *finding gangs in war.*

*This research is supported in part by an NSERC Discovery grant, the Canada Research Chair program, and a sponsored research grant from Huawei Technologies Co. Ltd. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

KDD '16, August 13-17, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4232-2/16/08. . . $15.00
DOI: http://dx.doi.org/10.1145/2939672.2939855

Technically, we assume a *signed network* where edges between vertices are either cohesive or oppositive. A real number as the weight can be associated with each edge to model the strength of the cohesion (a positive weight) or opposition (a negative weight) [23]. Given a signed network, finding gangs in war tries to find a set of subgraphs such that within each subgraph edges are dense and cohesive, and the edges crossing different subgraphs are dense and oppositive.

Finding gangs in war is a useful exercise that enjoys many interesting application scenarios. For example, the international relationships between countries can be modeled as a signed network [1], where each vertex represents a country. Friendly countries are connected by cohesive edges and hostile countries are connected by oppositive edges. Finding gangs in war in this network reveals hostile groups of allied forces, such as the Allied and Axis power during World War II. Social opinion networks with cohesive and oppositive relationships are well studied [9], where each vertex represents a user, a cohesive edge indicates a cohesive relationship and an oppositive edge represents an oppositive relationship. Finding gangs in war from such signed networks discovers multiple groups of friends with strong inter-group enmity, such as the rival groups of voters that support different political leaders. Interestingly, finding gangs in war can be used in applications well beyond social networks. For example, the network of adjectives in the WordNet database [16] is also a signed network, where each vertex represents an adjective, the synonyms are connected by cohesive edges and the antonyms are connected by oppositive edges. From this signed network, finding gangs in war can identify groups of synonyms that are antonymous with each other, which will be further demonstrated in Section 5.6.

As to be reviewed in Section 2, there are established methods for signed network partitioning, antagonistic community mining and dense subgraph finding. Can we adapt the existing methods to tackle the problem of finding gangs in war from signed networks? Unfortunately, the answer is no due to the following unique challenges.

First, the "gangs in war" are typically small groups and are hard to be located. Most vertices and edges in a network may not even participate in a "gang in war". Although there are many existing methods on partitioning signed networks so that each partition is relatively cohesive [3, 11], those methods cannot find needles from a haystack – they cannot find small groups of dense cohesive subgraphs effectively.

Second, there often exist multiple groups of "gangs" in war, such as multiple groups of synonym adjectives that are antonymous group-wise. Existing signed network partitioning methods are not able to find alternative partitionings.

Third, there may be more than two "gangs in war", such as the multiple groups of voters supporting different presidential candidates. The existing antagonistic community mining methods are not able to detect the community containing more than two "gangs".

Last but not least, the signed edge weights create a huge challenge for the existing dense subgraph detection methods. To the best of our knowledge, the existing dense subgraph extraction methods only consider edges of the same sign and cannot distinguish edges of two signs. The well defined concepts and algorithms for dense subgraph detection on unsigned networks [23] no longer hold in signed networks.

In this paper, we tackle the novel problem of finding gangs in war from signed networks. We make several contributions.

First, we formulate the problem of finding gangs in war as finding k-Oppositive Cohesive Groups (k-OCG). A k-OCG is a set of k subgraphs such that each subgraph is cohesive, that is, having high intra-subgraph cohesion, and the subgraphs are oppositive to each other, that is, among the subgraphs there is a high inter-subgraph opposition. We model the problem as a constrained quadratic optimization problem.

Second, we show that the classical Proximal Gradient method (PG) [18] is very costly in finding k-OCGs on large signed networks, since it has to use the entire adjacency matrix. We develop a two-phase iterative method FOCG that is experimentally two orders of magnitudes faster than PG on average. The key idea is that FOCG confines the search in small subgraphs instead of the whole graph so that it only uses small sub-matrices of the adjacency matrix. Specifically, FOCG iteratively conducts a locate phase and an update phase. The locate phase increases the intra-subgraph cohesion and inter-subgraph opposition of a subgraph by removing weak vertices that contribute little to the cohesion and opposition. The update phase further increases the cohesion and opposition of the small subgraph by adding new vertices that contribute more to the objective.

Third, we report an extensive experimental study on both synthetic and real-world data sets. The results clearly show that our method is accurate, efficient and scalable in finding k-OCGs. We also conduct a case study on the adjective network sampled from the WordNet database [16], where the detected k-OCGs accurately identify significant groups of synonyms that are antonymous with each other.

2. RELATED WORK

To the best of our knowledge, finding k-OCGs in general is a new problem that has not been touched in literature. It is related to the problems of signed network partitioning, antagonistic community mining, and dense subgraph extraction on unsigned networks.

2.1 Signed Network Partition

The signed network partitioning methods partition a signed network into several cohesive subgraphs. The partitioning problem is often transformed into a traditional data clustering problem and solved with classic techniques.

Spectral clustering is one of the most widely adopted techniques. Kunegis *et al.* [10] proposed the signed Laplacian matrix for signed networks. Performing classic spectral clustering algorithms [26] with such signed Laplacian matrix results in subgraphs with only cohesive edges. However, Kunegis's Laplacian matrix [10] tends to separate vertices connected by oppositive edges rather than group vertices connected by cohesive edges. To solve this problem, Zheng *et al.* [31] proposed the balanced normalized signed Laplacian matrix.

As an alternative approach, Doreian *et al.* [6] first defined the objective function for signed network partitioning as $E = \alpha N_n + (1-\alpha)N_p$, where N_p is the number of cohesive edges between subgraphs and N_n is the number of oppositive edges within subgraphs. Many methods attempt to partition a signed network by minimizing Doreian's objective function E. For example, Traag *et al.* [24] used simulated annealing, and Liu *et al.* [11] applied the k-balanced social theory to solve a similar optimization problem.

All the signed network partitioning methods partition the entire signed network. However, in real applications, significant k-OCGs always exist in small local regions of a large and sparse signed network. Such local regions are not known beforehand. Thus, partitioning the entire signed network cannot effectively find significant k-OCGs.

2.2 Antagonistic Community Mining

The antagonistic community mining methods [29, 15, 14, 30, 7] aim to find two cohesive communities that are antagonistic against each other. This can be regarded as a special case where there are two "gangs" in war.

There are generally two categories of such methods. The direct methods [15, 14, 7] mine antagonistic communities directly from signed networks. The indirect methods [29, 30] take a transaction database of user ratings as the input and detect antagonistic communities using frequent patterns. Both the direct and indirect methods cannot be straightforwardly extended to find more than two communities that are antagonistic against each other.

2.3 Dense Subgraph Extraction

The dense subgraph finding problem on unsigned network has been well investigated [25, 5]. Motzkin *et al.* [17] formulated the dense subgraph seeking problem on an un-weighted graph as a quadratic optimization problem on the simplex. Pavan *et al.* [19] extended the method by Motzkin *et al.* to weighted graphs by reformulating the dense subgraph detection problem as a standard quadratic optimization (StQP) problem and solved it using the Dominant Set (DS) method [19]. According to Bulò *et al.* [2], the time complexity of DS is $\mathcal{O}(n^2)$ for a graph of n vertices. Bulò *et al.* [2] also proposed a more efficient method, Infection Immunization Dynamics (IID), to solve the StQP problem [19].

Both DS and IID search an entire weighted graph. Liu *et al.* [13, 12] discovered that most dense subgraphs exist in local regions. By leveraging such locality property of dense subgraphs, Liu *et al.* [13, 12] efficiently solved the StQP problem by searching only small subgraphs.

All these methods find dense subgraphs on unsigned graphs by seeking the local maximums of the StQP problem [19]. However, due to the existence of oppositive edges, the StQP problem [19] becomes undefined on signed networks. As a result, existing dense subgraph detection methods cannot be straightforwardly extended to solve the k-OCG detection problem on signed networks.

3. PROBLEM FORMULATION

In this paper, we use bold lower case characters (e.g., $\mathbf{x}$, $\mathbf{y}$, $\mathbf{e}$) and upper case characters with subscript (e.g., X_j, X_h, X_l) to represent column vectors. Upper case characters (e.g., X, M, E) are used to represent matrices, sets or graphs. Some frequently used notations are summarized in Table 1.

Table 1: Frequently used notations.

Notation	Description
U	The universal index set of all vertices.
$\mathcal{S}$	A k-subgraph set. $\mathcal{S} = \{S_1, S_2, ..., S_k\}$, S_i is the i-th subgraph.
$\mathbb{S}$	A set of k-OCGs. $\mathbb{S} = \{\mathcal{S}_1^*, \mathcal{S}_2^*, ..., \mathcal{S}_p^*\}$, $\mathcal{S}_i^*$ is the i-th k-OCG.
X	An n-by-k dimensional matrix that represents a k-subgraph set.
X^*	A KKT point of $F(X)$ on graph G.
X_j^*	A KKT point of $f(X_j)$ on graph G.
$\hat{X}_j$	A KKT point of $f(X_j)$ on subgraph S_j.
$\hat{S}_j$	The subgraph induced from $\hat{X}_j$.

A *signed graph* is a triple $G = (V, E, A)$, where $V = \{v_1, \dots, v_n\}$ is a set of n vertices, $E = \{(v_i, v_j) \mid i, j \in [1, n]\}$ is a set of edges, and A is an n-by-n *signed adjacency matrix* that describes the relationship between vertices. The entry $A_{i,j}$ at the i-th row and the j-th column of A is positive if (v_i, v_j) is a cohesive edge, negative if (v_i, v_j) is an oppositive edge, and 0 otherwise. The absolute value $|A_{i,j}|$ measures the strength of the cohesion or opposition.

The signed network adjacency matrix A can be rewritten into $A = A^+ - A^-$. A^+ is the *cohesion network* and is composed of all cohesive edges in A, that is, $A^+_{i,j} = \max(A_{i,j}, 0)$. A^- is the *opposition network*, where $A^-_{i,j} = |\min(A_{i,j}, 0)|$.

Let $U = \{1, \dots, n\}$ be the universal index set of all vertices in the set of vertices V. From any index subset $S \subset U$, we can induce a subgraph $G_S = \{V_S, E_S, A_S\}$, where $V_S = \{v_i \mid i \in S\}$, $E_S = \{(v_i, v_j) \mid i, j \in S\}$, and $A_S = [A_{i,j} \mid i, j \in S]$ is the signed adjacency matrix that describes the relationship between each pair of vertices in V_S.

A common way to represent a subgraph G_S is to associate it with a non-negatively-valued n-dimensional column vector $\mathbf{x} = [x_1, x_2, \dots, x_n]$ in the standard simplex $\triangle^n$, where the i-th dimension x_i indicates the participation of vertex v_i in G_S, that is, the weight of vertex v_i in G_S, and simplex $\triangle^n = \{\mathbf{x} \mid \sum_{i=1}^n x_i = 1, x_i \geq 0\}$. Particularly, if $x_i = 0$, vertex v_i does not belong to G_S. If $x_i > 0$, v_i participates in G_S. Given G_S and its vector representation $\mathbf{x}$, we can immediately have the set of indexes of the vertices in G_S as $S = \{i \in U \mid x_i > 0\}$. In the rest of the paper, we refer to a subgraph by the vector representation $\mathbf{x}$ and the set of indexes of vertices S interchangeably.

We are interested in finding *k-oppositive cohesive groups* (k-OCG), where each cohesive group is a dense subgraph dominated by cohesive edges and thus has high intra-subgraph cohesion. At the same time, among the groups there are dense oppositive edges and thus the k-OCG as a set has high inter-subgraph opposition.

To model a k-OCG, we introduce the notion of *k-subgraph set*, denoted by $\mathcal{S} = \{S_1, S_2, ..., S_k\}$, which is a set of k subgraphs. We represent each subgraph $S_j \in \mathcal{S}$ $(1 \leq j \leq k)$ by an n-dimensional column vector $X_j \in \triangle^n$. Consequently, we represent $\mathcal{S}$ by an n-by-k dimensional matrix X where the j-th column $X_j \in \triangle^n$ represents subgraph S_j. Obviously, we have $S_j = \{i \in U \mid X_{i,j} > 0\}$, where $X_{i,j}$ is the entry at the i-th row and the j-th column of matrix X.

Pavan *et al.* [19] proposed a widely used measure for intra-subgraph cohesion. For subgraph $\mathbf{x} \in \triangle^n$, define

$$g^+(\mathbf{x}) = \mathbf{x}^\top A^+ \mathbf{x} = \sum_{i=1}^n \sum_{j=1}^n x_i x_j A^+_{i,j} \tag{1}$$

Since $\sum_{i=1}^n x_i = 1$ as $\mathbf{x} \in \triangle^n$, Equation 1 is the weighted average of cohesive edge weights in subgraph $\mathbf{x}$.

Similarly, we can measure the weighted average of oppositive edge weights between two subgraphs $\mathbf{x}$ and $\mathbf{y}$ by

$$g^-(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top A^- \mathbf{y} = \sum_{i=1}^n \sum_{j=1}^n x_i y_j A^-_{i,j}$$

We can further define the intra-subgraph cohesion of a k-OCG $\mathcal{S}$ as the sum of all intra-subgraph cohesion of each $S_j \in \mathcal{S}$, that is,

$$g^+(\mathcal{S}) = \sum_{j=1}^k g^+(X_j) = tr(X^\top A^+ X)$$

where $tr(\cdot)$ is the *trace operator*.

We can further define the inter-subgraph opposition of $\mathcal{S}$ as the sum of the inter-subgraph opposition for each pair of subgraphs in $\mathcal{S}$, that is,

$$g^-(\mathcal{S}) = \sum_{h \neq j} g^-(X_h, X_j) = \overline{tr}(X^\top A^- X)$$

where $\overline{tr}(M)$ is the *complementary trace* that sums up all non-diagonal entries of matrix M, that is, $\overline{tr}(M) = \sum_{i \neq j} M_{i,j}$. It can be verified that $\overline{tr}(M_1) \pm \overline{tr}(M_2) = \overline{tr}(M_1 \pm M_2)$.

A *k-oppositive cohesive groups* (*k-OCG*) is a k-subgraph set that has a large intra-subgraph cohesion $g^+(\mathcal{S})$ and a large inter-subgraph opposition $g^-(\mathcal{S})$. Accordingly, we define the k-OCG detection problem as

$$\begin{aligned} &\max_{X \in \triangle^{n \times k}} F(X) \\ &F(X) = g^+(\mathcal{S}) + \alpha g^-(\mathcal{S}) - \beta \overline{tr}(X^\top X) \\ &\qquad = tr(X^\top A^+ X) + \overline{tr}(X^\top H X) \end{aligned} \tag{2}$$

where $H = (\alpha A^- - \beta I)$, I is the identify matrix, parameter $\alpha > 0$ controls the tradeoff between intra-subgraph cohesion and inter-subgraph opposition, and the term $\beta \overline{tr}(X^\top X)$ penalizes the overlap between different cohesive subgraphs.

Every local maximum of $F(X)$ corresponds to a k-OCG in graph G. However, not all local maximums are of the same significance. More often than not, in real applications we want to find the significant k-OCGs of large intra-subgraph cohesion and inter-subgraph opposition. Such significant k-OCGs are induced by the local maximums with large values of $F(X)$. Since every local maximum of $F(X)$ satisfies the Karush-Kuhn-Tucker (KKT) conditions [8], every KKT point X^* of $F(X)$ is a potential local maximum of $F(X)$. In the rest of the paper, we focus on detecting significant k-OCGs by seeking the KKT points with large $F(X^*)$ values.

The problem in Equation 2 is a constrained optimization problem that can be solved by classic numerical optimization methods, such as Proximal Gradient method (PG) [18]. However, the classic numerical optimization methods generally operate with the entire adjacency matrix A and often involves the computationally expensive gradient calculation. Therefore, when graph G is large, A is large. The classic numerical optimization methods are not efficient in solving the problem in Equation 2 on large graphs.

Since k-OCGs usually exist in small local regions of a signed network, a lot of information carried by A is redundant and we can accurately and efficiently find a k-OCG using only small submatrices of A without calculating the gradient. Next, we propose the FOCG algorithm that solves the problem in Equation 2 by iteratively seeking the KKT points [8] of $F(X)$, and achieves high efficiency by confining all iterations on small subgraphs.

4. THE FOCG ALGORITHM

We first re-organize Equation 2. For any $j \in [1,k]$,

$$F(X) = f(X_j) + \sum_{h \neq j} X_h^\top A^+ X_h + \sum_{l \neq j} \sum_{\substack{h \neq j \\ h \neq l}} X_l^\top H X_h \quad (3)$$

where $f(X_j) = X_j^\top A^+ X_j + 2\sum_{h \neq j} X_j^\top H X_h$ consists of all the terms in $F(X)$ that are related with X_j.

According to Equation 3, when the other columns of X (i.e., $\{X_h \mid h \neq j, h \in [1,k]\}$) are fixed, we can monotonically increase the value of $F(X)$ by maximizing $f(X_j)$ on variable X_j. Thus, we can find the KKT points of $F(X)$ by optimizing $f(X_j)$ on each column of X (i.e., $\{X_j \in \triangle^n \mid j \in [1,k]\}$). The corresponding optimization problem is

$$\max_{X_j \in \triangle^n} f(X_j) \quad (4)$$

where $f(X_j) = X_j^\top A^+ X_j + 2X_j^\top M_j$ and $M_j = \sum_{h \neq j} HX_h$ is an n-dimensional column vector.

The first term $X_j^\top A^+ X_j$ of $f(X_j)$ in Equation 4 is the intra-subgraph cohesion of subgraph S_j. This term is exactly the objective function of the dense subgraph seeking methods [19, 13, 12]. To understand the second term in Equation 4, we notice the following. First, since $H = (\alpha A^- - \beta I)$, we have $H_{i,l} = \alpha A^-_{i,l}$ when $i \neq l$. Thus, $H_{i,l}$ $(i \neq l)$ represents the opposition between vertices v_i and v_l. Second, we rewrite the i-th entry of HX_h to $(HX_h)_i = \sum_{l \in S_h} H_{i,l} X_{l,h}$. Since $X_{l,h}$ is the weight of vertex v_l in subgraph S_h, $(HX_h)_i$ represents the average opposition between vertex v_i and all the vertices in subgraph S_h. Then, since $M_j = \sum_{h \neq j} HX_h$, the i-th entry of M_j can be written as $M_{i,j} = \sum_{h \neq j} (HX_h)_i$, which is the sum of the average opposition between vertex v_i and all the $(k-1)$ subgraphs in $\mathcal{S} \setminus S_j = \{S_h \in \mathcal{S} \mid h \neq j\}$. By expanding the second term of $f(X_j)$ $2X_j^\top M_j = 2\sum_{i \in S_j} X_{i,j} M_{i,j}$, we can see that $2X_j^\top M_j$ is the weighted average opposition between all the vertices of subgraph S_j and all the other subgraphs in $\mathcal{S} \setminus S_j$.

Next, we illustrate how to efficiently find a k-OCG by seeking a KKT point X^* of $F(X)$. In Section 4.1, we prove that we can find X^* by seeking the KKT points X_j^* for all $j \in [1,k]$. Section 4.2 introduces the LUA method that efficiently finds X_j^* of $f(X_j)$ by searches in small subgraphs. Section 4.3 summarizes the FOCG method, which finds a KKT point X^* of $F(X)$ by seeking X_j^* for all $j \in [1,k]$ with LUA. We also introduce the initialization method IOCG and illustrate how to select significant k-OCGs.

4.1 KKT Points

The following result establishes the relation between a KKT point of $F(X)$ and the KKT points of $f(X_j), j \in [1,k]$. Therefore, finding the KKT points of $f(X_j)$ for all $j \in [1,k]$ can be used to find KKT points of $F(X)$.

THEOREM 1. *$X^* \in \triangle^{n \times k}$ is a KKT point of $F(X)$ if and only if $\forall j \in [1,k]$, X_j^* is a KKT point of $f(X_j)$.*

PROOF SKETCH. We prove here only the biconditional logical connectivity with respect to the stationarity condition of KKT conditions. The proof on the biconditional logical connectivity with respect to the primal feasibility, dural feasibility and complementary slackness is straightforward and thus is omitted for the interest of space.

For X and X_j, since $X \in \triangle^{n \times k} = \{X \mid X_j \in \triangle^n, j \in [1,k]\}$ and $X_j \in \triangle^n = \{X_j \mid \sum_{i=1}^n X_{i,j} = 1, X_{i,j} \geq 0\}$, the Lagrange functions of $F(X)$ and $f(X_j)$ can be written to Equations 5 and 6, respectively.

$$\begin{cases} \mathcal{L}_F(X, \boldsymbol{\mu}, \boldsymbol{\lambda}) = F(X) + R(X) \\ R(X) = \sum_{j=1}^{k} \sum_{i=1}^{n} \mu_{i,j} X_{i,j} - \sum_{j=1}^{k} \lambda_j (\sum_{i=1}^{n} X_{i,j} - 1) \end{cases} \quad (5)$$

$$\begin{cases} \mathcal{L}_f(X_j, \boldsymbol{\mu_j}, \lambda_j) = f(X_j) + r(X_j) \\ r(X_j) = \sum_{i=1}^{n} \mu_{i,j} X_{i,j} - \lambda_j (\sum_{i=1}^{n} X_{i,j} - 1) \end{cases} \quad (6)$$

where $\boldsymbol{\mu}$, $\boldsymbol{\lambda}$, $\boldsymbol{\mu_j}$ and λ_j are Lagrangian multipliers, $\boldsymbol{\mu_j}$ is the j-th column of the n-by-k dimensional matrix $\boldsymbol{\mu}$, and λ_j is the j-th element of the k-dimensional vector $\boldsymbol{\lambda}$. Apparently, $R(X) = \sum_{j=1}^k r(X_j)$.

Denote by $\nabla_X \mathcal{L}_F$ the gradient of $\mathcal{L}_F(X, \boldsymbol{\mu}, \boldsymbol{\lambda})$, which is an n-by-k dimensional matrix. $(\nabla_X \mathcal{L}_F)_j$ is the j-th column of $\nabla_X \mathcal{L}_F$. We have

$$(\nabla_X \mathcal{L}_F)_j = \nabla_{X_j} F(X) + \nabla_{X_j} R(X) \quad (7)$$

where ∇_{X_j} is the gradient operator over variable X_j.

According to Equation 3, we have $\nabla_{X_j} F(X) = \nabla_{X_j} f(X_j)$. Since $R(X) = \sum_{j=1}^k r(X_j)$, we have $\nabla_{X_j} R(X) = \nabla_{X_j} r(X_j)$. By substituting the above two equations into Equation 7, we have

$$(\nabla_X \mathcal{L}_F)_j = \nabla_{X_j} f(X_j) + \nabla_{X_j} r(X_j) = \nabla_{X_j} \mathcal{L}_f \quad (8)$$

where $\nabla_{X_j} \mathcal{L}_f$ is the gradient of $\mathcal{L}_f(X_j, \boldsymbol{\mu_j}, \lambda_j)$.

Sufficiency. If X^* is a KKT point, then $\nabla_{X^*} \mathcal{L}_F = 0$. According to Equation 8, we have $\nabla_{X_j^*} \mathcal{L}_f = 0$, $\forall j \in [1,k]$, thus the stationarity condition holds for each $f(X_j^*)$.

Necessity. If $\forall j \in [1,k]$, X_j^* is a KKT point of $f(X_j)$, then we have $\nabla_{X_j^*} \mathcal{L}_f = 0$ for all $j \in [1,k]$. Thus, $\nabla_{X^*} \mathcal{L}_F = 0$. Therefore, the stationarity condition holds for $F(X^*)$. □

THEOREM 2. *$X_j^* \in \triangle^n$ is a KKT point of $f(X_j)$ if and only if*

$$R_i(X_j^*) \begin{cases} = Q(X_j^*) & if \quad i \in S_j^* \\ \leq Q(X_j^*) & if \quad i \in U \setminus S_j^* \end{cases} \quad (9)$$

where $S_j^* = \{i \in U \mid X_{i,j}^* > 0\}$ is the subgraph induced from X_j^*, $R_i(X_j^*) = (A^+X_j^*)_i + M_{i,j}$, $Q(X_j^*) = (X_j^*)^\top R(X_j^*)$, and $(A^+X_j^*)_i$ is the i-th entry of $A^+X_j^*$.

PROOF. A KKT point X_j^* of $f(X_j)$ must satisfy the KKT conditions: (1) Stationarity: $2(A^+X_j^*)_i + 2M_{i,j} + \mu_{i,j} - \lambda_j = 0, \forall i \in [1,n]$; (2) Complementary slackness: $\sum_{i=1}^n \mu_{i,j} X_{i,j}^* = 0$; (3) Dual feasibility: $\mu_{i,j} \geq 0, \forall i \in [1,n]$; and (4) Primal feasibility: $X_{i,j}^* \geq 0, \forall i \in [1,n]$ and $\sum_{i=1}^n X_{i,j}^* = 1$. Note that, the primal feasibility trivially holds, since $X_j^* \in \triangle^n$.

Sufficiency. If X_j^* is a KKT point, then X_j^* satisfies all the above KKT conditions. Considering $X_{i,j}$ and $\mu_{i,j}$ are non-negative for all $i \in [1,n]$, the complementary slackness condition can be rewritten as

$$\forall i \in [1,n], \text{ if } X_{i,j}^* > 0, \text{ then } \mu_{i,j} = 0 \quad (10)$$

Since $X^*_{i,j} > 0$ means $i \in S^*_j$, we transform Equation 10 into

$$\forall i \in [1, n], \text{ if } i \in S^*_j, \text{ then } \mu_{i,j} = 0 \tag{11}$$

By doing simple calculations on both Equation 11 and the stationary condition, we can rewrite the KKT conditions as

$$R_i(X^*_j)\begin{cases} = \frac{\lambda_j}{2} & if \quad i \in S^*_j \\ \le \frac{\lambda_j}{2} & if \quad i \in U \setminus S^*_j \end{cases} \tag{12}$$

which indicates $R_i(X^*_j) = \frac{\lambda_j}{2}$, for all $i \in S^*_j$.

Since $\sum_{i\in S^*_j} X^*_{i,j} = 1$, we have $Q(X^*_j) = (X^*_j)^\top R(X^*_j) = \sum_{i\in S^*_j} X^*_{i,j} R_i(X^*_j) = \frac{\lambda_j}{2}$. Plugging this into Equation 12, we have Equation 9.

Necessity. If Eqnation 9 holds, then there always exists a set of Lagrangian multipliers $\mu_{i,j}, i \in [1, n]$ and λ_j as follows that make the KKT conditions hold.

$$\mu_{i,j} = \begin{cases} 0 & if \quad i \in S^*_j \\ \lambda_j - 2R_i(X^*_j) & if \quad i \in U \setminus S^*_j \end{cases}$$

where $\lambda_j = 2Q(X^*_j)$. This means that X^*_j satisfies the KKT conditions and thus is a KTT point of $f(X_j)$. $\square$

For $R_i(X^*_j)$ in Equation 9, the first term $(A^+X^*_j)_i$ is the average cohesion between vertex v_i and all vertices in subgraph S^*_j, and captures the contribution from vertex v_i to the intra-subgraph cohesion of S^*_j. The second term $M_{i,j}$ is the average opposition between v_i and the vertices of the other $(k-1)$ subgraphs in $\mathcal{S} \setminus S^*_j$, and captures the contribution from v_i to the inter-subgraph opposition between S^*_j and the other subgraphs in $\mathcal{S} \setminus S^*_j$. Thus, $R_i(X^*_j)$ measures the contribution from v_i to both the intra-subgraph cohesion and inter-subgraph opposition of S^*_j. For $Q(X^*_j)$ in the same equation, since $Q(X^*_j) = (X^*_j)^\top R(X^*_j) = \sum_{i\in S^*_j} X^*_{i,j} R_i(X^*_j)$, we know that $Q(X^*_j)$ is the weighted average contribution from all vertices in S^*_j.

Theorem 2 indicates that, for a KKT point X^*_j, the contribution $R_i(X^*_j)$ by each vertex v_i inside subgraph S^*_j is equal to the average contribution $Q(X^*_j)$ by S^*_j. Moreover, the contribution $R_i(X^*_j)$ by each vertex v_i outside subgraph S^*_j is not larger than the average contribution $Q(X^*_j)$ by S^*_j.

According to Theorem 2, any point X^*_j satisfying the KKT conditions in Equation 9 is a KKT point of $f(X_j)$ in graph G. Thus, any $\hat{X}_j \in \triangle^n$ is a KKT point in subgraph S_j if

$$R_i(\hat{X}_j)\begin{cases} = Q(\hat{X}_j) & if \ i \in \hat{S}_j \\ \le Q(\hat{X}_j) & if \ i \in S_j \setminus \hat{S}_j \end{cases} \tag{13}$$

where $\hat{S}_j = \{i \in U \mid \hat{X}_{i,j} > 0\}$ is a subset of S_j.

Apparently, a KKT point $\hat{X}_j$ on S_j induces a subgraph $\hat{S}_j \subset S_j$, where no vertex v_i in $S_j \setminus \hat{S}_j$ has a larger contribution $R_i(\hat{X}_j)$ than the average contribution $Q(\hat{X}_j)$. However, since $S_j \setminus \hat{S}_j$ is not equal to $U \setminus S^*_j$, a KKT point in subgraph S_j is not necessarily a KKT point in graph G. Therefore, we further explore the relationship between a KKT point in subgraph S_j and a KKT point in graph G as follows.

Corollary 2.1. *If $\hat{X}_j$ is a KKT point of $f(X_j)$ in subgraph S_j, then $\hat{X}_j$ is a KKT point of $f(X_j)$ in graph G if and only if $\Omega_j = \{i \in U \setminus \hat{S}_j \mid R_i(\hat{X}_j) > Q(\hat{X}_j)\} = \emptyset$.*

Proof. (**Sufficiency.**) If $\Omega_j = \emptyset$, then $\hat{X}_j$ also satisfies the KKT conditions in graph G (i.e., Equation 9), thus $\hat{X}_j$ is also a KKT point in graph G.

(**Necessity.**) $\Omega_j \neq \emptyset$ indicates the KKT conditions in graph G do not hold for $\hat{X}_j$, which means $\hat{X}_j$ is not a KKT point in graph G. $\square$

Ω_j contains all the vertices that are outside subgraph $\hat{S}_j$ and contribute more cohesion and opposition to $\hat{S}_j$ than the average contribution $Q(\hat{X}_j)$. Adding the vertices in Ω_j into subgraph $\hat{S}_j$ can further increase the average cohesion and opposition of $\hat{S}_j$, thus can increase the value of $f(\hat{X}_j)$.

In summary, a KKT point of $f(X_j)$ corresponds to a potential dense subgraph S^*_j that possesses both large intra-subgraph cohesion within itself and large inter-subgraph opposition with the other subgraphs in $\mathcal{S} \setminus S^*_j$. Since a dense subgraph usually consists of small subsets of vertices [13], the size of S^*_j is usually small if X^*_j is a KKT point of G. Based on this insight, we propose the Locate and Update Algorithm next, which finds KKT points of graph G by constraining searches in small subgraphs.

4.2 The Locate and Update Algorithm (LUA)

In this section, we introduce the Locate and Update Algorithm (LUA) that efficiently finds a KKT point of $f(X_j)$ in graph G. The key to the efficiency of LUA is that it always works on a small subgraph S_j and iteratively updates S_j until a KKT point in graph G is found.

We first transform the problem in Equation 4 into the following standard form of dense subgraph seeking problem

$$\max_{X_j \in \triangle^n} f(X_j) \tag{14}$$

where $f(X_j) = X_j^\top B_j X_j$ and $B_j = A^+ + \mathbf{e}M_j^\top + M_j\mathbf{e}^\top$ is an n-by-n dimensional matrix. $\mathbf{e}$ is a column vector with all entries equal to 1.

When matrix B_j is given, there are many existing dense subgraph seeking algorithms [19, 2, 13] that can be used to solve the problem in Equation 14. However, since matrix B_j is not sparse, it is hard to calculate and store B_j when graph G is large. Without materializing B_j, we cannot solve the problem in Equation 14 by a simple extension of the existing dense subgraph seeking algorithms [19, 2, 13].

To tackle this problem, we design the LUA algorithm, which effectively avoids computing the entire matrix B_j by confining computation in small subgraphs. LUA iteratively conducts a locate phase and an update phase. The *locate phase* locates a KKT point $\hat{X}_j$ in subgraph S_j and reduces S_j into its subgraph $\hat{S}_j$. The *update phase* updates subgraph $\hat{S}_j$ by taking more vertices in Ω_j whose contribution $R_i(\hat{X}_j)$ to the intra-subgraph cohesion and inter-subgraph opposition of subgraph $\hat{S}_j$ is larger than the average contribution $Q(\hat{X}_j)$. The iteration continues until $\Omega_j = \emptyset$. According to Corollary 2.1, $\hat{X}_j$ is also a KKT point in graph G. Next, we discuss the details of LUA.

4.2.1 The Locate Phase

The locate phase locates a KKT point $\hat{X}_j$ in subgraph S_j and reduces S_j into its subgraph $\hat{S}_j = \{i \in U \mid \hat{X}_{i,j} > 0\}$.

Given an initialization of $X_j(0)$ that is obtained by a heuristic method to be discussed in Algorithm 3, the locate phase finds a KKT point $\hat{X}_j$ in subgraph S_j by the Replicator Dynamics (RD) iteration [27]. At the t-th iteration,

$$X_{i,j}(t+1) = X_{i,j}(t)\frac{(B_j X_j(t))_i}{X_j(t)^\top B_j X_j(t)} \tag{15}$$

where $(B_j X_j(t))_i$ is the i-th entry of the n-dimensional vector $B_j X_j(t)$. According to [27], the iterations converge to $\hat{X}_j$. The resulting subgraph is $\hat{S}_j$.

A nice property of Equation 15 is that, if $X_{i,j}(t) = 0$, then $X_{i,j}(t+1) = 0$. Thus, we can confine all computation of Equation 15 within subgraph S_j by initializing $X_j(0)$ as $X_j(0) = \{X_{i,j}>0 \mid i \in S_j\}$. Since $\forall i \notin S_j, X_{i,j}(t) = 0$, we only need to calculate and store a sub-matrix B_{S_j} of B_j that corresponds to the edge set E_{S_j} of subgraph S_j.

The *locate phase* efficiently finds a KKT point $\hat{X}_j$ of subgraph S_j and reduces S_j into its subgraph $\hat{S}_j$. However, according to Corollary 2.1, $\hat{X}_j$ may not necessarily be a KKT point in graph G. Thus, we use an update phase to further increase the value of $f(\hat{X}_j)$ and make sure that LUA converges to a KKT point in graph G.

4.2.2 The Update Phase

According to Corollary 2.1, if $\Omega_j = \emptyset$, then $\hat{X}_j$ is already a KKT point in graph G, thus the LUA iteration converges. However, if $\Omega_j \neq \emptyset$, then $\hat{X}_j$ is not a KKT point in graph G. In this case, we update $\hat{X}_j$ with a carefully designed n-dimensional vector $\mathbf{b}$ and a step size σ, such that $f(\hat{X}_j + t\mathbf{b}) > f(\hat{X}_j)$ under the constraint $(\hat{X}_j + \sigma\mathbf{b}) \in \triangle^n$.

The i-th entry of $\mathbf{b} = [b_1, \ldots, b_n]$ is defined as

$$b_i = \begin{cases} R_i(\hat{X}_j) - Q(\hat{X}_j) & if \quad i \in \Omega_j \\ -s\hat{X}_{i,j} & if \quad i \in \hat{S}_j \\ 0 & otherwise \end{cases} \tag{16}$$

where $s = \sum_{i \in \Omega_j} b_i$. We know $s>0$ from the definition of Ω_j.

According to Corollary 2.1, if $i \in \Omega_j$, then the *contribution* $R_i(\hat{X}_j)$ of vertex v_i is larger than the average *contribution* $Q(\hat{X}_j)$ of all vertices in subgraph $\hat{S}_j$. Adding v_i into subgraph $\hat{S}_j$ further increases the intra-subgraph cohesion and inter-subgraph opposition of $\hat{S}_j$, thus increases the value of $f(\hat{X}_j)$. Therefore, we set $b_i = R_i(\hat{X}_j) - Q(\hat{X}_i)$. In this case, b_i assigns a positive weight to the i-th entry of $(\hat{X}_j + \sigma\mathbf{b})$, which is equivalent to adding v_i into the updated subgraph.

Another useful property of b_i is that the constraint $(\hat{X}_j + \sigma\mathbf{b}) \in \triangle^n$ always holds for all $\sigma \in [0, \frac{1}{s}]$ due to the following. First, since $\sum_{i \in \hat{S}_j} \hat{X}_{i,j} = 1$, we have $\sum_{i=1}^n b_i = \sum_{i \in \Omega_j} b_i + \sum_{i \in \hat{S}_j}(-s\hat{X}_{i,j}) = s - s = 0$, thus $\sum_i (\hat{X}_{i,j} + \sigma b_i) = 1$. Second, since $\sigma \in [0, \frac{1}{s}]$ and $s > 0$, we have $\hat{X}_{i,j} + \sigma b_i \geq 0$, $\forall i \in [1, n]$. Then, we can derive the optimal step size by maximizing $f(\hat{X}_j + \sigma\mathbf{b}) - f(\hat{X}_j)$ as

$$\begin{aligned} f(\hat{X}_j + \sigma\mathbf{b}) - f(\hat{X}_j) &= \mathbf{b}^\top A^+ \mathbf{b}\sigma^2 + 2\mathbf{b}^\top (A^+ \hat{X}_j + M_j)\sigma \\ &= \mathbf{b}^\top A^+ \mathbf{b}\sigma^2 + 2(\sum_{i \in \Omega_j} b_i^2)\sigma \end{aligned} \tag{17}$$

When $\Omega_j \neq \emptyset$, we have $\sum_{i \in \Omega_j} b_i^2 > 0$. Thus, the optimal step size σ^* is

$$\sigma^* = \begin{cases} \frac{1}{s} & if \quad \mathbf{b}^\top A^+ \mathbf{b} \geq 0 \\ \min\left(\frac{1}{s}, -\frac{\sum_{i \in \Omega_j} b_i^2}{\mathbf{b}^\top A^+ \mathbf{b}}\right) & if \quad \mathbf{b}^\top A^+ \mathbf{b} < 0 \end{cases} \tag{18}$$

which guarantees $f(\hat{X}_j + \sigma^*\mathbf{b}) - f(\hat{X}_j) > 0$.

To sum up, when $\Omega_j \neq \emptyset$, the update phase updates subgraph $\hat{S}_j$ by taking some vertices in Ω_j and further increases the value of $f(\hat{X}_j)$.

Algorithm 1: The Locate and Update Iteration

```
Input: $X_j(0) \in \triangle^n$.
Output: A KKT point $X_j^*$ of $f(X_j)$ in graph $G$.
 1: Set $\hat{X}_j = X_j(0)$ and update $\Omega_j$.
 2: repeat
 3:   if $\Omega_j \neq \emptyset$ then
 4:     Update phase: $X_j \leftarrow (\hat{X}_j + \sigma^*\mathbf{b})$.
 5:   end if
 6:   Locate phase: Start from $X_j$ and find a KKT
      point $\hat{X}_j$ in subgraph $S_j$ using Equation 15.
 7:   Update $\Omega_j = \{i \in U \setminus \hat{S}_j \mid R_i(\hat{X}_j) > Q(\hat{X}_j)\}$.
 8: until $\Omega_j = \emptyset$.
 9: $X_j^* \leftarrow \hat{X}_j$.
10: return A KKT point $X_j^*$ of $f(X_j)$ in graph $G$.
```

Algorithm 2: The FOCG Algorithm

```
Input: $X(0) \in \triangle^{n \times k}$.
Output: A KKT point $X^*$ of $F(X)$ in graph $G$.
 1: Set $X = X(0)$.
 2: repeat
 3:   for all $j \in [1, k]$ do
 4:     Find a KKT point $X_j^*$ of $f(X_j)$ in graph $G$ by
        Algorithm 1.
 5:     Update the $j$-th column of $X$ by $X_j \leftarrow X_j^*$.
 6:   end for
 7: until $\forall j \in [1, k]$, a KKT point $X_j^*$ of $f(X_j)$ is found.
 8: return a KKT point $X^*$ of $F(X)$ in graph $G$.
```

4.2.3 The Locate and Update Iteration

Algorithm 1 gives the pseudocode of the LUA algorithm. The main computational cost of LUA lies in the locate phase, whose efficiency is largely affected by the size of subgraph S_j. In real world applications, the graph G is usually very sparse, thus the size of both Ω_j and S_j are usually small, which leads to the high efficiency of locate phase. As a result, LUA converges pretty fast on sparse graphs.

We prove the convergence of LUA as follows.

THEOREM 3. *The LUA iteration in Algorithm 1 converges to a KKT point X_j^* in graph G.*

PROOF. By setting all entries of B_j to the maximum value in B_j, we can easily obtain a trivial upper bound of $f(X_j)$. In both the locate phase and the update phase, $f(X_j)$ monotonously increases, therefore, the LUA iteration converges in the perspective of numeric optimization.

Algorithm 1 converges only when $\Omega_j = \emptyset$. According to Corollary 2.1, when LUA converges (i.e., $\Omega_j = \emptyset$), the KKT point $\hat{X}_j$ found in subgraph S_j is a KKT point X_j^* in G. □

4.3 The Complete Algorithm

Algorithm 2 gives the pseudocode of the FOCG algorithm, which finds a KKT point of $F(X)$ (Equation 3) by alternatively optimizing $f(X_j)$ over each column X_j of X.

THEOREM 4. *The FOCG algorithm converges to a KKT point X^* of graph G.*

PROOF. We first prove that FOCG converges. According to Equation 3, increasing $f(X_j)$ equivalently increases

Algorithm 3: The IOCG Algorithm

```
Input: Adjacency matrices of $A^+$ and $A^-$.
Output: An initialization $X(0)$ for FOCG algorithm.
 1: Initialize the seed index set $\boldsymbol{\eta} = \emptyset$.
 2: Calculate the vertex degree vector on positive network
    as $\mathbf{d}^+ = [d_1^+, d_2^+, ..., d_n^+]$, where $d_i^+ = \sum_{j=1}^n A_{i,j}^+$.
 3: Select the first seed $h_1 = Roul(\mathbf{d}^+)$ by roulette wheel
    selection [20] and update $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} \cup h_1$.
 4: for $l \in [2, k]$ do
 5:     Calculate opposition vector $\mathbf{o}^- = [o_1^-, ..., o_i^-, ..., o_n^-]$,
        where $o_i^- = \frac{1}{|\boldsymbol{\eta}|} \sum_{j \in \boldsymbol{\eta}} A_{i,j}^-$.
 6:     Select seed by $h_l = Roul(\mathbf{o}^-)$ and update $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} \cup h_l$.
 7: end for
 8: Initialize $X$ as a $n$-by-$k$ dimensional all zero matrix.
 9: for $j \in [1, k]$ do
10:     Set $X_{\eta_j, j} = 1$, where $\eta_j$ is the $j$-th seed index in $\boldsymbol{\eta}$.
11: end for
12: return $X(0) \leftarrow X$.
```

$F(X)$. Since for each iteration in FOCG, the LUA algorithm monotonously increases $f(X_j)$, the value of $F(X)$ is monotonously increased as well. Due to the fact that $F(X)$ has an upper bound, the FOCG algorithm converges.

The FOCG algorithm does not terminate until X_j^* is a KKT point of $f(X_j)$ for all $j \in [1, k]$. The reason is that, if there exist an X_j^* that is not a KKT point of $f(X_j)$, then $F(X)$ can be further increased by LUA. Since FOCG converges when all X_j^* ($j \in [1, k]$) are KKT points of $f(X_j)$, according to Theorem 1, X^* is a KKT point of graph G. □

For Algorithm 2, a proper initialization of $X(0)$ usually improves the possibility of getting a KKT point X^* with large value of $F(X^*)$. Here, we propose an initialization method IOCG (Algorithm 3) for the initialization of $X(0)$.

IOCG randomly selects k seed vertices as the initializations for the k subgraphs S_i ($1 \leq i \leq k$) in $\mathcal{S}$. The first seed is selected according to the degree of each vertex on network A^+. The *roulette wheel selection* method [20] $h = Roul(d^+)$ randomly selects a vertex v_h with probability $\frac{d_h}{\sum_{i=1}^n d_i}$, thus the vertices with larger degrees are more likely to be selected. Heuristically, a vertex with a large degree in network A^+ is more likely to be a member of a dense subgraph. The other $(k-1)$ seed vertices are selected under the criterion that the opposition between seed vertices should be large. Such seed vertices are more likely to belong to different dense subgraphs such that the group of subgraphs in whole possesses a large inter-subgraph opposition. As a result, IOCG provides a good start point for FOCG to detect a KKT point X^* with large value of $F(X^*)$. Such a KKT point often corresponds to a significant k-OCG $\mathcal{S}^* = \{S_j^* \mid j \in [1, k]\}$, where S_j^* is the subgraph induced by X_j^*.

Let ψ be the set of all KKT points of $F(X)$. The size of ψ is often very large and it is impractical to compute the entire set. However, in real applications, more often than not we are interested in only the significant k-OCGs of large value of $F(X)$. Similar to most dense subgraph detection methods, we adopt the "peeling-off" method [19, 13, 5]. Due to its simplicity and robustness, such a "peeling-off" method is widely used in the task of dense subgraph detection to enumerate the set of KKT points.

Specifically in our case, when a KKT point X^* is obtained, it is first added into the answer set. Then, we remove the vertices and edges that belong to the corresponding k-subgraph set $\mathcal{S}^*$ from graph G and find another KKT point using a new initialization of $X(0)$. Such a process iterates until all vertices in graph G are removed and a set of KKT points $\hat{\psi}$ is obtained. Then, we can search $\hat{\psi}$ for the significant k-OCGs of large value of $F(X^*)$.

5. EXPERIMENTS

In this section, we evaluate the performance of the proposed FOCG algorithm and compare it with the state-of-the-art related signed network partitioning methods including (1) Simple Normalized Signed Graph Laplacian method (SNS) [31], (2) Signed Normalized Laplacian method (SNL) [10], (3) Balance Normalized Cut method (BNC) [4], and (4) Ratio Associate method (RA) [4]. Both the SNS and SNL methods are incorporated in the standard spectral clustering framework [26] to perform the partitioning task on signed network. The codes for BNC and RA were provided by Chiang *et al.* [4]. We also compare the scalability of FOCG and the Proximal Gradient method (PG) [18]. We use the default parameters for all compared methods. For FOCG and PG, we set $\alpha = 0.9$, $\beta = 50$ and $k = 10$ by default. All experiments are performed using MATLAB. We use a PC with Core-i7-3370 CPU (3.40GHz), 16GB memory, and a 5400 rpm hard drive running Ubuntu 15.04. The source code of FOCG is available on GitHub [21].

The following five data sets are used. For the directed networks, we symmetrize the adjacency matrix by $A = \frac{A + A^\top}{2}$.

Synthetic Data Set. The synthetic data set is generated by the data generation method proposed by Chiang *et al.* [4]. We generate four networks with different sparsity. Each network contains 10,000 vertices that form 20 subgraphs.

Slashdot Data Set. The public Slashdot data set is from SNAP [22]. We use the version "soc-sign-Slashdot081106", which contains 77,357 vertices and 516,575 edges.

Epinions Data Set. The Epinions data set is a public data set on SNAP [22]. It is a directed signed network containing 131,828 vertices and 841,372 edges.

Douban Data set. The Douban data set [28] contains a social network of users and the movie ratings of each user. We build the signed network in three steps. First, we induce a cohesive network $G^+ = \{V, E^+\}$ by treating each user as a vertex in vertex set V and their friendships as cohesive edges in edge set E^+. Second, we build an oppositive network $G^- = \{V, E^-\}$ by calculating the average movie rating difference between each pair of users. If the difference is greater than 1, we build an oppositive edge between them. Last, we obtain the signed network $G = \{V, E\}$ by merging G^+ and G^-. Since the set of vertices V in G^+ and G^- are the same, we only merge the set of edges $E = E^+ \cup E^-$. If there are both cohesive edge and oppositive edge between a pair of users, we keep the oppositive edge in G. The network contains 1.59 million vertices and 19.67 million edges.

WordNetAdj Data Set. The WordNetAdj data set is a subset of adjectives sampled from the adjective network of the WordNet database [16]. It contains 12,883 vertices and 39,460 edges, where each vertex represents an adjective. The edge between a pair of synonyms is cohesive and the edge between a pair of antonyms is oppositive.

5.1 Performance Measures

Let $\mathbb{S} = \{\mathcal{S}_1^*, \mathcal{S}_2^*, \ldots, \mathcal{S}_p^*\}$ be a set of p detected k-OCGs, where $\mathcal{S}_i^* = \{S_{i,j}^* \mid j = 1, \ldots, k\}$ is a k-OCG and $S_{i,j}^*$ is a cohesive subgraph in $\mathcal{S}_i^*$. Let n_p be the total number of vertices contained by the set of k-OCGs $\mathbb{S}$. Apparently, we have $n_p \leq n$, where n is the number of vertices in G.

The intra-subgraph cohesion of a single subgraph $S_{i,j}^*$ is

$$\textbf{Cohe}(S_{i,j}^*) = \frac{1}{|S_{i,j}^*| \, (|S_{i,j}^*| - 1)} \sum_{h \in S_{i,j}^*} \sum_{\substack{l \in S_{i,j}^* \\ l \neq h}} A_{h,l}^+$$

where $|S_{i,j}^*|$ is the number of vertices in subgraph $S_{i,j}^*$. $\textbf{Cohe}(S_{i,j}^*)$ is the average cohesive edge weight of $S_{i,j}^*$, which is widely used to measure intra-subgraph cohesion [25].

The inter-subgraph opposition between two subgraphs $S_{i,j}^*$ and $S_{i,h}^*$ can be measured by

$$\textbf{Oppo}(S_{i,j}^*, S_{i,h}^*) = \frac{1}{|S_{i,j}^*| \, |S_{i,h}^*|} \sum_{r \in S_{i,j}^*} \sum_{l \in S_{i,h}^*} A_{r,l}^-$$

We define the **Mean Average Cohesion (MAC)** as the mean of the average intra-subgraph cohesion for all k-OCGs in $\mathbb{S}$, that is,

$$\text{MAC} = \frac{1}{p} \sum_{i=1}^{p} \left(\frac{1}{k} \sum_{j=1}^{k} \textbf{Cohe}(S_{i,j}^*) \right)$$

Moreover, the **Mean Average Opposition (MAO)** is the mean of the average inter-subgraph opposition for all k-OCGs in $\mathbb{S}$, that is,

$$\text{MAO} = \frac{1}{p} \sum_{i=1}^{p} \left(\frac{1}{k(k-1)} \sum_{j \in [1,k]} \sum_{\substack{h \in [1,k] \\ h \neq j}} \textbf{Oppo}(S_{i,j}^*, S_{i,h}^*) \right)$$

Finally, we define **Harmonic Mean (HAM)** as

$$\text{HAM} = \frac{2 \times (\text{MAC} \cdot \text{MAO})}{\text{MAC} + \text{MAO}}$$

Mean Average Precision (MAP) is only used on the synthetic data set, where the vertex index set for each of the 20 subgraphs is known and used as ground truth. Denote such vertex index set of the j-th subgraph as $S_j^{gt}, j \in [1, 20]$, we measure the average precision of a single k-OCG $\mathcal{S}_i^*$ by

$$\textbf{Prec}(\mathcal{S}_i^*) = \frac{1}{20} \sum_{j=1}^{20} \frac{|S_{i,j}^* \cap S_j^{gt}|}{|S_{i,j}^*|}$$

and further evaluate MAP by

$$\text{MAP} = \frac{1}{p} \sum_{i=1}^{p} \textbf{Prec}(\mathcal{S}_i^*)$$

Since all the signed network partitioning methods (i.e., SNS, SNL, BNC and RA) partition the entire graph G into a single k-OCG, the size of $\mathbb{S}$ is $p = 1$ for those methods. For FOCG, the k-OCGs in $\mathbb{S}$ are obtained by selecting the top-p KKT points with large value of $F(X^*)$ from the set of KKT points $\hat{\psi}$. Thus, the size of $\mathbb{S}$ for FOCG is $p > 1$.

Using a small value of p (e.g., $p = 10$) FOCG returns an answer $\mathbb{S}$ that contains the top ten k-OCGs, which usually achieve very high MAC, MAO and HAM performance. However, for the fairness of the comparison, we set p to a large value so that $n_p = 0.5 \times n$, which forces FOCG to produce k-OCGs covering 50% of the vertices in graph G.

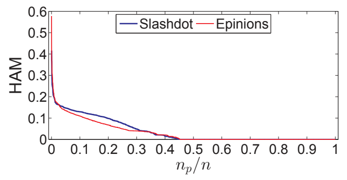


Figure 1: Ranked HAM performances on real world data sets. n is the number of vertices in G.

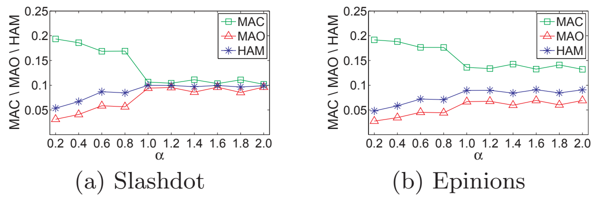


Figure 2: The effect of parameter α.

5.2 Effect of Parameters

We analyze the effect of parameters n_p and α (Equation 2) of FOCG. To analyze the effect of n_p, we sort all the KKT points X^* in $\hat{\psi}$ in descending order of $F(X^*)$, then evaluate the HAM value for each KKT point X^* by regarding it as the only k-OCG in $\mathbb{S}$. Figure 1 shows the results on the two real data sets with respect to the percentage of vertices in G that are covered by the top-p KKT points.

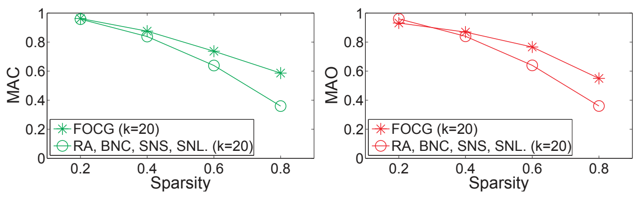
The KKT points ranked on the top (i.e., $\frac{n_p}{n} < 0.01$) achieve very high HAM on both data sets. The HAM decreases and approaches zero when $\frac{n_p}{n} \approx 0.45$. This indicates that about 45% of the vertices in graph G can form significant k-OCGs. Therefore, for the fairness of experiment, we evaluate the performance of FOCG by the average performance of all k-OCGs that cover 50% of the vertices of G, that is, setting $n_p = 0.5 \times n$.

Figure 2 shows the effect of parameter α on the performances of FOCG. In Equation 2, α controls the tradeoff between intra-subgraph cohesion and inter-subgraph opposition. A larger α results in a smaller intra-subgraph cohesion thus a lower MAC, and a larger inter-subgraph opposition thus a higher MAO. As shown in Figure 2, the larger α, the smaller MAC and larger MAO. However, when $\alpha > 1$, MAC, MAO and HAM all become stable. This indicates that the second term $\alpha g^-(\mathcal{S})$ of Equation 2 dominates $F(X)$ when $\alpha > 1$, thus the detected k-OCGs favor MAO most and do not change much when α increases further. HAM on both data sets becomes stable when $\alpha = 0.9$, thus we set $\alpha = 0.9$ as default in our experiments.

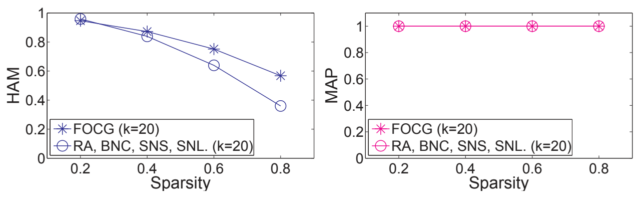
5.3 Effect of Network Sparsity

We analyze the effect of network sparsity using the synthetic data set. The *sparsity* of the signed network is defined as the percentage of zero entries in the adjacency matrix A.

A higher sparsity weakens both the cohesive and oppositive connections between graph vertices, thus decreases both the intra-subgraph cohesion and inter-subgraph opposition. Thus, in Figure 3(a)-(c), MAC, MAO and HAM all decrease when the sparsity increases. However, in Figure 3(d), MAP of all graph partitioning methods is not sensitive to sparsity. This is because the connectivity of the cohesive network and oppositive network is not affected too much by the sparsity.



(a) MAC performances (b) MAO performances



(c) HAM performances (d) MAP performances

Figure 3: The effect of network sparsity.

Thus, the partitioning methods can still accurately find the 20 subgraphs in the ground truth. FOCG achieves the same good performance in MAP as the graph partitioning methods, which means the k-OCGs detected by FOCG are also consistent with the ground truth.

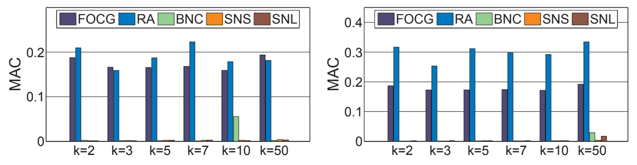
In Figure 3(a)-(c), when the sparsity is 0.2, SNS, SNL, BNC and RA achieve equivalently good performance as FOCG. This is because when the sparsity is small, the 20 subgraphs of the synthetic data set form a single 20-OCG. Thus, partitioning the entire graph into 20 subgraphs leads to the perfect result. However, when the sparsity increases, the original single k-OCG will be scattered into many small k-OCGs. In this case, partitioning the entire graph does not effectively obtain such small significant k-OCGs, thus the performance of the graph partitioning methods degrades quickly. Nevertheless, FOCG is able to accurately detect such small k-OCGs, thus achieves better performance under high sparsity. It is worth noting that real word signed network are often sparse. For example, the network sparsity of Slashdot and Epinions are both larger than 0.99.

5.4 Results on Real Data

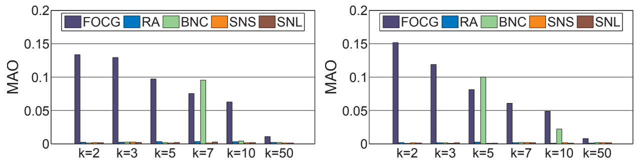
We compare the performance of all methods on Slashdot and Epinions. In such real world networks, a k-OCG represents k groups of people with strong intra-group cohesion and strong inter-group opposition. Apparently, the chance of finding $k = 50$ groups of people with strong inter-group opposition is much smaller than finding $k = 2$ groups of such people. Thus, it is more difficult to achieve good performance in MAO when k is large. As a result, in Figure 4(c)-(d), the MAO of FOCG decreases when k increases.

Since the real world networks are highly sparse, the entire network cannot form a single significant k-OCG. Instead, there are many small sized k-OCGs in different local regions. Since FOCG is designed to detect such small significant k-OCGs, it achieves much better HAM in Figure 4(e)-(f).

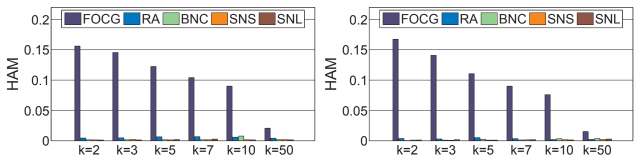
In Figure 4(a)-(b), FOCG is not always the best in MAC. In Figure 4(c)-(d), BNC outperforms FOCG in MAO when $k = 7$ and $k = 5$, respectively. The reason is that the k-OCGs in $\mathbb{S}$ of FOCG are selected according to the value of $F(X^*)$, which leads to a high HAM and a balanced performance of MAC and MAO. Since we are most interested in finding the significant k-OCG with both strong intra-subgraph cohesion and strong inter-subgraph opposition,



(a) MAC on Slashdot (b) MAC on Epinions

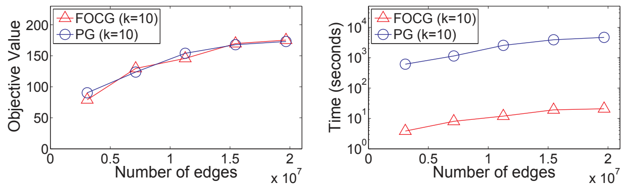


(c) MAO on Slashdot (d) MAO on Epinions



(e) HAM on Slashdot (f) HAM on Epinions

Figure 4: Performances on Slashdot and Epinions.



(a) Objective value (b) Running time

Figure 5: Scalability analysis on Douban data set.

a good HAM performance with balanced MAC and MAO guides the objective. Although FOCG may not achieve the best in MAC or MAO, its advantage on HAM is significant in Figure 4(e)-(f).

5.5 Scalability Analysis

We compare the scalability of FOCG and the Proximal Gradient method (PG) [18] on the Douban data set. We obtain four sub-networks from the Douban data set as follows. First, we start a breadth first search (BFS) from a randomly picked vertex on the cohesive network G^+ until the desired number of vertices are visited. Let S be the set of all vertices visited by the BFS. We use S to induce a cohesive sub-network G_S^+ and an oppositive sub-network G_S^-. Last, we obtain the signed sub-network G_S by merging the edge sets of G_S^+ and G_S^-. The number of vertices and that of edges of the 5 networks are listed in Table 2, where the 5-th network is simply the entire Douban data set.

On each of the 5 data sets, we run FOCG and PG 10 times and report the average results. In each run, we randomly initialize $X(0)$ by the same initialization method of PG, then run FOCG and PG using the same initialization.

Figure 5(a) shows the objective value $F(X^*)$ of the KKT point X^* detected by FOCG and PG. The objective values of FOCG and PG are close. Both FOCG and PG perform well in solving the optimization problem of Equation 2.

Table 2: The sampled Douban data sets.

Sample ID	1	2	3	4	5
vertices ($\times 10^3$)	31.8	79.4	135.0	238.3	1,588.5
Edges ($\times 10^6$)	3.1	7.1	11.3	15.5	19.7

Table 3: k-OCGs detected on WordNetAdj.

ID	Group 1	Group 2
1	outgoing, outer, external, outward	inner, internal, inward, interior
2	imprudent, improvident, short	long, prudent, farsighted, provident
3	descending, down, falling	ascending, rising, up
4	junior, insignificant, minor	leading, senior, better, major
5	noncurrent, backmost, back, , hindermost, rear	frontal, front, advanced, advancer, advance
6	unnecessary, excess, inessential, spare, extra	inevitable, essential, necessary
7	proud, stately, dignified, distinguished, courtly	silly, undignified, infra dig, demeaning, pathetic
8	active, alive, operational, existent	dormant, inactive, quiescent
9	inexperienced, unfledged, unfeathered	fledgling, full-fledged, fledged, feathered
10	involuntary, unwilled, unwilling	voluntary, ready, willing, inclined

Figure 5(b) shows the running time of FOCG and PG. The running time increases as the number of edges increases. FOCG is two orders of magnitudes faster. PG is a generic solution for constrained optimization problems, and is not specifically designed for k-OCG detection. It calculates the gradient of $F(X)$ in each iteration. The computational cost in calculating such gradients is very expensive when the number of edges is large. On the contrary, all FOCG iterations are efficiently performed on small subgraphs.

5.6 Case Study

We conduct a case study on the WordNetAdj data set, where each vertex represents an adjective, a cohesive edge indicates synonymous relationship and an opposite edge indicates antonymous relationship. In this network, a cohesive subgraph consists of a group of synonyms and a k-OCG is k groups of synonyms such that the adjectives in different groups are mostly antonymous with each other. Since the antonymous relationship between adjectives are usually bipolar, it is reasonable to set $k = 2$.

Table 3 shows the top-10 k-OCGs detected. Each row shows the two adjective groups of a detected k-OCG. This case study verifies that significant k-OCGs reveal interesting patterns in WordNetAdj.

6. CONCLUSIONS

In this paper, we tackled the novel problem of finding k-oppositive cohesive groups from signed networks. We formulated the k-OCG detection problem as a constrained quadratic optimization problem and designed FOCG, an effective and efficient algorithm. Our extensive experiments showed that FOCG can find interesting “gangs in war”, and is two orders of magnitudes faster than the traditional proximal gradient method. As future work, we will extend FOCG to automatically estimate the best value of k and effectively control the size of detected subgraphs.

7. REFERENCES

[1] R. Axelrod and D. S. Bennett. A landscape theory of aggregation. *BJPS*, 23(02):211–233, 1993.

[2] S. R. Bulò and I. M. Bomze. Infection and immunization: a new class of evolutionary game dynamics. *GEB*, 71(1):193–211, 2011.

[3] K.-Y. Chiang, *et al.* Prediction and clustering in signed networks: a local to global perspective. *JMLR*, 15(1):1177–1213, 2014.

[4] K.-Y. Chiang, *et al.* Scalable clustering of signed networks using balance normalized cut. In *CIKM*, 2012.

[5] L. Chu, *et al.* Alid: scalable dominant cluster detection. *PVLDB*, 8(8):826–837, 2015.

[6] P. Doreian and A. Mrvar. A partitioning approach to structural balance. *Social Networks*, 18(2):149–168, 1996.

[7] M. Gao, *et al.* On detecting maximal quasi antagonistic communities in signed graphs. *DMKD*, 30(1):99–146, 2016.

[8] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of Berkeley Symposium*, pages 481–492, 1951.

[9] J. Kunegis, *et al.* The slashdot zoo: mining a social network with negative edges. In *WWW*, pages 741–750, 2009.

[10] J. Kunegis, *et al.* Spectral analysis of signed graphs for clustering, prediction and visualization. In *SDM*, 2010.

[11] C. Liu, *et al.* A multiobjective evolutionary algorithm based on similarity for community detection from signed social networks. *Cybernetics, IEEE Trans. on*, 44(12):2274–2287, 2014.

[12] H. Liu, *et al.* Fast detection of dense subgraphs with iterative shrinking and expansion. *TPAMI*, 35(9):2131–2142, 2013.

[13] H. Liu and S. Yan. Robust graph mode seeking by graph shift. In *ICML*, pages 671–678, 2010.

[14] D. Lo, *et al.* Mining direct antagonistic communities in signed social networks. *IPM*, 49(4):773–791, 2013.

[15] D. Lo, *et al.* Mining direct antagonistic communities in explicit trust networks. In *CIKM*, 2011.

[16] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[17] T. S. Motzkin and E. G. Straus. Maxima for graphs and a new proof of a theorem of turán. *CJM*, 17(4):533–540, 1965.

[18] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2013.

[19] M. Pavan and M. Pelillo. Dominant sets and pairwise clustering. *TPAMI*, 29(1):167–172, 2007.

[20] Roulette-Selection. Wikipedia. https://en.wikipedia.org/wiki/Fitness_proportionate_selection.

[21] Source-Code. FOCG. https://github.com/lingyangchu/KOCG.SIGKDD2016.git.

[22] SNAP. https://snap.stanford.edu/data/.

[23] J. Tang, Y. Chang, C. Aggarwal, and H. Liu. A survey of signed network mining in social media. *arXiv*, 2015.

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

[25] C. Tsourakakis, *et al.* Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *SIGKDD*, pages 104–112, 2013.

[26] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[27] J. W. Weibull. *Evolutionary game theory*. MIT press, 1997.

[28] T. Xu, *et al.* Towards annotating media contents through social diffusion analysis. In *ICDM*, pages 1158–1163, 2012.

[29] K. Zhang, *et al.* Mining antagonistic communities from social networks. In *PAKDD*, pages 68–80, 2010.

[30] K. Zhang, *et al.* Mining indirect antagonistic communities from social interactions. *KAIS*, 35(3):553–583, 2013.

[31] Q. Zheng and D. Skillicorn. Spectral embedding of signed networks. In *SDM*, 2015.