

THE NATIONAL UNIVERSITY  
of SINGAPORE



School of Computing  
Computing 1, 13 Computing Drive, Singapore 117417

**TRB4/12**

***Large Scale Cohesive Subgraphs Discovery for  
Social Network Visual Analysis***

***Feng Zhao and Anthony K. H. Tung***

*April 2012*

# Technical Report

## Foreword

*This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.*

OOI Beng Chin  
Dean of School

# Large Scale Cohesive Subgraphs Discovery for Social Network Visual Analysis

Feng Zhao  
School of Computing  
National University of Singapore  
zhaofeng@comp.nus.edu.sg

Anthony K. H. Tung  
School of Computing  
National University of Singapore  
atung@comp.nus.edu.sg

## ABSTRACT

Graphs are widely used in large scale social network analysis nowadays. Not only analysts need to focus on cohesive subgraphs to study patterns among social actors, but also normal users are interested in discovering what happening in their neighborhood. However, effectively storing large scale social network and efficiently identifying cohesive subgraphs is challenging. In this work we introduce a novel subgraph concept to capture the cohesion in social interactions, and propose an I/O efficient approach to discover cohesive subgraphs.

Besides, we propose an analytic system which allows users to perform intuitive, visual browsing on large scale social networks. Our system stores the network as a social graph in the graph database, retrieves a local cohesive subgraph based on the input keywords, and then visualizes the subgraph out on orbital layout, in which more important social actors are located in the center. By summarizing textual interactions between social actors as tag cloud, we provide a way to quickly locate active social communities and their interactions in a unified view.

## 1. INTRODUCTION

Graphs play a seminal role in social network analysis nowadays. A large and rapidly growing social network companies store social data as graph structures, such as Facebook<sup>1</sup> and Twitter<sup>2</sup>. In a social graph, vertices represent social actors, while edges represent relationships or interactions between actors. One fundamental operation on social graph is to identify groups of social actors that are highly connected with each other, represented by a cohesive subgraph [5, 26], in which analysts may discover interesting structural patterns among social actors, and normal users can know what happening in their neighborhood.

<sup>1</sup><https://www.facebook.com>

<sup>2</sup><https://www.twitter.com>

Cohesive subgraph discovery is an intriguing problem and has been widely studied for decades. One fundamental structure is the clique in which every pair of vertices is connected. Finding cliques is NP-Hard [11] and many recent work try to relax the clique problem to improve scalability [18, 2, 22, 5, 26]. However, these methods do not directly take the characteristics of social network analysis into consideration. Take the graph in Figure 1a as an example. If we set  $k$  equal to 3, all the solid edges and connected vertices form a 3-core subgraph  $g$  since every vertex  $v$  inside it satisfies  $d(v) \geq 3$ . However,  $g$  is not cohesive enough as a whole. Considering cliques inside  $g$ , we can find a 5-clique  $(a, b, c, d, f)$  and a 4-clique  $(c, d, e, f)$  on the left, as well as two 4-cliques  $\{(m, n, p, q), (p, q, t, u)\}$  on the right. In other word, vertex  $a$  and  $p$  are not tightly coupled with each other and the subgraph  $g$  is better viewed as two separated cohesive groups.

This phenomenon, denoted as the tie strength concept, is well studied in the sociological area. Mark Granovetter in his landmark paper [16] indicates that two actors  $A$  and  $B$  are likely to have many friends in common if they have a strong tie. In another state-of-the-art sociological paper, White et al. [27] observes that a group is cohesive to the extent that pairs of its members have multiple social connections, direct or indirect, but within the group, that pull it together. One intuitive real life example is that you and your intimate friends in Facebook may have high possibility to share lots of mutual friends. However, this observation has been missing from many of the cohesive subgraph definitions, which drives us to define a “mutual-friend” structure to capture the tie strength in a quantitative manner for social network analysis. In particular, one can perceive the intuition about social structures: if two actors are strongly tied, it is likely that they also share ties to others. Thus, for a tie between two actors  $A$  and  $B$ , we consider it legitimate only if supported by at least  $k$  other actors who are both tied to  $A$  and to  $B$ . For instance, in Figure 1, the red edge  $(a, p)$  is less important than other solid edges since it is only supported by one triangle  $(a, j, p)$ . We will formally define the problem and compare it to other definitions in details in the subsequent section.

How to improve the scalability is one potential challenge of cohesive subgraph discovery for social network analysis. Most of the existing approaches [25, 26, 28] mainly focus on the dense region recognition for moderate size graphs. However, many practical social network applications need to store the large scale graph in disks or databases. Like Facebook, over 800 million active actors use its service per month

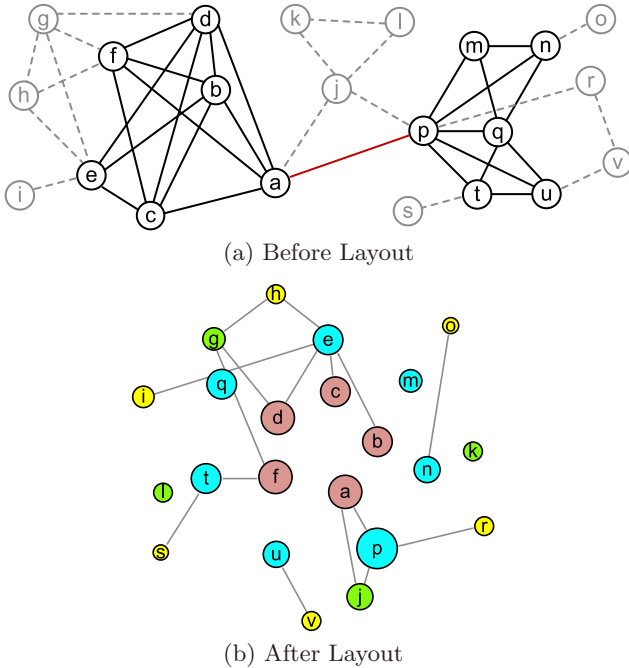


Figure 1: Cohesive Graph Example

all over the world [4], which is impossible to store in memory. Therefore, besides providing memory based solutions, we focus on developing a solution to handling large scale social graphs stored in a graph database, which is more scalable for graph operations than a relational database. Like Twitter, recently it migrated its social graph to FlockDB [1], a distributed, fault-tolerant graph database for managing data at webscale. By leveraging graph databases, we extend memory based algorithms to I/O efficient solutions for large scale social networks.

Additionally, exploring and analyzing social network can be time consuming and not user-friendly. Visual representation of social networks is important for understanding the network data and conveying the result of the analysis. However, it is a challenge to summarize the structural patterns as well as the content information to help users analyze the social network. One previous work [25] proposes a novel linear plot for graph structure, which sketches out the distribution of dense regions and is suitable for static dense pattern discovery. Unlike this work, our system insulates users from the complexities of social analysis by visualizing cohesive subgraphs and the contents in a unified view. For graph structure, we propose an orbital layout to clearly display it with respect to the cohesive value, in which more important social actors are located in the center. Figure 1b shows an orbital layout for the graph in Figure 1a. Briefly speaking, this layout consists of four orbits in different colors representing vertices with four cohesive values. The more cohesive vertices are located closer to the center. Like the 5-clique  $(a, b, c, d, f)$ , all five vertices are in the innermost orbit. As for vertices size setting, ordering and edge filtering, we will explain them in details later in this paper. For the contents, we make use of tag cloud technique to summarize the major semantics for a group of social actors. Generally speaking, our visualization is flexible and can be easily applied to other cohesive graph concepts.

In this paper, we develop a novel social network visual

analytic framework for large scale cohesive subgraphs discovery. Our contributions are summarized as follows:

- We have introduced a novel cohesive subgraph concept to capture the intrinsic feature of social network analysis nicely.
- By leveraging graph databases, we have devised an offline algorithm to compute global cohesive subgraphs efficiently. Moreover, we have developed an online algorithm to further refine local cohesive subgraphs based on the results of offline computations.
- We have developed an orbital layout to decompose the cohesive subgraph into a set of orbits, and coupled with tag cloud summarization, which allows users to locate important actors and their interactions inside subgraphs clearly.
- We have conducted extensive experiments, and the results show that our approach is both effective and efficient.

The rest of the paper is organized as follows. Section 2 defines the cohesive subgraph discovery problem handled throughout this paper. Section 3 introduces the overview of our system. Section 4 presents the offline computations in the graph database, and the online visual analysis is described in Section 5. Our extensive experimental study is reported in Section 6. Section 7 reviews the related literature on cohesive subgraph finding and social network analysis. Section 8 concludes the paper.

## 2. PROBLEM DEFINITION

In this section, we first introduce the preliminary knowledge, then define the maximal  $k$ -mutual-friend finding problem, and show several important properties about this concept. To clarify the novelty of our definition, we further compare it with the clique,  $k$ -core as well as  $DN$ -Graph [26] in depth.

### 2.1 Preliminaries

As stated in Section 1, we model a social network as an undirected, simple **social graph**  $G(V, E)$  in which vertices represent social actors and edges represent interactions between actors. The  $k$ -mutual-friend subgraph proposed in this paper is derived from clique and  $k$ -core [22]. Clique is a fully connected subgraph, in which every pair of vertices is connected by an edge. If the size of a clique is  $c$ , we call the clique a  $c$ -clique.  $k$ -core is one successful degree relaxation of clique concept defined as follows.

DEFINITION 2.1. (*k-core Subgraph*)

A  $k$ -core is a connected subgraph  $g$  such that each vertex  $v$  has degree  $d(v) \geq k$  within the subgraph  $g$ .

The  $k$ -core is motivated by the property that every vertex has degree  $d(v) = c - 1$  in a  $c$ -clique.  $k$ -core also needs to satisfy the degree condition, but the restriction on subgraph size is not required. As such,  $k$ -core can be efficiently computed in  $O(|E|)$  time complexity [22]. Differently, based on the observation in Section 1, we propose the  $k$ -mutual-friend subgraph to emphasize on tie strength. One important property about edges in clique is that every edge is supported by  $tr(e) = c - 2$  triangles in a  $c$ -clique. Analogous to the  $k$ -core

definition, the  $k$ -mutual-friend sets a lower bound for every edge’s triangle count. Next we will formally define the  $k$ -mutual-friend and show its relationships to other cohesive structures.

## 2.2 The $k$ -mutual-friend Subgraph

DEFINITION 2.2. ( *$k$ -mutual-friend Subgraph*)

A  $k$ -mutual-friend is a connected subgraph  $g \in G$  such that each edge is supported by at least  $k$  pairs of edges forming a triangle with that edge within  $g$ . The  $k$ -mutual-friend number of this subgraph, denoted as  $\mathcal{M}(g)$ , equals  $k$ .

Note that we need to exclude the trivial situation to consider an isolated vertex as a mutual-friend. Given the parameter  $k$ , we may discover many  $k$ -mutual-friend subgraphs that overlap with each other. In the worst case, the number of  $k$ -mutual-friend subgraphs can be exponential to the graph size. Therefore, we further define the maximal  $k$ -mutual-friend subgraph to avoid redundancy.

DEFINITION 2.3. (*Maximal  $k$ -mutual-friend Subgraph*)

A maximal  $k$ -mutual-friend subgraph is a  $k$ -mutual-friend subgraph that is not a proper subgraph of any other  $k$ -mutual-friend subgraph.

To compare with clique and core, we present two interesting properties about the  $k$ -mutual-friend subgraph.

PROPERTY 2.1. *Every  $(k + 2)$ -clique in  $G$  is contained in a  $k$ -mutual-friend of  $G$ .*

PROOF. *Since a  $(k + 2)$ -clique is a fully connected subgraph with order  $k + 2$ , each edge is supported by  $k$  triangles. Therefore, it is contained in a  $k$ -mutual-friend subgraph by Definition 2.2.  $\square$*

PROPERTY 2.2. *Every  $k$ -mutual-friend of  $G$  is a subgraph of a  $(k + 1)$ -core of  $G$ .*

PROOF. *Since every vertex  $v$  connects to at least one edge,  $v$  is adjacent to at least  $k + 1$  vertices in  $g_k$  because every edge is supported by at least  $k$  triangles in  $g_k$ . So  $g_k$  qualifies as a  $(k + 1)$ -core of  $G$ .  $\square$*

The above two properties indicate that the mutual-friend is in place between the clique, which is too restrictive, and the core, which is too lax. Note that the reverse of the above two properties are not true. Again, we take Figure 1 as an example. The 4-clique  $(m, n, p, q)$  is a subgraph of the 2-mutual-friend  $(m, n, p, q, t, u)$ , while 2-mutual-friend  $(a, b, c, d, e, f)$  and  $(m, n, p, q, t, u)$ , both of them are contained in the 3-core  $(a, b, c, d, e, f, m, n, p, q, t, u)$ . Finally, we define the main problem we investigate in this paper as follows.

PROBLEM 1. (*Maximal  $k$ -mutual-friend Subgraph Finding*) *Given a social graph  $G(V, E)$  and the parameter  $k$ , find all the maximal  $k$ -mutual-friend subgraphs.*

Finally, we summarize the notations to illustrate our idea throughout this paper in Table 1.

### 2.2.1 Comparison to DN-Graph

Before we illustrate the solution to Problem 1, we further state an interesting connection between the mutual-friend concept and the DN-Graph concept proposed by Wang et al. [26] recently. A DN-Graph, denoted by  $G'(V', E', \lambda)$ , is a connected subgraph  $G'(V', E')$  of graph  $G(V, E)$  that satisfies the following two conditions: (1) Every connected pair of vertices in  $G'$  shares at least  $\lambda$  common neighbors. (2) For any  $v \in V \setminus V'$ ,  $\lambda(V' \cup \{v\}) < \lambda$ ; and for any  $v \in V'$ ,  $\lambda(V' - \{v\}) \leq \lambda$ .

At the first glance, DN-graph is similar to the maximal  $k$ -mutual-friend subgraph. However, these two concepts are distinct due to the second condition in DN-Graph definition. Intuitively, the DN-graph defines a strict condition that the maximal subgraphs need to reach the local maximum even for adding or deleting only one vertex. On the other hand, the maximal  $k$ -mutual-friend defines the local maximal subgraph that is not a proper subgraph of any other  $k$ -mutual-friend subgraph. As demonstrated in Figure 1a,  $(m, n, p, q)$ ,  $(p, q, t, u)$  and  $(m, n, p, q, t, u)$  are all DN-Graphs with  $\lambda = 2$ , since the  $\lambda$  value can only decrease if adding or removing any vertices. However, only  $(m, n, p, q, t, u)$  is the maximal 2-mutual-friend since other two are its subgraphs. This example shows that the DN-Graph finding may generate many redundant subgraphs. Furthermore, due to the hardness of satisfying the second condition, solving the DN-Graph problem is NP-Complete as proven by the authors. To solve it they iteratively refine the upper bound for each edge to approach the real value. Although this is a proper method to handle the DN-Graph problem, it is hard to scale to large scale graph. By avoiding the iterative method, we will show that the Problem 1 can be efficiently solved in polynomial time subsequently. We also make it scalable by means of storing and processing the social graph in a graph database.

Table 1: Notations

Notation	Description
$G(V, E)$	A social graph with vertices $V$ and edges $E$
$g$	A subgraph of the social graph
$g_k$	A $k$ -mutual-friend subgraph
$d(v)$	The degree of vertex $v$
$N(v)$	The neighbor vertices of vertex $v$
$tr(e)$	The triangle count of edge $e$
$\mathcal{M}(g)$	The mutual-friend number of subgraph $g$
$\mathcal{M}(e)$	$\max_{e \in g}(\mathcal{M}(g))$
$\mathcal{M}(v)$	$\max_{v \in g}(\mathcal{M}(g))$

## 3. SYSTEM ARCHITECTURE

In Figure 2, we provide a client-server architecture to solve Problem 1 and support social network visual analysis. The solution can be divided into offline computations and online visual analysis. As for offline computations, we discover the maximal  $k$ -mutual-friend subgraphs based on social graph stored in graph database, and index labels of vertices for keyword query.

For online analysis, we retrieve a local subgraph  $g$  close to the user selected vertex on top of offline computing result, online compute the exact  $\mathcal{M}$  values for graph elements inside  $g$ , and generate the orbital layout for visualization. Moreover, we select representative tags to summarize the textual information in the local graph. In the client side,

user can search and browse the visualized subgraph. The elaboration will be presented in the following two sections.

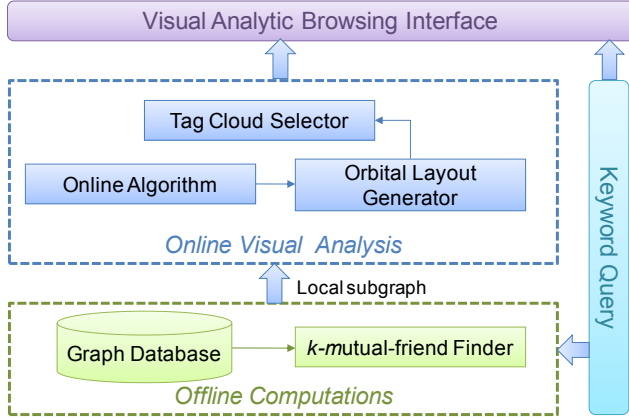


Figure 2: Social Network Visual Analytic System

## 4. OFFLINE COMPUTATIONS

In this section, we first propose memory based solutions to solve Problem 1 in polynomial time, and then leverage the graph database to extend the solution for large scale social network analysis.

### 4.1 Memory based Solution

Given a social graph  $G$  and the parameter  $k$ , the intuitive idea of discovering the maximal  $k$ -mutual-friend is to remove the unsatisfied vertices and edges from  $G$ , resulting in a naive solution in Algorithm 1. Based on the Definition 2.2, we iteratively remove edges that are not contained in  $k$  triangles until all of them satisfy the condition  $Tr(e) \geq k$ . The procedure is illustrated in Example 1.

**EXAMPLE 1.** *Considering a maximal  $k$ -mutual-friend finding with  $k = 2$  over the graph in Figure 3a, the left part of Figure 1a. The procedure needs three loops in Algorithm 1. First, edges  $\{(e, i), (e, h), (e, g), (f, h)\}$  are removed since their triangle counts are less than 2. Next,  $\{(d, g), (f, g), (g, h)\}$  are further removed since their triangle counts become less than 2, while  $e(d, e)$  is still part of the 2-mutual-friend due to  $Tr(e(d, e)) = 2$ . In the third loop,  $Tr(e(d, f))$  reduces to 3 but still satisfies the condition. Because all the remaining edges with triangle counts larger than or equal to 2, the graph remains unchanged and the loop terminates. Lastly, we delete all the isolated vertices and obtain 2-mutual-friend  $(a, b, c, d, e, f)$  as in Figure 3b.*

Although this is a straight forward solution, the computational complexity is relatively high. One fundamental operator is to compute the triangle count for an edge. For  $e(u, v)$ , this operator takes  $d(u) + d(v)$  cost. One while loop from line 2 to 8 can remove at least one edge from  $G$ , otherwise the graph remains unchanged and the algorithm terminates. Therefore, in the worst case the algorithm removes one edge at a time and needs  $|E|$  times loops to remove all the edges from  $G$ . As such, the total complexity is  $|E| \times \sum_{e(u, v) \in G} (d(u) + d(v))$ . This expression can be further simplified to the order of  $|E| \times \sum_{v \in G} d(v)^2$ , because we need to get the  $v$ 's neighbors  $d(v)$  times in one while loop. For practical case, we seldom encounter this extreme situation,

### Algorithm 1: Naive Memory $k$ -mutual-friend

---

**Input:** Social graph  $G(V, E)$  and parameter  $k$   
**Output:**  $k$ -mutual-friend subgraphs

```

1 bool isChanged = true
2 while isChanged == true do
3   isChanged = false
4   foreach  $e(u, v) \in G$  do
5     compute  $Tr(e)$  based on  $N(u), N(v)$ 
6     if  $Tr(e) < k$  then
7       remove  $e$  from  $G$ 
8       isChanged = true
  // remove isolated vertices
9 foreach  $v \in G$  do
10  if  $d(v) == 0$  then remove  $v$  from  $G$ 
11 return  $G$ 

```

---

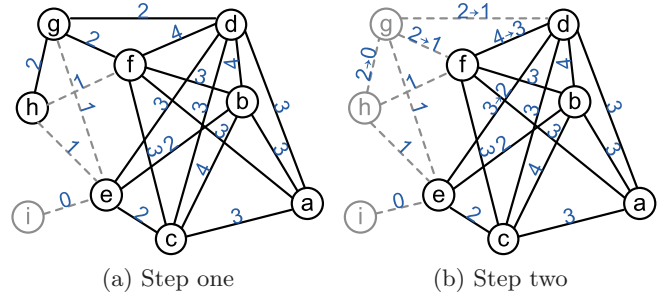


Figure 3: Example of in Memory Algorithm

but a large number of iterations is still a bottleneck of this solution.

We next prove the correctness of Algorithm 1 in two aspects. On one hand, the remaining vertices and edges are part of the maximal- $k$ -mutual-friend subgraphs. This aspect is true according to the definition of  $k$ -mutual-friend subgraph. On the other hand, the removed vertices and edges are not part of the maximal- $k$ -mutual-friend subgraphs. Because the only modification on  $G$  is the removal of edges, bringing about the decrease of triangle counts, the edges supported by less than  $k$  triangles can be safely deleted since they cannot be part of a  $k$ -mutual-friend subgraph any more.

Algorithm 1 is inefficient because it has lots of unnecessary triangle computations. When an edge is deleted, it only changes the triangle counts of the edges which are forming triangles with that edge. Thus we can obtain edges affected by the deleted edge and only recompute triangle counts for them. This intuition is reflected in Algorithm 2, which can be divided into three steps. First, one necessary condition for  $Tr(e(u, v)) \geq k$  is  $d(u) \geq k + 1$  and  $d(v) \geq k + 1$  as in the proof of Property 2.2. This is a lightweight method of dispatching many vertices and their adjacent edges before doing the harder work of eliminating edges with insufficient triangles. The remaining graph is then processed by the second step, which costs most of the workload to remove edges not supported by at least  $k$  triangles. From line 6 to 9, we first check all the edges' triangle counts. The  $Q$  is implemented as a hash set to record non-redundant removed edge elements. Next, instead of computing the triangle on all the edges to check the stability of the graph, we iteratively retrieve the affected edges from  $Q$  until  $Q$  is empty.

This is the indicator that the graph becomes unchanged. Finally, the removal of inadequate edges likely results in isolated vertices, which are removed in the last step. We show the procedure on the running example as follows.

EXAMPLE 2. We consider a maximal 2-mutual-friend finding in Figure 3a again based on Algorithm 2. According to the degree condition, we first remove vertex  $i$  and the edge  $(e, i)$  since the degree of  $i$  is less than 3. We then check the edge’s triangle counts and delete  $\{(e, g), (e, h), (f, h)\}$ . Moreover, we record these edges in  $Q$  for affected edges. Edges  $\{(d, g), (f, g), (g, h)\}$  are further removed until  $Q$  is empty. Finally, we delete all the isolated vertices and generate the same result as in Example 1.

---

**Algorithm 2: Improved  $k$ -mutual-friend**

---

```

Input: Social graph  $G(V, E)$  and parameter  $k$ 
Output:  $k$ -mutual-friend subgraphs
// filter by degree of vertices
1 foreach  $v \in V$  do
2   if  $d(v) < k + 1$  then
3      $\lfloor$  remove  $v$  and related  $e$  from  $G$ 
// remove edges with insufficient triangles
4 initialize a queue  $Q$  to record removed edges
5 initialize a hash table  $Tr$  to record triangle counts
6 foreach  $e = (u, v) \in E$  do
7   compute  $Tr(e)$  based on  $N(u), N(v)$ 
8   if  $Tr(e) < k$  then
9      $\lfloor$  enqueue  $e$  to  $Q$ 
10 while  $H \neq \emptyset$  do
11   dequeue  $e$  from  $Q$ 
12   find out edges  $E'$  forming triangles with  $e$ 
13   remove  $e$  from  $G$ 
14   foreach  $e' \in E'$  do
15      $Tr(e') - -$ 
16     if  $Tr(e') < k$  then
17        $\lfloor$  enqueue  $e'$  to  $Q$ 
// remove isolated vertices
18 foreach  $v \in G$  do
19    $\lfloor$  if  $d(v) == 0$  then remove  $v$  from  $G$ 
20 return  $G$ 

```

---

The proof of the correctness of Algorithm 2 is similar to Algorithm 1. In essence, because the algorithm only reduces the triangle counts by means of edge deletion, it can discover all the maximum  $k$ -mutual-friend. As for complexity analysis, the improved algorithm outperforms the naive one remarkably because it avoids a great deal of unnecessary triangle computations. The first step takes  $O(|V|)$  complexity to check vertices’ degree. The second step dominates the whole procedure. The initial triangle counting is the same as one while loop in Algorithm 1, which has time complexity  $\sum_{v \in G} d(v)^2$ . From line 10 to 17, finding all the edges forming triangles with the current edge  $e(u, v)$  takes  $d(u) + d(v)$  work. In the worst case, all the edges are removed from  $Q$ . Since  $Q$  only stores each edge one time, the total cost is  $\sum_{e(u, v) \in G} (d(u) + d(v))$ , equal to  $\sum_{v \in G} d(v)^2$ . The last step also takes  $O(|V|)$  complexity to remove isolated vertices. As a whole, the total time complexity is  $O(\sum_{v \in G} d(v)^2)$ . It not

only avoids the unnecessary iterations, but also reduces the graph size with relative small effort in the first step. Although the above algorithm is efficient, but is not suitable for large scale graph processing stored in disk. Retrospect the algorithm, it needs  $O(|E|)$  space complexity, which is too large to store in memory. So we extend it to the disk based solution in the following section.

## 4.2 Solution in Graph Database

In this section, we first introduce the concept of graph database, and then present a streaming solution in graph database and improve it by means of partitioning.

### 4.2.1 The graph database

A graph database [21] represents vertices and edges as a graph structure instead of storing data in separated tables. It is designed specifically for graph operations. To this end, a graph database provides index-free adjacency that every vertex and edge has a direct reference to its adjacent vertex or edge. Since it is a single data structure, elements (i.e. vertices and edges) are already “joined” among each other. As such, this data model becomes difficult to scale beyond a single machine and still maintain the speed of a traversal across partition borders.

Despite this drawback the model possesses a significant advantage: there is a near constant time cost for graph traversal, referring to visiting adjacent elements in a graph in some algorithmic fashion. This is actually a primitive operation in graph-like queries or algorithms, naturally suitable for shortest path finding, maximal connected subgraph problem and graph’s diameter computations and so on. Furthermore, it can scale more naturally to large data sets as they do not typically require expensive join operations. We define the traversal operator as  $traverse(elem, step)$  for better demonstration, which means that the length of shortest paths from graph element  $elem$  to the satisfied results cannot be larger than  $step$ . For example,  $traverse(u, 1)$  retrieves all the vertices that are directly connected to  $u$  and the edges among them. Example 3 shows a comparison between graph database and relational database.

EXAMPLE 3. Consider the triangle counting in the memory based algorithm. Given  $e(u, v)$ , one basic operation is to fetch  $N(u)$  and  $N(v)$ . If the social graph is stored as one vertex table and one edge table in relational database (assume the related index has been built), we need two steps to obtain  $N(u)$ . First, we utilize  $u.id$  to query the edge table index with  $O(\log |V|)$  I/O cost. Second, since  $v$  has  $d(v)$  connected vertices, we further need to query the vertex table index to get the related attributes with  $O(d(v) \cdot \log |V|)$  I/O cost. This procedure can be largely improved in graph database. Both steps can finish with  $O(d(v))$  I/O cost, which is invariant to the graph size.

Later in this section, we make use of the traversal operator to extend the in memory algorithm to I/O-efficient algorithms in a graph database. For implementation, we utilize the Neo4j<sup>3</sup> graph database. Note that we could easily migrate our algorithms to other popular graph databases as long as they support traversal, such as DEX<sup>4</sup>, OrientDB<sup>5</sup> and so forth.

<sup>3</sup><http://neo4j.org>

<sup>4</sup><http://www.sparsity-technologies.com/dex>

<sup>5</sup><http://www.orienttechnologies.com>

### 4.2.2 Streaming based solution

The streaming based solution is modified from Algorithm 2 and implemented in the graph database. The major changes are two-fold. On one hand, we use graph traversal to access vertices and edges (line 1 and 3), as well as compute triangle counts (line 5 and 6). On the other hand, we build index on edge attributes to mark edges as deleted (line 7, 9 and 15) and record edges' triangle counts (line 8, 13 and 14). Note that the edge attributes are in the order of  $O(|E|)$ , so they still need to be maintained out of core for large graph datasets. In this way, we make full use of the graph database, and keep all the advantages in the improved memory algorithm.

---

#### Algorithm 3: Streaming based Algorithm

---

**Input:** Social graph  $G(V, E)$  and parameter  $k$   
**Output:**  $k$ -mutual-friend subgraphs  
*// filter by degree of vertices*  
1 traverse the vertices of  $G$   
2 remove  $v$  and related edges if  $d(v) < k + 1$   
*// remove edges with insufficient triangles*  
3 traverse the edges  $E$  of  $G$   
4 **foreach**  $e = (u, v) \in E$  **do**  
5      $N(u) \leftarrow \text{traverse}(u, 1); N(v) \leftarrow \text{traverse}(v, 1)$   
6     compute  $tr(e)$  according to  $N(u), N(v)$   
7     **if**  $tr(e) < k$  **then** mark  $e$  as deleted  
8     **else** set  $e$ 's mutual number attribute as  $tr(e)$   
9 **while** exist edges  $e(u, v)$  marked as deleted **do**  
10    $E' \leftarrow$  edges form triangles with  $e$  in  $\text{traverse}(e, 1)$   
11   remove  $e$  from  $G$   
12   **foreach**  $e' \in E'$  **do**  
13      $tr(e') \leftarrow$   
14     **if**  $tr(e') < k$  **then**  
15       mark  $e'$  as deleted  
16 remove isolated vertices from  $G$   
17 **return**  $G$

---

We next analyze the I/O cost in this algorithm. Filtering by degree and removing isolated vertices need  $O(|E|)$  I/O. The most costly part is removing edges with insufficient triangles. For edge  $(u, v)$ , finding triangle count takes  $O(d(u) + d(v))$  I/O work. Similar to the analysis for memory based algorithm, each edge can only be marked as deleted once. We conclude that this step needs  $O(\sum_{v \in G} d(v)^2)$  I/O cost, which is also the total order of I/O consumptions. Besides, the traversal on vertices and edges is dominated by sequential I/O, which further reduces the I/O cost.

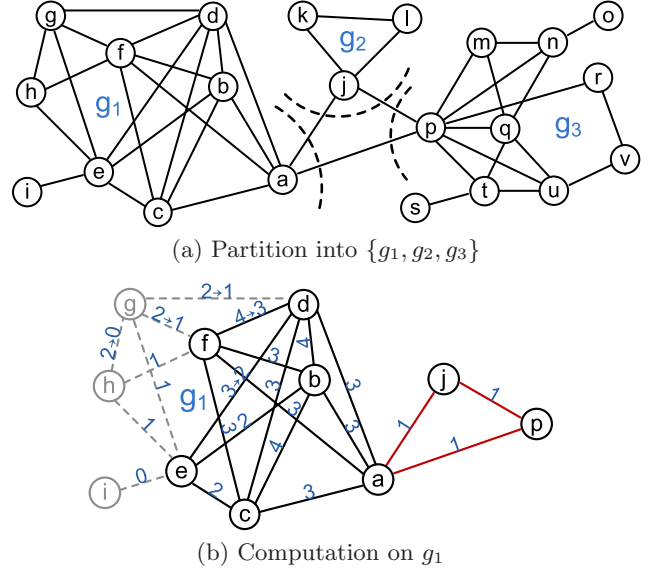
### 4.2.3 Partition based solution

Since all the triangle computations are directly operated in graph database, the streaming algorithm fails to make full use of the memory. Therefore, we proposed an improved approach based on the graph partitioning, and load partitions into memory to perform in memory triangle computations to save I/O cost and improve efficiency. To begin with, we derive a greedy based partitioning method in Algorithm 4 from the heuristics in paper [24]. The basic idea is to streamingly process the graph and then assign every vertex to the partition where it has the largest number of edges connecting to. As in line 11 in Algorithm 4, *localPartitionNum* records

the number of edges in each partition,  $(1 - |g_i| \times p / |G|)$  suggests that partitions with larger size have smaller weight, and the product of the above two factors decides which partition the current vertex belongs to. This algorithm, requiring one breadth first graph traversal, is efficient with linear I/O complexity. However, the resulting partitions cannot be directly used because this algorithm is a vertex partitioning. Typically, it only extends partitions by including all the vertices connecting to the vertices inside the partition, which may result in the loss of triangles. As in Figure 4a, the running example is partitioned into three parts  $\{g_1, g_2, g_3\}$ . In this case, the triangle  $(a, j, p)$  is missing since its vertices are separated into three partitions. In order to keep all the triangles, we define an induced subgraph as in Definition 4.1.

#### DEFINITION 4.1. (Induced Subgraph)

Denote  $g_i+ = (V_i+, E_i+)$  as an induced subgraph of a partition  $g_i(V_i, E_i)$  of  $G$ . The extended vertex set is defined as  $V_i+ = V_i \cup \{v : u \in V_i, v \in V \setminus V_i, (u, v) \in E\}$ . The extended edge set is defined as  $E_i+ = \{(u, v) : (u, v) \in E, u \in V_i, (u, v), (u, w) \in E, v.\text{partition} \neq w.\text{partition}, u.\text{id} < v.\text{id}, u.\text{id} < w.\text{id}\}$ .



**Figure 4: Example of Partition based Algorithm**

Based on the induced subgraph, the triangle  $(a, j, p)$  in Figure 4a is allocated in  $g_1$  as shown in Figure 4b, because  $id_a$  is smaller than  $j, p$  in this triangle. Next we formally prove the correctness of the partitioning method in Lemma 4.1.

**LEMMA 4.1.** *Induced subgraphs  $\{g_1, \dots, g_p\}$  derived from  $p$  partitions of  $G$  have the same set of triangles as  $G$ .*

**PROOF.** *The lemma is equivalent to the statement that every triangle  $(u, v, w)$  in  $G$  appears once and only once in all partitions. The proof can be divided into three cases. If three vertices belong to  $V_i$  of partition  $i$ , the triangle can only be inside the same partition. If any two of three vertices belong to  $V_i$  of partition  $i$ , without loss of generality, we assume that  $u, v \in V_i$  and  $w \in V_j$ . The triangle is in partition  $i$  but not in partition  $j$ , since  $(u, v)$  can only be assigned to partition  $i$ . If three vertices are located in different partitions,*

we assign the triangle to the vertex with smallest id as defined in  $\Delta E_i$ , so this triangle only appears once in induced subgraphs.  $\square$

---

**Algorithm 4: Graph Partitioning**


---

**Input:** Social graph  $G(V, E)$ , partition number  $p$   
**Output:**  $\{g_1, \dots, g_p\}$  partitions

```

1 foreach  $v \in G$  in BFS order do
2   if  $d(v) < k + 1$  then
3      $\lfloor$  remove  $v$  and related edges; continue
4   initialize the array localPartitionNum with size  $p$ 
5    $N(v) \leftarrow \text{traverse}(v, 1)$ ; foreach  $u \in N(v)$  do
6      $ind \leftarrow u$ 's partition index
7     if  $ind > 0$  then localPartitionNum[ $ind$ ] $++$ 
8    $maxWeight \leftarrow 0$ ;  $curWeight \leftarrow 0$ 
9    $pIndex \leftarrow -1$ 
10  for  $i$  from 1 to  $p$  do
11     $curWeight \leftarrow$ 
12       $localPartitionNum[i] \times (1 - |g_i| \times p / |G|)$ 
13    if  $curWeight > maxWeight$  then
14       $maxWeight \leftarrow curWeight$ 
15       $pIndex \leftarrow i$ 
16  set  $v$ 's partition index as  $pIndex$ 
17 return  $G$ 

```

---

Finally, we provide a partition based solution in Algorithm 5. First we partition the graph into  $p$  partitions, and for each partition, we do the in memory edge removal. Note that we only consider inside edges, which only affect triangles satisfying  $\{(u, v, w), u, v, w \in V_i\}$ . As such, we make use of the memory to reduce the graph size as well as keeping the correctness of the solution. After this, we write the induced subgraphs back to graph database and use Algorithm 3 to do post processing. We take the induced subgraph  $g_1$  in figure 4b to find 2-mutual-friend subgraph. Note that edges  $\{(a, j), (a, p), (j, p)\}$  are outside edges, while others are inside edges. For inside edges, we directly apply in memory algorithm and remove edges in dotted lines with triangle counts less than 2. But for outside edges, we cannot delete them since they may affect triangle counts in other partitions. After we deal with all the partitions, we post process the refined graph using Algorithm 3 to obtain the final result. In the worst case, this algorithm has the same I/O complexity as Algorithm 3. But in practice, it loads and processes the induced subgraphs to memory and avoids many disk triangle computations. The detailed comparison between this two disk-based solutions will be presented in the experimental section.

## 5. ONLINE VISUAL ANALYSIS

To support online visual analysis, we implement a visual interactive system accessible on the Web<sup>6</sup>, and provide a use case on Twitter dataset in Figure 6 to illustrate our idea.

### 5.1 Online Algorithm

Based on the offline computations, we retrieve a local subgraph associated with the input keywords from graph

<sup>6</sup><http://db128gb-b.ddns.comp.nus.edu.sg:8080/vis/demo>

---

**Algorithm 5: Partition based Algorithm**


---

**Input:** Social graph  $G(V, E)$ , parameter  $k$ , and partition number  $p$   
**Output:**  $k$ -mutual-friend subgraphs

```

1 partition the graph based on Algorithm 4
2 for  $i$  from 1 to  $p$  do
3   load induced subgraph  $g_i$  into memory from the partition  $i$ 
4   // Do in memory edge removal
5   queue  $Q \leftarrow \emptyset$ 
6   hash table  $Tr \leftarrow \emptyset$ 
7   foreach  $e = (u, v) \in E_i + \wedge e$  is inside do
8     compute  $Tr(e)$  based on  $N(u), N(v)$ 
9     if  $Tr(e) < k$  then
10      enqueue  $e$  to  $Q$ 
11  repeatedly remove inside edges until  $Q$  is empty
12  write  $g_i$  back to the graph database
13 use Algorithm 3 to do post processing
14 return  $G$ 

```

---

database and compute exact  $\mathcal{M}$  values for every edge and vertex inside the subgraph. This is a fundamental step to support graph layout later in this section. User can select a focused vertex  $v$  from a list of vertices containing the keywords, and our system will return a local subgraph including all the vertices within the distance  $\tau$  from  $v$  and the edges among these vertices, i.e.  $\text{traverse}(v, \tau)$ . For efficient online computation, we show one important stability property of the  $k$ -mutual-friend subgraph as follows.

PROPERTY 5.1. *The  $k$ -mutual-friend's output does not change arbitrarily with the output size, i.e.  $g_{k+1} \subseteq g_k$ .*

For every edge  $e$  in subgraph  $g_{k+1}$ ,  $Tr(e) \geq k + 1 > k$  suggests that this subgraph is also a  $g_k$ . Therefore, based on the stability property, if one wants to compute the exact  $\mathcal{M}$  values for graph elements, we can make use of the offline result as input, with much less work than computing from scratch. Furthermore, the offline computations provide a useful upper bound for online computations.

LEMMA 5.1. *Given  $G(V, E)$  after offline computation, the edges from the online local subgraph  $g \subseteq G$  satisfy  $\{\mathcal{M}_g(e) \leq Tr_g(e) \leq Tr_G(e), e \in g\}$ .*

PROOF. *Since  $g$  is a subset of  $G$ , for every edge  $e \in g$ , its local triangle count should be smaller or equal to the global triangle count, i.e.  $Tr_g(e) \leq Tr_G(e)$ . Based on the definition of  $k$ -mutual-friend subgraph, the local triangle count bounds the  $\mathcal{M}_g$  value. All in all, we obtain the relationship  $\mathcal{M}_g(e) \leq Tr_g(e) \leq Tr_G(e)$ .  $\square$*

We implement Algorithm 6 based on the above observations. The first step is to retrieve the local subgraph within the distance  $\tau$  to  $v$ . Then, we iteratively compute the exact  $g_m$  from  $m = \mathcal{M}_{min}$  to  $m = \mathcal{M}_{max}$ . Finally, we merge all the  $g_m$  to obtain the local subgraphs with exact  $\mathcal{M}$  values. To illustrate, we retrieve a local subgraph by  $\text{traverse}(a, 2)$  from the graph in Figure 1, and the result local graph is shown in Figure 5a. The number shows the triangle counts computed by the offline algorithm, which are the upper bound for the exact  $\mathcal{M}$  values. Vertices  $\{k, l, j\}$  and edges

in dotted lines are immediately removed since their triangle counts are smaller than 2. In the first loop, we remove vertex  $g$  and edges  $e(d, g), e(f, g)$  because their  $\mathcal{M}$  values become one in the local graph. The rest of the graph is the 2-mutual-friend. In Figure 5b, we use the similar procedure to find 3-mutual-friend from the 2-mutual-friend, which includes vertices  $\{a, b, c, d, f\}$  and edges connecting them. The algorithm terminates since the  $\mathcal{M}_{max}$  is updated to the current largest triangle count equal to three. For the case in Figure 6, we first retrieve a local subgraph by a  $\tau = 2$  traversal with respect to a Twitter actor “The White House”, and find its  $\mathcal{M}$  values from 3 to 7 using the online algorithm.

---

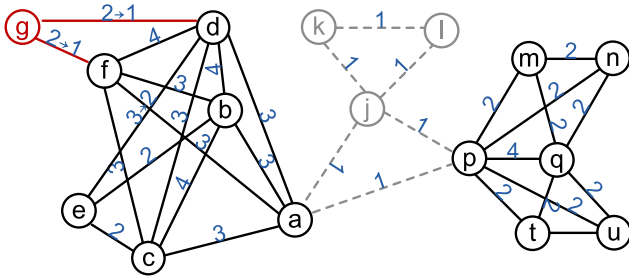
**Algorithm 6: Online Algorithm**


---

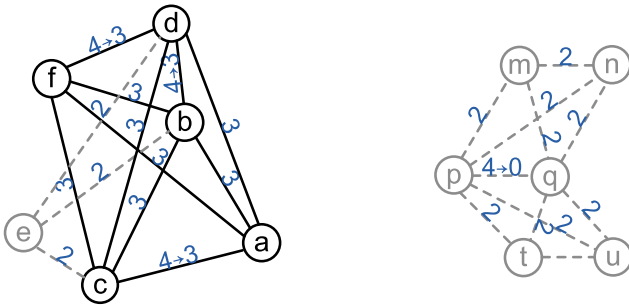
**Input:**  $G(V, E)$ ,  $k$ , vertex  $v$ , and distance threshold  $\tau$   
**Output:** Local subgraphs with exact  $\mathcal{M}$  values

- 1  $g \leftarrow \text{traverse}(v, \tau)$
- 2  $\mathcal{M}_{max} \leftarrow \max\{Tr_G(e) : e \in g\}$
- 3  $\mathcal{M}_{min} \leftarrow k$
- 4 **for**  $m$  **from**  $\mathcal{M}_{min}$  **to**  $\mathcal{M}_{max}$  **do**
- 5     compute  $m$ -mutual-friend and update  $g$  by Algorithm 2
- 6      $g_m \leftarrow \{e : e \in g, Tr(e) = m\}$
- 7      $\mathcal{M}_{max} \leftarrow \max\{Tr_g(e) : e \in g\}$
- 8 **return**  $g_{\mathcal{M}_{min}} \cup \dots \cup g_{\mathcal{M}_{max}}$

---



(a)  $\text{traverse}(a, 2)$  to 2-mutual-friend



(b) 2-mutual-friend to 3-mutual-friend

**Figure 5: Example of Online Computation**

## 5.2 Orbital Layout

Based on the online algorithm results, we expect to visualize the local subgraph reflecting the characteristics of the  $k$ -mutual-friend. However, how to display graph elements with different  $\mathcal{M}$  values is a challenge. Furthermore, because the cohesive local subgraph includes a great deal of edges, minimizing the visual complexity for better demonstration could be another challenge. We next demonstrate a novel layout to solve the above challenges.

Intuitively, a good layout for  $k$ -mutual-friend needs to distinguish graph elements with different  $\mathcal{M}$  values and emphasize elements with higher  $\mathcal{M}$  values since they represent more important information. With this observation we propose a layout with a set of concentric orbits. Vertices with higher  $\mathcal{M}$  values are located close to the center, while vertices with lower  $\mathcal{M}$  values are placed on orbits further away from the center. Since the layout is analogous to the planetary orbits, it is called orbital layout as depicted in Figure 6. Note that orbital layout is different from radial layout [6]. Although vertices are located in concentric circles, the radial layout positions a focused vertex in the center and other vertices according to their distances to the focus vertex. Differently, we put each vertex  $v$  in an orbit according to  $\mathcal{M}(v)$  without a special vertex in the center. Furthermore, we set different colors to distinguish vertices in different orbits. Figure 6 consists of five orbits in different colors representing vertices with five  $\mathcal{M}$  values from 7 inside to 3 outside. Moreover, the size of vertices is proportional to vertex degree to reflect the importance of vertices. For instance, vertex with name “The White House” has the largest degree so that it has the biggest size.

In order to make the graph layout clear and reduce the visual complexity, the challenge of crossing edges must be overcome. To this end, only edges in between two orbits are allowed to display. Since vertices in the same orbit probably connect with each other, displaying these edges can hardly provide extra information. Furthermore, we carefully order vertices to minimize crossing edges. From inner orbit to outer orbit, we put a vertex in the position that makes it close to connected vertices in the inner orbits. For example, vertex with name “Barack Obama” in Figure 6 is located in the left since most of its neighbors are near this region.

## 5.3 Representative Tag Cloud Selection

Besides structure visualization, another dimension of social network analysis is to understand the interactions among social actors, which come from, for instance, the newfeeds from Facebook or tweets from Twitter. Since users may select a group of social actors with a great number of textual contents, we incorporate the tag cloud approach to summarizing various topics inside it. A potential challenge is how to select the most important tags to capture the major interests of these actors. Moreover, for distinct topics, the challenge might be how to discover a set of tags so that they could be comprehensive enough to cover different interests inside the same group.

To tackle these challenges, we compute a score for each tag by multiplying two factors, the significance and diversity. On the one hand the significance measure guarantees the truly popular tags can be selected, and on other hand the diversity measure captures various rather than only similar topics. In our implementation, we adopt the TF-IDF approach for significance and the semantic distance in WordNet [7] for diversity. In representative tag selection, we first generate top  $N$  frequent words to form a candidate set, and filter out infrequent words to improve the efficiency. Then, we utilize a greedy strategy that iteratively moves tags with the largest score from the candidate set to the representative set until the number of selected tags reaches  $n$ ,  $n < N$ , a user adjustable parameter. As such, we discover representative tags summarizing the interactions inside the local subgraph. Users can quickly select and browse preferred

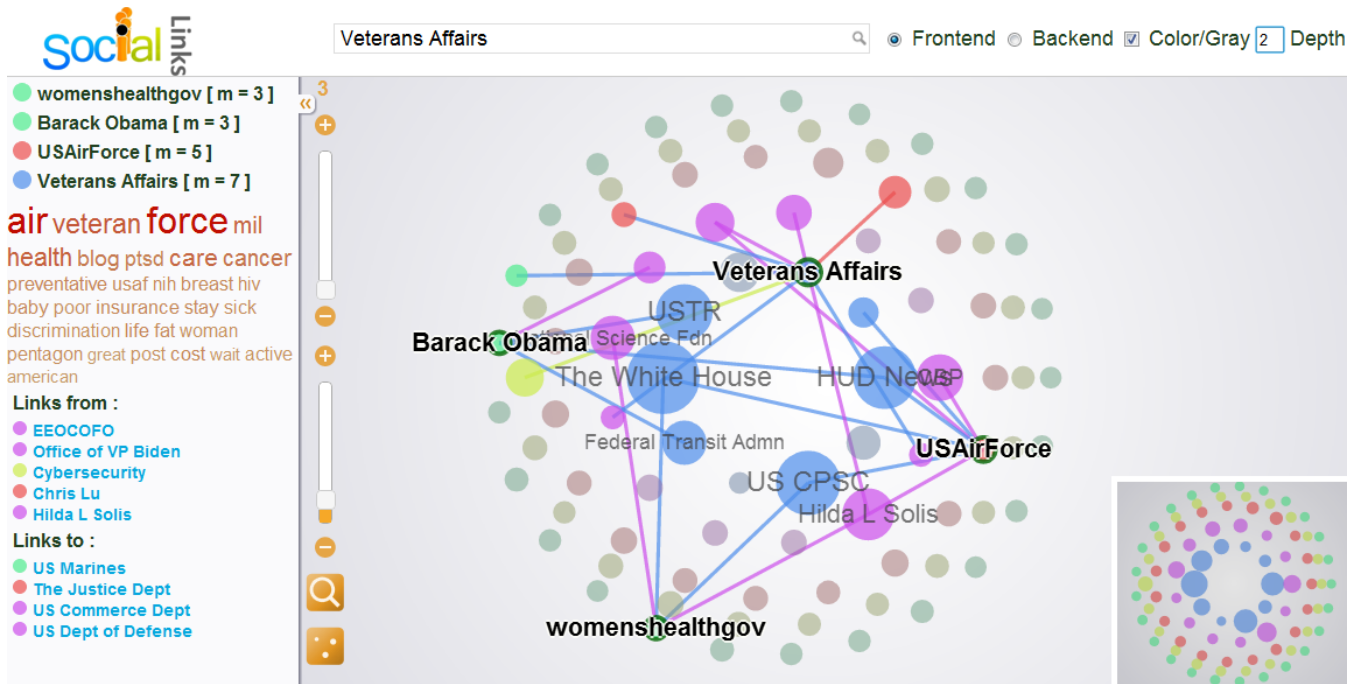


Figure 6: Visual Analysis Interface

subgroup of actors to explore what activities they are involved in, or what topics they are taking about, etc. Finally, based on the real use case on Twitter social graph, we illustrate the functionalities and the advantages of our visual analytic browsing interface.

## 5.4 Case Study

The visual interface in Figure 6 consists of three parts, i.e. search input area on the top, information summarization in the left column, and subgraph visualization in the main frame. After users input keywords in search box and select a focused vertex matching the keywords, our system visualizes the local subgraph in the main frame, so that users can select vertices they are interested in with the summarization in the left column. Without loss of generality, this example shows the result in Twitter dataset with respect to the keyword “white house”, in which vertices represent twitter actors and edges represent the “following” relationships. The depth, equivalent to the distance threshold, is set to 2.

With the help of online algorithm and layout generation, we dramatically reduce the visual complexity in the main frame. The visible subgraph only contains 89 vertices and 527 edges, which is much smaller the initial local subgraph with 2006 vertices and 2838 edges. As a result, we could quickly perceive that the networking of “The White House” is dominated by various US departments and government officials, which is unlikely to obtain from thousands of vertices with messy information. Furthermore, users can highlight several vertices and their neighbors while other vertices and edges become transparent. Considering in some cases subgraphs are quite large, users can use frontend search to locate preferred vertices within the current subgraph, or adjust the  $\mathcal{M}$  value lower bound to filter out unsatisfied graph elements using the slide bar at the top left corner. Moreover, we support zoom in/out function to focus on part of

the graph and users can view the sketch of the whole subgraph with a thumbnail at the bottom right corner.

The left column displays the  $\mathcal{M}$  values of the highlighted vertices, the corresponding tag cloud as well as the link information for the last selected vertex “Veterans Affairs”. The tag cloud is a helpful tool that summarizes the most significant and diverse topics those twitter actors are interested in. In this example, we select 30 representative tags out of 100 candidates, where “Veterans Affairs” may show great concern about the PTSD (Post Traumatic Stress Disorder) and discrimination problems while “womenshealthgov” mainly focuses on topics like health, breast cancer and baby. In order to know the source of these tags, hovering over specific tag in the tag cloud will trigger the source vertices being highlighted. If we point to the “insurance” tag, the Twitter actor “Barack Obama” will be highlighted indicating that he pays close attention to the insurance issue.

## 6. EXPERIMENTS

We present experimental studies to evaluate our social network visual analysis system in this section. For simplification, we refer to Algorithm 1 as *mNaive*, Algorithm 2 as *mImproved*, while refer to Algorithm 3 as *dStream*, Algorithm 5 as *dPartition*. The *mOnline* is short for the online algorithm. We implement these algorithm in Java language and evaluate on the Windows operating system with Quad-Core AMD Opteron(tm) processor 8356 and 128GB RAM.

We compare our solutions on a great deal of real social network datasets described in Table 2, most of which are collected from the Stanford Network Analysis Project’s website<sup>7</sup>. The datasets are sorted in increasing order of edge number. We utilize moderate size datasets to compare in memory algorithms, while use large size datasets to compare algorithms in graph database. Moreover, Twitter and

<sup>7</sup><http://snap.stanford.edu/>

DBLP datasets are selected for online visual analysis since they contain rich textual information.

Table 2: Dataset Statistics

Dataset	Vertex Num	Edge Num	Type
Epinions	75,879	405,741	memory
Twitter	452,710	813,964	memory
DBLP	916,978	3,063,286	memory
Flickr	1,715,255	22,613,982	disk
FriendFeed	653,646	27,811,816	disk
Facebook	72,661,577	160,975,756	disk

## 6.1 Offline Computations Evaluation

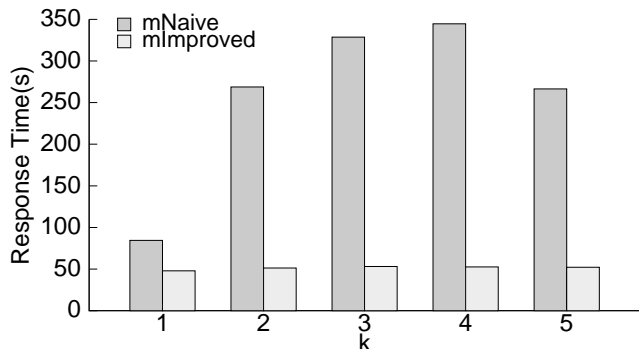
### 6.1.1 Memory based Algorithms

We compare *mNaive* and *mImproved* algorithms on three datasets and results are summarized in Figure 7. This figure depicts the effect of  $k$  on the response time of three datasets. For Epinions and DBLP datasets, *mImproved* outperforms *mNaive* evidently, while their performances on Twitter dataset are in the same level. This is because Twitter dataset having average degree less than 2 is much more sparse than the other two datasets. Therefore, even the naive algorithm can reach the stable state very fast without incurring a great deal of unnecessary triangle computations. For other two datasets, *mImproved* is about one order faster than *mNaive* averagely.

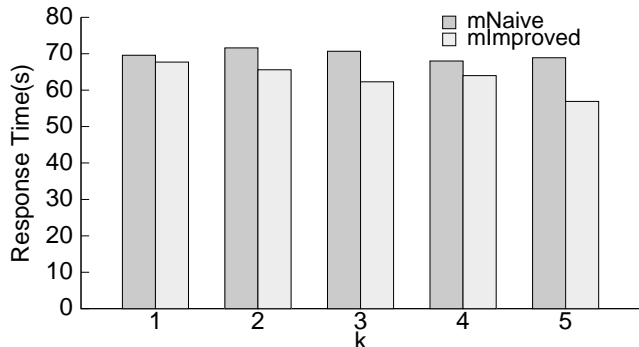
One interesting observation is that the response time is not quite related to  $k$ , but mainly determined by the triangle computing times in each algorithm. As in the first two rows in Table 3, the triangle computing times for Epinions dataset in *mNaive* is about ten times of that in *mImproved*, which is close to the ratio of response time. Thus, the result again justifies our conclusion in Section 4.1 that *mImproved* outperforms *mNaive* mainly because it largely reduces the amount of triangle computations. More specifically, when  $k = 1$ , because we only remove edges not in any triangles without affecting other edges, *mNaive* can finish in two iterations (make sure that the graph is unchanged in the second iteration), and *mImproved* only needs one iteration. The response time for *mNaive* decreases when  $k$  equals to 5 since the number of triangle computations drops to  $2,439k$ , smaller than the number when  $k$  equals to 3 and 4. The triangle computing times for DBLP dataset in the last two rows in Table 3 have the similar pattern. For Twitter dataset, both algorithms need the number of triangle computations in the same level, which determines that their response time also close to each other. To sum up, *mImproved* is much faster than *mNaive* mainly because it reduces the number of triangle computations, especially when the graph is dense.

Table 3: Triangle Computing Times

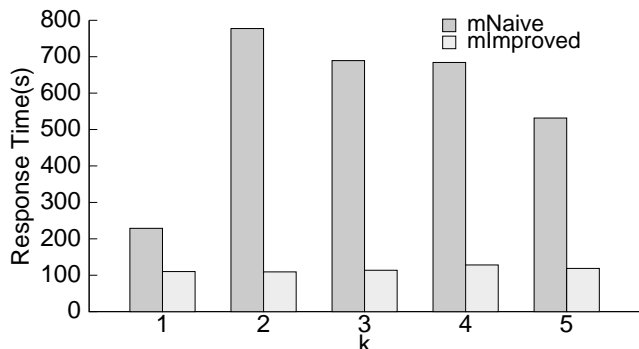
	1	2	3	4	5
<i>mNaive</i>	717k	2,219k	2,840k	3,088k	2,439k
<i>mImproved</i>	130k	202k	249k	284k	311k
<i>mNaive</i>	1,097k	1,261k	1,324k	1,364k	1,391k
<i>mImproved</i>	873k	867k	836k	819k	817k
<i>mNaive</i>	5,950k	24,767k	22,950k	25,166k	21,085k
<i>mImproved</i>	288k	1,028k	1,921k	2,671k	3,240k



(a) Epinions



(b) Twitter



(c) DBLP

Figure 7: Comparison of Memory Algorithms

### 6.1.2 Disk based Algorithms

Next we evaluate the disk based algorithms with three large scale datasets. For partition based algorithm, we control the usage of memory by only allowing to store a sub-graph with at most 1GB size. As such, we can estimate the number of partitions  $p$  for each dataset according to the graph size in graph database as in Table 4. Since the response time is not determined by  $k$ , we set  $k$  as 3 to compare the performance of two disk based algorithms. The results in Figure 8 depicts the response time for the three datasets with two parts: I/O time and CPU time. All in all, the partition based algorithm is about five times faster than the streaming based algorithm, and the response times for both of them are increasing with respect to the increase of graph size. In particular, *dStream* algorithm is dominated by the I/O time, while *dPartition* is dominated by the CPU time, in accord with our analysis in Section 4.

In essence, the major difference between *dStream* and

$dPartition$  is the cost for triangle computations. As shown in Table 5, the average cost for triangle computations in  $dPartition$  is only one tenth of that in  $dStream$ , because most of the triangle computations in the former approach are in memory while all the triangle computations in the later one are in graph database. Comparing three datasets, the average triangle computing time for Facebook is the fastest for both algorithms due to the smallest average degree of Facebook. As a result, although the number of edges in Facebook is much larger than that in FriendFeed, the response time of Facebook is slightly larger than that of FriendFeed. Moreover, Table 6 summarizes the percentages of the partitioning part and the computing part for  $dPartition$  algorithm. Because the partitioning algorithm reads the input graph only once and writes the partitions back to graph database, the partitioning part costs small amount of time comparing to the computing part.

Table 4: Number of Partitions in Algorithm 5

	Flickr	FriendFeed	Facebook
Size(GB)	1.57	1.92	11.6
$p$	2	2	12

In conclusion,  $dPartition$  trades off a lightweight graph partitioning for fast triangle computing in memory. The result verifies our claim in Section 4 that the partition based algorithm is I/O-efficient in practice.

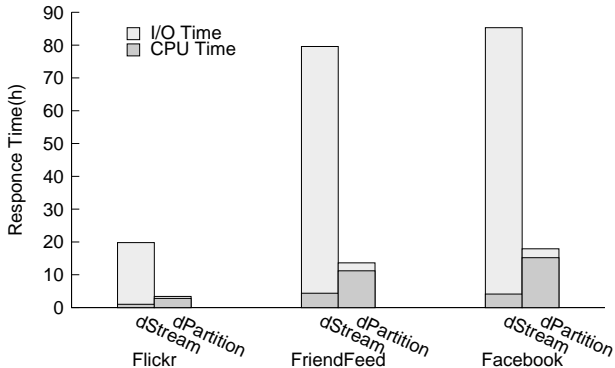


Figure 8: Comparison of Disk Algorithms

Table 5: 10k Times Triangle Computing Cost

Dataset	$dStream$	$dPartition$
Flickr	122.1s	11.3s
FriendFeed	349.6s	33.5s
Facebook	12.9s	1.3s

Table 6: Percentages of Response Time

	Flickr	FriendFeed	Facebook
Partitioning part	9.1%	10.5%	13.2%
Computing part	90.9%	89.5%	86.8%

## 6.2 Online Analysis Evaluation

By randomly selecting 10 focused vertices on Twitter and DBLP datasets respectively, we obtain the average performance of online analysis with three components:  $mOnline$  algorithm, orbital layout generation and tag cloud selection. All the experiments are based on the 3-mutual-friend graph calculated by the offline solution. For tag cloud selection, we obtain 20 representative tags out of 100 candidates from the text in focused vertices. The major objective is to test whether our system can well support online analysis.

Table 7 shows the efficiency measures by varying the distance threshold  $\tau$  from 1 to 3. It is clear that the total response time has an ascending trend with the increase of  $\tau$  for both datasets. Taken separately, the costs of online algorithm and the layout generation are largely increasing with respect to  $\tau$ . The major reason is that the response time for the first two components is proportional to the number of edges, which increases obviously with respect to  $\tau$ , as in the bottom row of Table 7. However, the speed of tag cloud selection remains stable since it is only affected by the textual content in the focused vertex. Comparing the difference between two datasets, the tag cloud selection for Twitter is much slower because the number of words in tweets is large than that in paper title.

Moreover, the average edge number suggests that distance threshold  $\tau = 2$  is a practical setting for online analysis, which can generate local subgraph with reasonable size. Note that we don't consider network transmission time since it is unstable and highly affected by the network condition, which is not the focus of this evaluation. In summary, the whole analytical procedure can be finished less than three second so that it is acceptable for online interactive applications.

Table 7: Average Response Time(in ms)

	distance threshold $\tau$					
	Twitter			DBLP		
Component	1	2	3	1	2	3
OnlineAlgo	1	32	563	2	16	498
Layout	2	6	138	2	5	108
TagCloud	1986	1726	1829	164	176	189
Avg edge num	2	368	9856	22	348	7727

## 7. RELATED WORK

Modeling a cohesive subgraph mathematically has been extensively studied for decades. One of the earliest graph models was the clique model [19], in which there exists an edge between any two vertices. However, the clique model idealizes cohesive properties so that it seldom exists and hard to compute. Alternative approaches are suggested that essentially relaxes the clique definition in different aspects. Luce [18] introduces a distance based model called  $k$ -clique and Alba [2] introduces a diameter based model called  $k$ -club. Generally speaking, these models relax the reachability among vertices from 1 to  $k$ . Another line of work focuses on a degree based model, like  $k$ -plex [23] and  $k$ -core [22]. The  $k$ -plex is still NP-Complete since it restricts the subgraph size, while  $k$ -core further relaxes it to achieve the linear time complexity with respect to the number of edges. In this paper, we follow a new direction based on the edge triangle model, which is more suitable for social network analysis since it captures the tie strength between actors inside the subgroup. Recently, database researchers try to scale up the disk based cohesive subgraph discovery. Cheng et al. [8] propose a partition based solution for massive  $k$ -core mining. They also develop a disk based triangulation method [9] as a fundamental operation for cohesive subgraph discovery. Differently, we store the social graph in graph database that is more scalable for graph traversal based algorithms.

Besides, social network has been investigated in many other different contexts. One related research direction is community detection. Leskovec et al. [17] summarize the

state-of-the-art community detection methods and compare them empirically. Typical approach of community detection is to choose an objective function that captures a set of vertices with better internal connectivity than external connectivity, such as betweenness centrality [14] and modularity [20]. Another is the tie strength theory and applications. Mark Granovetter introduces the tie strength concept in his landmark paper [16]. Recently, many social network researchers investigate this important theory in online social network, such as the user behaviors in Facebook [13, 4] and Twitter [15]. Their conclusions show that the strength of tie is still a tenable theory in social media, which are the bases of the mutual-friend subgraph definition in this paper.

Social network visualization and analysis has received a great deal of attention from both sociology and computer science communities. Freeman [12] summarizes the use of graphic imaging in social network analysis from the sociology perspective. Research from the computer science perspective puts more efforts into the graph representation and exploration of social networks. Wang et al. [25] proposes a linear plot based on graph traversal to capture the dense subgraph distribution in the whole graph. Zhang et al. [28] extends it to compare the pattern changing between two graph snapshots. Place vertices in concentric circles with different levels is a popular way to visualize graph structures, such as  $k$  shell decomposition [3], centralities visualization [10] and so on. We leverage the circular idea and devise the orbital layout to visualize  $k$ -mutual-friend subgraphs in an interactive manner. Note that the orbital layout is perpendicular to linear plot. We could seamlessly integrate the linear plot for global subgraph distribution and the orbital layout for local subgraph representation.

## 8. CONCLUSIONS

In this paper, we have introduced a novel framework that integrates the cohesive subgraphs discovery with the visual social network analysis. Unlike previous works, we proposed a new cohesive subgraph definition called  $k$ -mutual-friend to take the tie strength into consideration. Moreover, a memory based solution is proposed and extended to the scalable solution in the graph database. To further consolidate this interesting framework, we provided a visual analytic browsing interface that helps navigate users in searching and browsing the graph structure as well as semantics. The outcomes from an experimental study demonstrated that our solution is both efficient and effective.

As for future research, we expect to extend our framework for other graph based analytic applications, such as protein-protein interaction analysis, RDF graph analysis etc. By interacting with the visualized interface, users could explore and perceive interesting patterns in a more intuitive way. Another challenge direction is to maintain the cohesive subgraphs with frequently updates. As such, we shall provide a real time analytic toolkit to monitor everyone's evolving social network. Finally, how to extend the partition based graph processing to distributed environment is probably another interesting area of future work.

## 9. REFERENCES

- [1] <http://en.wikipedia.org/wiki/FlockDB>.
- [2] R. D. Alba. A graph-theoretic definition of a sociometric clique. *Journal of Mathematical Sociology*, pages 113–126, 1973.
- [3] J. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the  $k$ -core decomposition. *NIPS*, pages 41–50, 2006.
- [4] E. Bakshy, I. Rosenn, C. Marlow, and L. Adamic. The role of social networks in information diffusion. In *WWW*, 2012.
- [5] B. Balasundaram, S. Butenko, and I. V. Hicks. Clique relaxations in social network analysis: The maximum  $k$ -plex problem. *Oper. Res.*, pages 133–142, 2011.
- [6] U. Brandes and C. Pich. More flexible radial layout. *J. Graph Algorithms Appl.*, pages 107–118, 2011.
- [7] A. Budanitsky and G. Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources*, 2001.
- [8] J. Cheng, Y. Ke, S. Chu, and M. Ozsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.
- [9] S. Chu and J. Cheng. Triangle listing in massive networks and its applications. In *SIGKDD*, pages 672–680, 2011.
- [10] C. Correa, T. Crnovrsanin, and K. Ma. Visual reasoning about social networks using centrality sensitivities. *TVCG*, pages 1–15, 2010.
- [11] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost  $np$ -complete. In *FOCS*, pages 2–12, 1991.
- [12] L. Freeman. Visualizing social networks. *Journal of social structure*, 2000.
- [13] E. Gilbert and K. Karahalios. Predicting tie strength with social media. In *CHI*, pages 211–220, 2009.
- [14] M. Girvan and M. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, pages 7821–7826, 2002.
- [15] P. A. Grabowicz, J. J. Ramasco, E. Moro, J. M. Pujol, and V. M. Eguluz. Social features of online networks: the strength of weak ties in online social media. *CoRR*, 2011.
- [16] M. Granovetter. The strength of weak ties. *American journal of sociology*, pages 1360–1380, 1973.
- [17] J. Leskovec, K. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640, 2010.
- [18] R. Luce. Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, pages 169–190, 1950.
- [19] R. Luce and A. Perry. A method of matrix analysis of group structure. *Psychometrika*, pages 95–116, 1949.
- [20] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, pages 413–421, 2004.
- [21] M. Rodriguez and P. Neubauer. The graph traversal pattern. In *Graph Data Management*, pages 29–46, 2011.
- [22] S. Seidman. Network structure and minimum degree. *Social networks*, pages 269–287, 1983.
- [23] S. Seidman and B. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, pages 139–154, 1978.
- [24] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs. In *WWW*, 2012.
- [25] N. Wang, S. Parthasarathy, K. Tan, and A. Tung. Csv: visualizing and mining cohesive subgraphs. In *SIGMOD*, pages 445–458, 2008.
- [26] N. Wang, J. Zhang, K. Tan, and A. Tung. On triangulation-based dense neighborhood graph discovery. In *VLDB*, pages 58–68, 2010.
- [27] D. White and F. Harary. The cohesiveness of blocks in social networks: Node connectivity and conditional density. *Sociological Methodology*, pages 305–359, 2001.
- [28] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle  $k$ -core motifs within networks. In *ICDE*, 2011.