Community Mining on Dynamic Weighted Directed Graphs*

Dongsheng Duan Yuhua Li† Yanan Jin Zhengding Lu
Inteligent and Distributed Computing Lab, Huazhong University of Sci. & Tech.
duandongsheng@gmail.com idcliyuhua@hust.edu.cn
jin.yan.an@smail.hust.edu.cn zdlu@hust.edu.cn

ABSTRACT

This paper focuses on community mining including community discovery and change-point detection on dynamic weighted directed graphs(DWDG). Real networks such as e-mail, co-author and financial networks can be modeled as DWDG. Community mining on DWDG has not been studied thoroughly, although that on static(or dynamic undirected unweighted) graphs has been exploited extensively. In this paper, *Stream-Group* is proposed to solve community mining on DWDG. For community discovery, a two-step approach is presented to discover the community structure of a weighted directed graph(WDG) in one time-slice: (1)The first step constructs compact communities according to each node's single compactness which indicates the degree of a node belonging to a community in terms of the graph's relevance matrix; (2)The second step merges compact communities along the direction of maximum increment of the modularity. For change-point detection, a measure of the similarity between partitions is presented to determine whether a change-point appears along the time axis and an incremental algorithm is presented to update the partition of a graph segment when adding a new arriving graph into the graph segment. The effectiveness and efficiency of our algorithms are validated by experiments on both synthetic and real networks. Results show that our algorithms have a good trade-off between the effectiveness and efficiency in discovering communities and change-points.

Categories and Subject Descriptors

H.2.8 [**Database applications**]: Data mining

General Terms

Algorithm

*This work was supported by National Natural Science Foundation of China under Grant 70771043 and 60873225.
†Yuhua Li is the correspondence author. Email: idcliyuhua@hust.edu.cn

CNIKM'09, November 6, 2009, Hong Kong, China.

Keywords

dynamic weighted directed graph, community discovery, change-point detection, modularity, compactness

1. INTRODUCTION

Because of extensive application domains, community mining on graphs has attracted more and more attentions in recent years[8, 19, 26, 25, 23, 9, 28, 16]. Community mining can help identify the overall structure of networks and discover the latent rule of community evolution. Many real networks can be modeled as dynamic weighted directed graphs (DWDG), such as e-mail, co-author and financial networks. Although community mining on static(or dynamic unweighted undirected) graphs has been exploited extensively[23], that on DWDG has not been studied thoroughly.

There are roughly two classes of community mining on graphs. One is static community detection and another is community mining on dynamic graphs.

For community detection, there are many prevailing algorithms, including METIS[12], spectral partitioning [1], hierarchical clustering[13, 10, 17, 20], Information-theory based algorithm [6, 22] and Markov clustering[29] and so on. In addition, Du et al.[8] proposed an algorithm called ComTector which is based on computing the cliques of graphs. Pizzuti et al.[19] proposed a genetic algorithm based community detection algorithm. But the existing algorithms have one or more drawbacks: needing user-specified number k of communities, considering no directions or weights, inefficient and so on.

In recent years, the dynamic characteristics of real networks begin to be studied. Leskovec et al.[15] discovered the shrinking diameter phenomena on time-evolving graphs. Backstrom et al.[3] studied community evolution in social networks. Berger-Wolf et al.[4] proposed a framework of dynamic social network analysis. Sun et al.[24] presented dynamic tensor analysis, which incrementally summarizes tensor streams as smaller core tensor streams and projection matrices. Tantipathananandht et al.[26] proposed frameworks and algorithms for identifying communities in social networks that change over time. Tang et al.[25] studied community evolution on multi-mode networks. As an original work on change-point detection, Sun et al.[23] proposed a information-theory based algorithm, GraphScope, to identify communities and to detect change-points on dynamic graphs without using any parameter. Falkowski et al.[9] developed a software tool to visualize community evolution and change-points. However, these algorithms considered only graphs whose edges have no directions and weights.

In this paper, we focus on community mining including community discovery and change-point detection on DWDG. There are mainly two techniques we used in our algorithms. One technique is *Random Walk with Restart*[28], which is adopted to compute the relevance matrix of a weighted directed graph(WDG). Another is *modularity*[16] which is extended to evaluate the goodness of a given partition with respect to a WDG.

The primary contributions of this paper are as following: (1)The relevance matrix computed by *Random Walk with Restart* is applied to the first step OBO-Group of community discovery algorithm, which can efficiently construct compact communities with only one scan of the nodes of a WDG; (2)The classic measure called *modularity* is used and extended onto the WDG, and the second step M-Group of community discovery algorithm is devised to merge compact communities according to heuristics of maximizing the modularity of these communities; (3)A similarity measure between partitions of continuous time segments is devised to detect the change-points and algorithm Inc-Group is presented to update the partition of a graph segment when it is necessary to add a new arriving graph into the graph segment.

The rest of the paper is organized as follows: Section 2 gives out the formal description of community mining problem. Section 3 presents the community mining algorithm including community discovery algorithm (Section 3.1) and change-point detection algorithm (Section 3.2). Section 4 shows the experiments and results and we conclude this paper in section 5.

2. PROBLEM DEFINITION

The main goal of our work is similar to[23], in which two problems *community discovery* and *change-point detection* are formally defined and addressed in a streaming fashion without any user-defined parameters. The major difference between our work and GS is that we consider both problems on DWDG but GS considered that on dynamic unweighted undirected graphs.

GraphScope(GS)[23] could not be directly used to solve our more general problems. Because GS is an algorithm based on information theoretical principle, which can be adopted to compress bool matrix and other discrete data. But in weighted graphs, weights on edges may be any real number which is continuous not discrete.

To facilitate our elaborations, some necessary definitions and notations are formally introduced.

Definition 1. (Weighted Directed Graph). A *weighted directed graph* is defined as $G(V, E, W, F)$ where V denotes the node set, $E = \{< u, v > | u, v \in V\}$ denotes the edge set where $< u, v >$ is an ordered pair of nodes, $W = \{w_{ij} \in R | i, j \in V and < i, j > \in E\}$ denotes the set of weights on edges, and F is a mapping assigning weights to edges: $F : E \rightarrow W$.

We also denote the node set, the edge set and the weight set of G by $V(G)$, $E(G)$ and $W(G)$ respectively. To simplify the presentation, we also refer to the *weighted directed graph* defined above as *graph* without explicit explanation.

Based on the definition of graph, there are some other important concepts. *Graph stream* $\mathcal{G}$ is a sequence of graph evolving over time infinitely. *Graph segment* $\mathcal{G}^s$ is a sequence of graph in the s-th time segment which include one or more continuous time-slices. *Partition* I^s is a set of communities for the graph segment $\mathcal{G}^s$. For *graph stream*, *graph segment* and *partition*, detailed definitions could be found in[23] except that G is defined as definition 1 in our setting.

Table 1: Symbols

Symbols	Definitions
G	A graph represented as a matrix with elements being weights on corresponding edges
$\tilde{G}$	Column normalized matrix of G
i, j	Node indices, $1 \leq i, j \leq n$
R	Relevance matrix of G with the i-th row and j-th column element representing relevance score of node i with respect to node j
n	Number of nodes in G
t	Time-slice index, $t \geq 1$
s	Graph or time segment index, $s \geq 1$
G^t	Graph in time-slice t
$\mathcal{G}$	*Graph stream*
$\mathcal{G}^s$	The s-th *Graph segment*
t^s	Starting time-slice of the s-th graph segment
k^s	Number of communities of $\mathcal{G}^s$
p, q	Community indices, $1 \leq p, q \leq k_s$
I^s	Partition of $\mathcal{G}^s$
I^s_p	The p-th community in I^s

Algorithm 1: *Stream-Group*($\mathcal{G}$,c_0)

```
1:Initialization:t ← 1, s ← 0, I^0 ← ∅, t^0 ← 0
2:while(G^t in 𝒢 is coming)
3:   (I^t, k) ← S-Group(G^t)
4:   c ← Sim(I^s, I^t)
5:   if c > c_0 then
6:      (I^s, k^s) ← Inc-Group(𝒢^s,I^s,G^t,I^t)
7:   else
8:      s ← s + 1, t^s ← t, I^s ← I^t
9:   report I^s,t^s
```

Definitions of main symbols throughout this paper are listed in table 1.

Formally, our task is to identify good partition I^s and good change time-slice t_s for every graph segment $\mathcal{G}^s$ where $1 \leq s < \infty$. It should be noted that our algorithms could solve community discovery and change-point detection problems on dynamic weighted directed graphs.

3. COMMUNITY MINING ALGORITHM

In this section, we propose the community mining algorithm *Stream-Group* aimed at mining community on DWDG.

The overall framework of *Stream-Group* is outlined in Algorithm 1.

From Algorithm 1, it can be clearly seen that *Stream-Group* is an incremental algorithm. *Stream-Group* listens to the new arriving graph along the graph stream continuously. When a new time-slice is coming, *Stream-Group* executes following steps: (1)Discover community structure of the arriving graph through S-Group; (2)Compute similarity between the partition(I^t) of the new arriving graph and that(I^s) of the previous graph segment; (3)Judge whether a change-point appears or not according to the partition similarity and the specified threshold(c_0), if the time-slice t is not a change points, then (4)Update the partition I^s of the

graph segment $\mathcal{G}^s$ through Inc-Group, otherwise (5)Start a new graph segment $\mathcal{G}^{s+1}$.

In the following sub-sections, S-Group, partition similarity and Inc-Group are discussed in more detail.

3.1 Community Discovery

In this section, we omit the subscripts s and t because we consider the community discovery problem in only one specific time-slice which is a special kind of graph segment including just one graph. The most important points of searching a good partition of a graph efficiently are as follows: (1)Choose a metric to evaluate the goodness of a partition summarizing the overall community structure of a graph; (2)Devise timesaving algorithms to find out the partition with the most optimal value of the above measure.

In the following subsections, we elaborate the two points respectively.

3.1.1 Evaluation of a Partition

Formally, Newman et al.[16] proposed a metric named *modularity* to evaluate the goodness of a partition of undirected graphs. Recently modularity is extended to weighted or directed graphs[14, 2, 7]. In this paper, we simply use the extended modularity to evaluate the partition of a graph.

First, we represent graph by a matrix. For graph G, assume there are totally n nodes in it, then the matrix form of the graph is,

$$G = \begin{pmatrix} g_{11} & \cdots & g_{1n} \\ \vdots & \ddots & \vdots \\ g_{n1} & \cdots & g_{nn} \end{pmatrix} \tag{1}$$

where $g_{ij} = \begin{cases} w_{ij} & < j, i > \in E \\ 0 & others \end{cases}$

Based on the matrix shown above, we rewrite the in-degree and out-degree of a node as $d_i^{in} = \sum_j g_{ij}$ and $d_i^{out} = \sum_j g_{ji}$ respectively. We denote by $A = \sum_{ij} g_{ij}$ the sum of each edge's weight in G. Then, the modularity[2] of a partition I with respect to G is as follows. We denote by p or q the community indices in the partition I and assume there are totally k communities in it.

$$Q = \frac{1}{A} \sum_{p=1}^{k} \sum_{i,j \in I_p} \left(g_{ij} - \frac{d_i^{in} d_j^{out}}{A} \right) \tag{2}$$

To be consistent with the conventional way, we present the modularity Q in a more concise way as below.

$$Q = \sum_{p=1}^{k} (e_{pp} - a_p b_p) \tag{3}$$

where $e_{pq} = \frac{1}{A} \sum_{i \in I_p} \sum_{j \in Iq} g_{ij}$ is the fraction of weights on edges between community p and q, $a_p = \frac{1}{A} \sum_{i \in I_p} \sum_j g_{ij}$ is the fraction of weights on edges that link to nodes in community p, and $b_p = \frac{1}{A} \sum_{i \in I_p} \sum_j g_{ji}$ is the fraction of weights on edges that link from nodes in community p.

It is easy to prove that the two computation formulas of Q above are equivalent.

Modularity described above is right the metric to evaluate the goodness of a partition of a graph. The larger the modularity Q is, the better the partition summarizes the overall community structure. Note that Q is a real number between 0 and 1 and when it is larger than 0.3 the community structure is remarkable[16]. The optimal modularity of real networks usually lies between 0.3 and 0.7.

Thus, given a graph G, the task of community discovery is to search a partition I maximizing the modularity Q. Unfortunately, optimization of modularity is a non-deterministic polynomial-time hard (NP-hard) problem[5].

3.1.2 Search Good Partition

To optimize the modularity Q, there are some heuristics. One of the most famous algorithms is first advanced by Newman[16]. This algorithm works as following: first regard each node as a community, then iteratively merge two communities which maximize ΔQ (the modularity difference of before and after merging two communities) until Q doesn't increase. However, recent study[30] shows that ΔQ usually makes mistake when communities are small. Good Efficiency and effectiveness is always a pair of conflict goals of community discovery.

To balance the efficiency and effectiveness, we propose a two-step algorithm to search for good partition with the optimal or near-optimal modularity. The first step constructs compact communities. The second step merges those compact communities into larger ones until the modularity doesn't increase. The following contents explain the two steps in more detail respectively.

First Step: Construct Compact Communities

What kind of community is compact? Given two communities, which one is more compact? To answer these questions, we need a metric to measure the degree of a set of nodes being like a community. There exist some kinds of definitions of community. For example, Radicchi et al.[21] proposed two definitions of community with one in a strong sense and another in a weak sense. But they can only answer whether a set of nodes form a community but can not answer the questions above.

Before introducing the measure of the goodness of a community, we first consider the relevance matrix computed through *Random Walk with Restart* (RWR) technique. It is right the observation that relevance scores between nodes within the same communities are usually higher than the ones between different communities[28] that inspires us. The computation formula of relevance matrix between each pair of nodes is as follows[28, 18, 11, 27].

$$R = (1 - d)(\mathcal{E} - d\tilde{G})^{-1} \tag{4}$$

where $\mathcal{E}$ is the identity matrix, $\tilde{G}$ is the column normalized matrix of G. The element r_{ij} represents the probability of random particle will finally stay at i when starting from j with restart probability $1 - d$.

Notice that the relevance matrix has more non-zero elements than the original matrix. If the original graph is strongly connected, there are not any zero value elements in its relevance matrix. Of course, the relevance matrix lose some detail information of the original graph, but it preserves and even strengthens the overall community structure, which is appreciated by us.

Then, we define a goodness metric *compactness* of a community based on the relevance matrix. In fact, *compactness* is a local modularity of communities and its idea is originated from the advanced concept of *modularity*. Given a sub-graph G' of G, the *compactness* of G' is computed by

the equation as follows. Denote $V(G')$ by V'.

$$C(V') = \frac{1}{B}\left(\sum_{i,j\in V'} r_{ij} - \frac{\sum_{i\in V'} r_{i*}\sum_{j\in V'} r_{*j}}{B}\right) \quad (5)$$

where $B = \sum_{ij} r_{ij}$ is the sum of all the elements in R, which is the relevance matrix of G. $r_{i*} = \sum_k r_{ik}$ and $r_{*j} = \sum_k r_{kj}$ are the sum of the i-th row's and j-th column's elements of R respectively.

Compactness is a local goodness measure of a sub-graph being a community. The larger $C(V')$ is, the more likely the sub-graph G' is a community.

Now, we present a fast heuristic OBO-Group to construct compact communities. The main idea of the algorithm is as following.

The overall view of the algorithm is to construct local compact communities one by one until there isn't any node left. In more detail, when constructing a new community, we expand the community by adding new nodes into it iteratively until Condition 1 is not satisfied. Before showing Condition 1, we give out the concept *single compactness* of a node with respect to a community, which indicates the degree of the node belonging to the community. The computation formula of *single compactness* of a node i with respect to a community V' is as following, where $i \notin V'$.

$$C(i, V') = \frac{1}{B}\left(r_{ii} + \sum_{j\in V'}(r_{ij} + r_{ji}) - \frac{r_{i*}r_{*i}}{B}\right) \quad (6)$$

where the meaning of notions r_{i*}, r_{*i} and B is the same as Equation (5).

Based on the definition of *single compactness*, we give out Condition 1 as below.

Condition 1. Current largest single compactness of possible nodes with respect to the current constructing community is larger than or equal to the previous one.

Note that theoretically the compactness of a community constructed under the constraint of condition 1 neither necessarily keep increasing nor decreasing. However, the overall trend is always increasing with little decreasing. When condition 1 is not satisfied, it also doesn't suggest that the compactness decrease.

Nonetheless, we only need to find compact communities at this step, that is, we put nodes that is obviously in the same community together first. If two nodes are in the same community very naturally, then the single compactness of one node which is added after the other is larger than or equal to the former one which is consistent with Condition 1. The reason is that as the augmentation of the community, intra-community relevance scores become more and more and inter-community ones become less and less. Thus if Condition 1 don't hold, then we think that there is not any more nodes which is definitely in the current constructing community, then start a new community construction.

The pseudo-code of the algorithm OBO-Group is presented in Algorithm 2. For a brief conclusion, OBO-Group is to put nodes into compact communities. The nodes in the same community of the partition output by OBO-Group will be definitely in the same community of the final partition but the reverse not true. OBO-Group is also a preprocessing step of the next step M-Group which gives out the final partition.

Algorithm 2: OBO-Group(G,R)

```
1:S ← {v|v ∈ V(G)}, I ← ∅, k ← 0
2:k ← k + 1, I_k ← ∅,CC ← 0
3:while S is not empty
4:   foreach v in S do
5:      compute v's single compactness with respect to
 I_k (Equation (6))
6:   record the node v with Largest Compactness(LC)
7:   if LC not less than CC then
8:      CC ← LC,add v into I_k
       else
9:      add I_k into I and Set corresponding variables as
 line 2
10:return I, k
```

Algorithm 3: M-Group(I,k)

```
1:I' ← I,k' ← k
2:while(k' larger than 1)
3:   foreach pair of communities p and q in I' do
4:      compute ΔQ according to Equation (7)
5:   record the community pair p and q with the largest
 ΔQ(LQ)
6:   if LQ larger than 0 then
7:      merge p and q in I',k' ← k' − 1
8:   else break
9:return I',k'
```

Second Step: Merge Compact Communities

Now, we introduce an algorithm M-Group to obtain the final partition that has optimal or near-optimal modularity through merging the communities output by OBO-Group.

The main idea of M-Group is to merge compact communities continuously until the modularity Q doesn't increase. That is, when merging any two existing communities produces a lower modularity, we stop the whole process of community discovery and report the final partition. The pseudo-code is described in Algorithm 3.

When merging community p and community q, the increasing amount of modularity can be calculated as Equation (7).

$$\Delta Q = e_{pq} + e_{qp} - a_p b_q - a_q b_p \quad (7)$$

The meaning of the notations above can be found in Section 3.1.1.

The straightforward derivation of Equation (7) is described below.

Denote by $D(I_p) = e_{pp} - a_p b_p$ the local modularity of community p, Then, the modularity Q_1 before merging p and q is:

$$Q_1 = \sum_{p'\neq p,q} D(I_{p'}) + D(I_p) + D(I_q) \quad (8)$$

And the modularity Q_2 after merging p and q is:

$$Q_2 = \sum_{p'\neq p,q} D(I_{p'}) + D(I_p \cup I_q) \quad (9)$$

Algorithm 4: S-Group(G)

1:**compute** *relevance matrix* R of G
2:(I,k)← OBO-Group(G,R)
3:(I,k)← M-Group(I,k)
4:**return** I,k

Then

$$\begin{aligned}\Delta Q &= Q_2 - Q_1 \\ &= D(I_p \cup I_q) - D(I_p) - D(I_q) \\ &= [e_{pp} + e_{qq} + e_{pq} + e_{qp} - (a_p + a_q)(b_p + b_q)] \\ &\quad -(e_{pp} - a_p b_p) - (e_{qq} - a_q b_q) \\ &= e_{pq} + e_{qp} - a_p b_q - a_q b_p \end{aligned} \quad (10)$$

which is in line with Equation (7)

Integrating two steps described above together, we obtain the overall framework of community discovery algorithm (see Algorithm 4 for pseudo-code) for a static graph.

3.2 Detection of Change-points

In subsection 3.1, we propose community discovery algorithm S-Group for a static graph in a specific time-slice. However, time is an important element considered by us. In this subsection, we discuss how to group time-evolving graphs into graph segments and uncover the change-points along time-slices.

Whether putting a new arriving graph into the current graph segment or starting a new graph segment is determined by the similarity between partitions of the current graph segment and that of the arriving graph. This similarity is defined as follows.

Definition 2. (Partition Similarity). Given two partitions I and I' of N nodes, the similarity between them is calculated as:

$$Sim(I, I') = \frac{P_I(I') + P_{I'}(I)}{2N} \quad (11)$$

where $P_I(I') = \sum_p max_{c \in I \cap I'} |I_p \cap c|$

For example, there are partitions $I = \{\{1,2\},\{3,4\},\{5,6\}\}$ and $I' = \{\{1,2\},\{3,4,5,6\}\}$ of six nodes. Then $I \cap I' = \{\{1,2\},\{3,4\},\{5,6\}\}$, $P_I(I') = 2+2+2 = 6$, $P_{I'}(I) = 2+2 = 4$ and $Sim(I, I') = (6+4)/12 = 0.83$.

Suppose we know the partition I^s of the current graph segment $\mathcal{G}^s$ and the partition I^t of the new arriving graph G^t, according to the size of the similarity between I^s and I^t we determine whether putting G into $\mathcal{G}^s$. If the similarity is larger than or equal to a specified similarity threshold c_0, then put the new arriving graph G^t into the current graph segment $\mathcal{G}^s$, otherwise start a new graph segment $\mathcal{G}^{s+1}$. Notice that I^t can be computed through Algorithm 4 and how to obtain and update I^s will be discussed in the coming paragraphs.

Now, we propose an incremental algorithm to find good partition of an updated graph segment $\mathcal{G}^s$. The naive way is to integrate all the graphs in $\mathcal{G}^s$ into a graph and then perform S-Group on it. To make advantage of the existing partitions I^s of $\mathcal{G}^s$ and I^t for the new arriving graph G^t, we present algorithm, Inc-Group, instead of the naive one to improve the overall performance.

Algorithm 5: Inc-Group($\mathcal{G}^s$,I^s,G^t,I^t)

1:$\mathcal{G}^s \leftarrow \mathcal{G}^s \cup \{G^t\}$
2:$H \leftarrow$ **integrate** $\mathcal{G}^s$
3:$I' \leftarrow I^s \cap I^t$,$k' \leftarrow |I'|$
4:$(I^s, k^s) \leftarrow$ M-Group(I',k')
5:**return** I^s,k^s

The main idea of Inc-Group is as follows. First, update the s-th graph segment $\mathcal{G}^s$ to $\mathcal{G}^s \cup \{G^t\}$ where $\mathcal{G}^s$ represents current or updated s-th graph segment and G^t represents the new arriving graph at time-slice t. Second, integrate graphs in $\mathcal{G}^s$ to one graph H. Third, compute the meet I' of I^s and I^t, and count the number of elements k' in I'. Forth, perform M-Group on I' and k'. At last, report the result of the forth step as the final partition I^s of the s-th updated graph segment. For more explicit, the pseudo-code is listed in Algorithm 5.

More details of line 2 in Algorithm 5 are as follows. Because in our setting, all the graphs along graph stream have the same number of nodes, so $V(H) \leftarrow V(G^t)$ and $E(H) \leftarrow \bigcup_t E(G^t)$ and $W(H) \leftarrow \frac{1}{t^{s+1}-t^s} \sum_t W(G^t)$, where $t \in [t^s, t^{s+1}-1]$.

Based on the algorithm above, we give out our overall framework of community discovery and change-point detection for infinite graph stream $\mathcal{G}$(see Algorithm 1).

Notice that the realized program of *Stream-Group* is an endless loop process because it must listen to the arriving graph on the graph stream continuously.

4. EXPERIMENTS AND RESULTS

In this section, we validate the effectiveness and efficiency of our algorithms through experiments. Results on community discovery and change-point detection are both evaluated on synthetic and real datasets. The experiments are conducted on Widows xp installed on PC with P4 1.7GHZ CPU and 512MB MEM.

4.1 Experiments on Community Discovery

In this subsection, we evaluate both the effectiveness and efficiency of community discovery algorithms through experiments on synthetic and real networks.

Synthetic Networks

For synthetic networks, data are generated by the way extended from Newman's[17]. Here, a graph stream including 16 random weighted directed graphs are generated. For each graph, there are 128 nodes divided into 4 known communities of 32 nodes. Moreover, there are 3 synthetic change-points which separate the time into 4 time segments $\{T_1 - T_4, T_5 - T_8, T_9 - T_{12}, T_{13} - T_{16}\}$. In one time segment, each graph has the same known community structure. But between any two continuous time segments, graphs have different known community structure. Table 2 has shown each graph's community structure where [i-j] means node set $\{i, i+1, i+2, \ldots, j\}$.

Then, we generate random weighted directed graphs according to their known community structure. To this end, suppose each node has on average $z_{in} = 12$ edges connecting it to members of the same community and $z_{out} = 4$ edges to members of other communities. From probability perspective, each node is connected to each member of the same community with probability 0.375(12/32) and to every member of other communities with probability 0.0417(4/96).

Table 2: The known community structure for graphs in the synthetic graph stream

Graphs	Community one	Community two	Community three	Community four
$G_1 - G_4$	[1-32]	[33-64]	[65-96]	[97-128]
$G_5 - G_8$	[121-128],[1-24]	[25-56]	[57-88]	[89-120]
$G_9 - G_{12}$	[113-128],[1-16]	[17-48]	[49-80]	[81-112]
$G_{13} - G_{16}$	[105-128],[1-8]	[9-40]	[41-72]	[73-104]

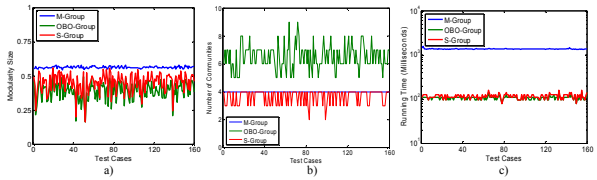

Figure 1: Running Results of algorithms M-Group, OBO-Group and S-Group on 160 Independent Cases

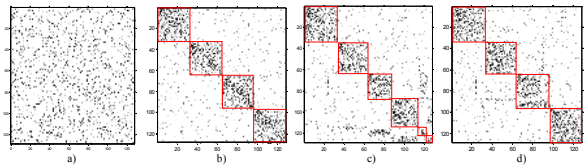

Figure 2: Grayscale Images of a) a Synthetic Graph and b-d) The Output Graphs

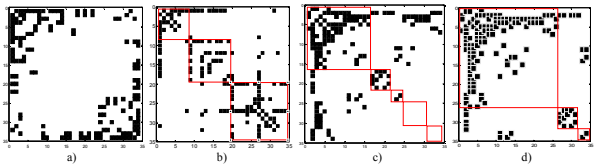

Figure 3: Grayscale Images of a) The Karate Graph and b-d) The Resulting Graphs

For weights, we generate an integer from 1 to 10 randomly for intra-community edges and 1 to 6 for inter-community ones.

First, we compare the effectiveness and efficiency of community discovery algorithms which are M-Group, OBO-Group and S-Group. To this end, 160 graphs are generated in the way described above. Then, run the three algorithms on all the generated graphs. Results are shown in Figure 1. Figure 1a) illustrates the modularity size of community structure discovered by the algorithms, while Figure 1b) shows the number of communities discovered through each algorithm. As discussed in previous sections, the larger the modularity, the better the community structure and the more effective the algorithm is. From figure 1a), we can see that M-Group is the most effective algorithm and OBO-Group is the most ineffective one while the effectiveness of S-Group falls in between. Actually, M-Group is similar with the algorithm proposed by Newman in[17]. From Figure 1b), we can see that the number of communities discovered by S-Group is the same as or near to the result of M-Group in most cases. Finally, Figure 1c) has shown the running time (in milliseconds) of the three algorithms. From these results, S-Group has nearly the same good effectiveness with M-Group and has nearly the same good efficiency with OBO-Group. It is concluded that S-Group is a good trade-off between effectiveness and efficiency comparing to the other two algorithms.

To see the effectiveness of the algorithms more deeply, we illustrate a graph and its output graphs after running different algorithms. Figure 2 has shown the results. Figure 2a) is the grayscale image of the original graph where the gray values reflect the weights of edges in the graph. The larger the gray value is, the larger the weight is. Figure 2 b) to d) are the output graphs output by M-Group, OBO-Group and S-Group respectively. From these results, we can see S-Group is nearly as effective as M-Group and only have a few nodes error-partitioned. In fact, when the graph is very large, people usually prefer to the algorithm losing some accuracy but with high efficiency. Taken in this sense, S-Group is more meaningful than others.

Real Networks

To test whether our algorithms can discover interesting communities in real networks, we choose four real networks to perform our experiments. The real networks are Karate, Football, Dolphins and Neural which are released by Newman and can be downloaded from http://www-personal.umich.edu/~ mejn/netdata/. The former three networks have no directions and weights on edges, and the last network is a weighted directed network.

For each real network, we run M-Group, OBO-Group and S-Group on it. For Figure 3 to Figure 5, the results are shown with grayscale images in which black reflects the weight of corresponding edge is 1 and white represents the weight is 0. For Figure 6, the gray value scales from 0(white) to 8(black) where white represents there are no arcs between corresponding node pair and black represents the largest weight on edges. In each of this figures, a) represents the input graphs, b) to d) are the resulting graphs after running M-Group, OBO-Group and S-Group on the input graphs respectively. Through brief analysis, the effectiveness of S-Group lies between that of M-Group and OBO-Group in all the four cases.

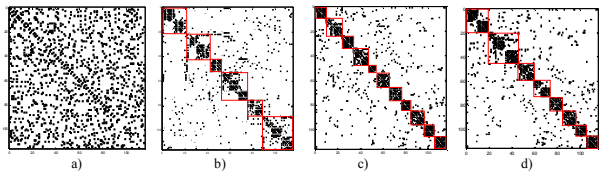

Figure 4: Grayscale Images of a) The Football Graph and b-d) The Resulting Graphs

Finally, we illustrate the scalability of our algorithms for community discovery. To this end, we vary the graph size (the number of nodes) from 100 to 1000 and generate ran-

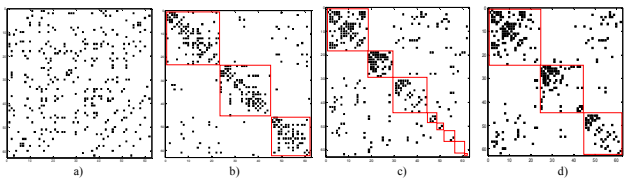

Figure 5: Grayscale Images of a) The Dolphins Graph and b-d) The Resulting Graphs

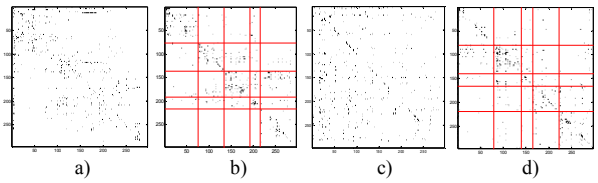

Figure 6: Grayscale Images of a) The Neural Graph and b-d) The Resulting Graphs

dom graphs of these sizes. The three algorithms are run on the generated graphs and we record their running time (in milliseconds) in Figure 7, from where we can see that S-Group as well OBO-Group have nearly linear time complexity which is much better than M-Group.

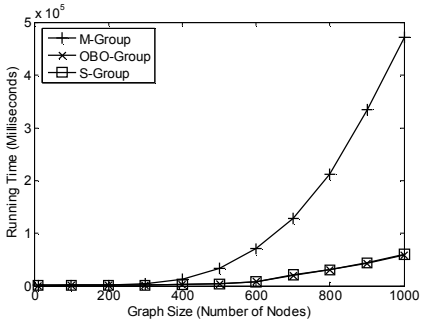

Figure 7: Running Time of Community Discovery algorithms on Graphs with Different Sizes

From the experiments in this subsection, it is concluded that S-Group can find meaningful communities as human intuition in a timesaving way.

4.2 Experiments on Change-point Detection

In this subsection, we test the ability of change-point detection of *Stream-Group* through experiments on both synthetic and real networks.

Synthetic Networks

For synthetic networks, we simply adopt the graph stream generated in subsection 4.1. Then we run *Stream-Group* on the graph stream. For the threshold of partition similarity, we choose three typical real values 0.65, 0.75 and 0.85. For each threshold and each time-slice of 16, we record the similarity between partitions of current time-slice and previous time segment. The results are shown in Figure 8. Similarities lower than thresholds indicate change-points. We can see that *Stream-Group* has detected all the three synthetic change-points(5, 9 and 12) when choosing 0.75 as the similarity threshold as Figure 8b) shows. When the threshold is

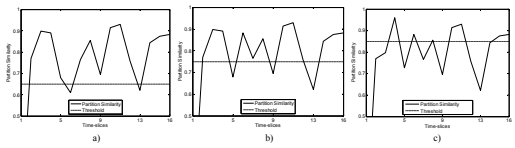

Figure 8: Similarities between Partitions of Previous Time segment and Current time-slice by Setting Three Different similarity Thresholds a) 0.65, b) 0.75 and c) 0.85

lower (0.65), the number of change-points becomes smaller as Figure 8a) shows. When the threshold is higher (0.85), the number of change-points becomes larger as Figure 8c) shows.

Real Networks

To test whether our algorithms can detect interesting change-points in real networks, we choose Enron dataset which is a well known public dataset for social network analysis. The MySql Script of Enron dataset can be downloaded from http://www.isi.edu/~adibi/Enron/Enron.htm. Totally 19673 e-mail addresses with suffix "@enron.com" are extracted from the original dataset. The graphs are constructed from communications among these extracted e-mails from Jan 1999 to July 2002. We choose one week as a time-slice and there are totally 185 time-slices. Edges represent there are communications between e-mails. The direction of an edge originates from the sender and terminates at the recipient. The weights on edges represent the frequency of corresponding communications.

Because of our ignorance of the actual partitions and change-points of Enron dataset, It is hard for us to set a reasonable threshold of partition similarity. Nevertheless, an aforehand exploration into the Enron dataset is first conducted by us, that is, we compute the similarity between partitions of any two adjacent time-slices at the beginning of the experiments. To this end, we set the similarity threshold c_0 to 1.0 in *Steam-Group*. Because it is impossible that the partition similarity is larger than 1.0, so any time-slice itself is an independent time segment under the constraint of the threshold 1.0.Through simple statistics, the mean value of similarities between continuous adjacent time-slices is 0.3351 and the median value is 0.3214.

Based on the above observations, the similarity threshold c_0 is set by us to 0.32, 0.31 and 0.30 which are a bit smaller than the mean and median value. The relative similarity (the partition similarity minus the similarity threshold) between the partitions of every time-slice and its previous time segment under different thresholds are recorded in Figure 9.

In Figure 9, the similarities below the relative thresholds (dotted lines) represent the change-points under the confidence of corresponding similarity thresholds. Through simple statistics, there are 65, 58 and 50 change-points under the three thresholds respectively. The average number of time-slices in one time segment are 2.8, 3.2 and 3.7 respectively. From Figure 9, we can see that the size of the time segments scales from several to dozens. To further investigate the experimental results, formally the continuous time-slices with partition similarity above the corresponding thresholds is called stable phase, and that with partition similarity below the thresholds is called fluctuant phase. Figure 9 has marked the primary stable phases and fluctu-

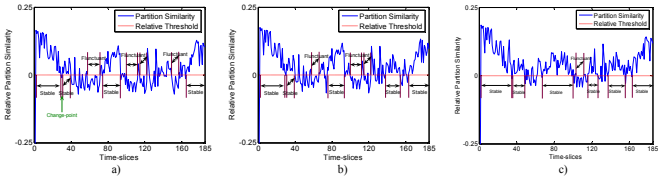

Figure 9: Relative Similarities Between Partitions of Previous Time segment and Current Time-slice For Enron Data by Setting Three Different Similarity Thresholds a) 0.32, b) 0.31 and c) 0.30

ant phases under different thresholds. The left unmarked time-slices are very small stable or fluctuant phases which is always not so meaningful to real applications. As the threshold increases, the primary stable phases become longer and the primary fluctuant phases become shorter, which accords with human intuition. From the primary stable and fluctuant phases, it can be see when the community structure of Enron communication network is stable and when it is fluctuant.

To investigate whether the change-points detected by *Stream-Group* are accordant with actual situation, we consider two adjacent stable phases t_1-t_{29} and t_{30}-t_{39} that are divided by the change-point 30 in Figure 9a). From the report of the counterpart algorithm GraphScope[23, Figure 1], about the 30-th timestamp (Nov 1999) is the time when Enron launched which is one of real change-points. So *Stream-Group* detected the actual change-points. Other change-points can be analyzed in the same way.

5. CONCLUSIONS

We propose *Stream-Group* to discovery communities and to detect change-points on DWDG. In the process of *Stream-Group*, algorithm S-Group is proposed to detect communities in a particular time-slice. In order to ease the contradiction between effectiveness and efficiency, S-Group combines two heuristic steps OBO-Group and M-Group. For change-point detection, a partition similarity measure is devised. According to the similarity between the partition of the new arriving graph and that of the current graph segment, it can be determined whether we need expand the current graph segment by adding the new arriving graph into it or start another graph segment. Our algorithms are carefully designed to achieve a balance between the effectiveness and efficiency in community discovery. Through extensive experiments on both synthetic and real networks, we have validated the efficiency and effectiveness of our algorithms. As community mining has a broad application domain, our work can be applied to solve related problems in many different applications. Future work includes how to find the underlying rule of these communities and change-points, such as classifying the discovered communities, finding out the cause of the change-points and so on.

6. REFERENCES

[1] C. Alper and S. Yao. Spectral partitioning: the more eigenvectors, the better. *ACM/IEEE Design Automation Conf*, pages 195–200, 1994.

[2] A. Arenas, J.Duch, A. Fernandez, and S. Gomez. Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(176), 2007.

[3] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. *KDD*, pages 44–45, 2006.

[4] T. Y. Berger-Wolf and J. Saia. A framework for analysis of dynamic social networks. *KDD*, pages 523–528, 2006.

[5] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. On finding graph clusterings with maximum modularity. *WG*, pages 121–132, 2007.

[6] D. Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. *EKDD*, pages 195–200, 1994.

[7] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E 70*, 2004.

[8] N. Du, B. Wu, X. Pei, B. Wang, and L. Xu. Community detection in large-scale social networks. *SNA-KDD*, pages 16–25, 2007.

[9] T. Falkowski, J. Bartelheimer, and M. Spiliopoulou. Community dynamics mining. *European Conference on Information Systems*, 2006.

[10] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, 2002.

[11] J. He, M. Li, H. Zhang, H. Tong, and C. Zhang. Manifold ranking based image retrieval. *ACM Multimedia*, pages 9–16, 2004.

[12] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[13] M. Kurucz and A. Bencz. Spectral clustering in telephone call graphs. *SNA-KDD*, pages 82–91, 2007.

[14] E. Leicht and M. E. J. Newman. Community structure in directed networks. *Phys Rev Lett*, 100(11), 2008.

[15] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. *KDD*, pages 177–178, 2005.

[16] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 2004.

[17] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 69(2), 2004.

[18] J. Y. Pan, H. J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia crossmodal correlation discovery. *KDD*, pages 653–658, 2004.

[19] C. Pizzuti. Community detection in social networks with genetic algorithms. *Annual Conference on Genetic and Evolutionary Computation*, pages 1137–1138, 2008.

[20] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *PNAS*, 101(9):2658–2663, 2004.

[21] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *PNAS*, 101(9):2658–2663, 2004.

[22] M. Rosvall and C. T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. *PNAS*, 104(18):7327–7331, 2007.

[23] J. Sun, C. Faloutsos, S. Papadimitriou, , and P. S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. *KDD*, pages 687–696, 2007.

[24] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. *KDD*, pages 374–383, 2006.

[25] L. Tang, H. Liu, J. Zhang, and Z. Nazeri. Community evolution in dynamic multi-mode networks. *KDD*, pages 677–685, 2008.

[26] C. Tantipathananandh, T. Berger-Wolf, and D. Kemne. A framework for community identification in dynamic social networks. *KDD*, pages 717–726, 2007.

[27] H. Tong and C. Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. *KDD*, 2006.

[28] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. *ICDM*, pages 613–622, 2006.

[29] S. van Dongen. Graph clustering by flow simulation. *PhD thesis, University of Utrecht*, 2000.

[30] Y. Wang, H. Song, W. Wang, and M. An. A microscopic view on community detection in complex networks. *Proceeding of the 2nd PhD workshop on Information and knowledge management*, 2008.