CHRONICLE: A Two-Stage Density-Based Clustering Algorithm for Dynamic Networks

Min-Soo Kim and Jiawei Han

Department of Computer Science, University of Illinois at Urbana-Champaign
{msk, hanj}@cs.uiuc.edu

Abstract. Information networks, such as social networks and that extracted from bibliographic data, are changing dynamically over time. It is crucial to discover time-evolving communities in dynamic networks. In this paper, we study the problem of finding time-evolving communities such that each community freely forms, evolves, and dissolves for any time period. Although the previous t-partite graph based methods are quite effective for discovering such communities from large-scale dynamic networks, they have some weak points such as finding only stable clusters of *single path* type and not being scalable *w.r.t.* the time period. We propose *CHRONICLE*, an efficient clustering algorithm that discovers not only clusters of single path type but also clusters of *path group* type. In order to find clusters of both types and also control the dynamicity of clusters, CHRONICLE performs the *two-stage* density-based clustering, which performs the 2nd-stage density-based clustering for the t-partite graph constructed from the 1st-stage density-based clustering result for each timestamp network. For a given data set, CHRONICLE finds all clusters in a fixed time by using a fixed amount of memory, regardless of the number of clusters and the length of clusters. Experimental results using real data sets show that CHRONICLE finds a wider range of clusters in a shorter time with a much smaller amount of memory than the previous method.

1 INTRODUCTION

Recently, there is an increasing interest to mining dynamics of information networks that evolve over time. Examples of dynamic networks include network traffic data [12], telephone traffic data[1], bibliographic data[2], dynamic social network data [6, 14], and time-series microarray data [16], where a dynamic network is regarded as a sequence of networks with different timestamps (or temporal intervals). A cluster in dynamic network data, which is also called a *community*, typically represents *cohesive subgroup of individuals* within a network that persists for a specific time interval [12, 14, 2]. That is, a community is cohesive both in a timestamp network and across time. Identifying communities from dynamic networks has been paid much attention lately as an important research topic.

[1] http://reality.media.mit.edu/download.php
[2] http://www.informatik.uni-trier.de/~ley/db

In this paper, we study the problem of finding time-evolving communities such that each community freely forms, evolves, and dissolves for any time period. Such communities would look like *chronicle patterns* of a historical diagram. Figure 1 shows some chronicle patterns of the European historical diagram of between A.D. 0 and 1,700, where multiple "dynasties" co-existed at the same period of time or started/ended at different times. Mining chronicle patterns over dynamic networks would be able to discover some interesting and important knowledge that could be invisible with the previous clustering methods for static networks. For example, it could find the evolutionary collaborating groups of researchers from the DBLP data and identify how they evolve as time goes on.

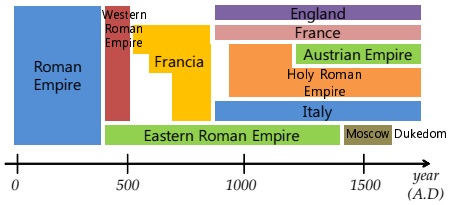


Fig. 1. Some chronicle patterns of the European historical diagram.

There have been several approaches for finding communities in dynamic networks. First, the concept of *evolutionary clustering* has been proposed to capture the evolutionary process of clusters in temporal data. Several evolutionary clustering methods [4, 13, 10] have been proposed, but they have some drawbacks such as assuming only a fixed number of communities over time and not being scalable with data size. Since the forming of new communities or the dissolving of existing communities is quite natural and common phenomena in real dynamic networks [1], and moreover, the size of real dynamic networks tends to be large, the existing evolutionary clustering methods could not be very useful in real applications. Second, there have been some methods to detect break points through clustering [12, 6]. For example, GraphScope [12] identifies the change points where the subgroup structure of hosts (or servers) is largely changed in network communication data. However, they do not allow arbitrary insertion/deletion of nodes or arbitrary start/stop of communities over time, which occurs quite often in real dynamic networks, and thus, they could be less useful for real data. Third, there have been few studies that discover communities by using t-partite graph from temporal data [11, 2]. They perform clustering for each network, construct a t-partite graph by connecting between similar local clusters at different times, and find a sequence of local clusters as a community. Especially, the methods proposed by Bansal et al. [2] explicitly handles dynamic network data, and the *BFS method* among them is known to be the most efficient method. It is scalable with the number of nodes, finds a variable number of communities with arbitrary start/stop over time, and allows arbitrary insertion/deletion of nodes.

Although the existing t-partite graph based method, especially the BFS method is quite effective for discovering clusters from large-scale dynamic net-

works, it has also some weak points: (1) finding only stable clusters of *single path* (*i.e.*, a sequence of local clusters over time); (2) finding a very small number of clusters (*i.e.*, the most stable *top-k* clusters); (3) not being scalable *w.r.t.* the length of dynamic networks; and (4) using a large amount of memory depending on its parameters. All these weak points are caused by the fact that the BFS method is based on a dynamic programming (DP) algorithm. Besides the clusters of single path type, actually, there are many cohesive clusters of non-single path type in t-partite graph. Figure 2 shows an example of t-partite graph over three timestamp networks. Each network has 3∼4 local clusters. The numbers on lines between T_1 and T_2 (or T_2 and T_3) indicate that they have a non-zero similarity. When $k = 1$, the BFS method finds a single path cluster $c_{11}c_{21}c_{31}$ because it has the strongest similarities between local clusters. However, if some members in c_{12} transfer to c_{24}, some members in c_{13} to c_{23}, and the members in c_{23} and c_{24} are merged into c_{33}, then there could be another cluster like $(c_{12}c_{13})(c_{23}c_{24})c_{33}$, where the similarity between $(c_{12}c_{13})$ and $(c_{23}c_{24})$ and the similarity between $(c_{23}c_{24})$ and c_{33} might be very high (*i.e.*, very cohesive) although the similarity of each single path (*e.g.*, $c_{13}c_{23}c_{33}$) are not so high. Here, () represents a consolidation of multiple local clusters. We call a cluster of this type as a *path group* cluster since there are multiple paths over time in the cluster. The BFS cannot find the clusters of this type.

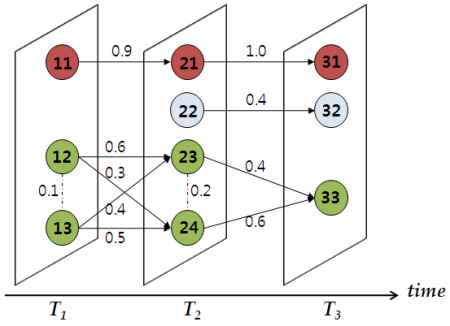


Fig. 2. An example of t-partite graph constructed from a dynamic network.

In this paper, we propose a density-based clustering algorithm, *CHRONICLE*, that efficiently discovers both single path clusters and path group clusters. For finding clusters of both types, CHRONICLE performs the density-based clustering in *two stages*: the 1st-stage density-based clustering for each timestamp network and the 2nd-stage density-based clustering for the t-partite graph. In case of the previous BFS method, it only performs the 1st-stage clustering and finds single path clusters by using a DP algorithm. A density-based clustering approach has several advantages such as discovering clusters of arbitrary shape, handling noise, and being fast. These features allow us to find a wider range of clusters (*i.e.*, not only single path clusters, but also path group clusters) in an efficient way. As the length of dynamic networks, the number of clusters, or the length of cluster (*i.e.*, path length) increases, the running time and the amount of memory usage of the BFS method largely increase. Using disk for sav-

ing the amount of memory would cause performance degradation. In contrast, for a given data set, CHRONICLE finds all clusters in a fixed time by using a fixed amount of memory, regardless of the number of clusters and the length of clusters. The density parameter ε allows us to control both the number of clusters and the dynamicity of communities. Here, the *dynamicity* indicates how much communities change over time.

Our proposed algorithm, CHRONICLE, has the following key features:

- Discovering a variable number of communities with arbitrary start/stop over time in dynamic networks
- Finding a wide range (*i.e.*, stable and dynamic) communities of both single path type and path group type
- Being scalable *w.r.t.* the length of dynamic networks
- Being fast and using only a small amount of memory irrespective of both the number of clusters and the length of clusters

The rest of this paper is organized as follows. Section 2 presents the most related piece of work, the BFS method [2]. Sections 3 presents our CHRONICLE algorithm, and Section 4 the results of experimental evaluation. Finally, Section 5 concludes the study.

2 RELATED WORK

In this section, we briefly explain the BFS method [2]. Given a dynamic network $\mathcal{G} = \{G_1, \ldots, G_t\}$, the BFS method first finds local clusters for each timestamp network G_i by using the biconnected component algorithm. Actually, biconnected components are not very good clusters for network/graph data. There are many other better methods using well-known measures such as min-max cut, normalized cut, modularity, betweenness, and structural similarity. Bansal et al. mentioned the BFS method used biconnected components as local clusters for simplicity and fast mining. After clustering for each timestamp network, the BFS method constructs a t-partite graph by connecting between similar local clusters at different times. The core and unique part of the BFS method is finding the most stable top-k clusters of length at least l_{min} over the t-partite graph.

For finding top-k paths of length l_{min}, the BFS method needs to maintain up to l_{min} heaps of size k for each node c_{ij} of the t-partite graph. We denote this heap as h_{ij}^x $(1 \le x \le l_{min})$, where each h_{ij}^x contains at most k entries. During scanning each node c_{ij} of each partite of the t-partite graph from 1 to t, the BFS method calculates top-k (or fewer) highest weighting subpaths of length x ending at c_{ij} $(1 \le x \le l_{min})$. This task is easily performed by merging the subpaths in the heaps of the previous partite, *i.e.*, $h_{i'j'}^x$ $(i' < i)$ with the subpaths in the current heaps, *i.e.*, h_{ij}^x, where $c_{i'j'}$ and c_{ij} are connected in the t-partite graph, and choosing new top-k subpaths. For finding paths of length greater than l_{min}, the BFS method should maintain additional data structure to contain the top scoring paths of all length greater than l_{min} for each c_{ij}.

In fact, the BFS method is a modified version of the Viterbi algorithm, a DP algorithm for finding the most probable state path in Hidden Markov

Model (HMM). The original Viterbi algorithm does not need to maintain heaps like h_{ij}^x because it finds only the top-1 state path of the same length with a given sequence. In contrast, the BFS method requires l_{min} heaps of size k because it finds the top-k paths of length at least l_{min}. Although the BFS method has a strong point that finds the exact top-k paths by using a DP algorithm, it consumes a large amount of memory due to a lot of heaps, and moreover, such memory consumption increases as the length of dynamic network n_{ts}, the number of clusters k, or the minimum length of clusters l_{min} gets larger. The excessive computation for heaps also incurs an increase of running time. Using disk swapping might save memory, but it would also cause an additional performance degradation. Besides, the BFS method tends to discover only a small number of very stable clusters that are hardly changed over time. However, many applications might need to find not only a small number of stable clusters, but also a large number of various clusters enough to capture the structure of the entire dynamic network.

3 CHRONICLE ALGORITHM

In this section, we present the three parts of our CHRONICLE algorithm, the 1st-stage clustering, constructing t-partite graph, and the 2nd-stage clustering in Sections 3.1, 3.2, and 3.3, respectively.

3.1 The 1st-Stage Clustering

In this section, we briefly present the 1st-stage density-based clustering method, which we denote as CHRONICLE$_{1st}$, to find all local clusters for each timestamp network. We define a dynamic network $\mathcal{G}$ as a sequence of networks $G_i\,(V_i, E_i)$, *i.e.*, $\mathcal{G} = \{G_1, \ldots, G_t\}$.

For similarity measure, CHRONICLE$_{1st}$ uses the *cosine similarity* (or *structure similarity*) [9, 15], which is one of the well-known measures for network data. Definitions 1∼2 show the concept of the structural similarity. Intuitively, $\sigma(v, w)$ indicates how many nodes v and w share *w.r.t.* the overall number of their adjacent nodes including themselves. By definition, $\sigma(v, w)$ becomes non-zero only if v is directly connected to w with an edge. The value of $\sigma(v, w)$ is in the range 0.0∼1.0 and especially becomes 1.0 when both v and w are in a clique.

Definition 1. *The neighborhood $N(v)$ of a node $v \in V$ is defined by $N(v) = \{x \in V \mid \langle v, x\rangle \in E\} \cup \{v\}$.*

Definition 2. *The structural similarity $\sigma(v, w)$ of a node pair $(v, w) \in V \times V$ is defined by*

$$\sigma(v, w) = \frac{|N(v) \cap N(w)|}{\sqrt{|N(v)| \times |N(w)|}}.$$

CHRONICLE$_{1st}$ is actually equivalent to the SCAN algorithm [15], which is again basically the same with the original density-based clustering algorithm DBSCAN [5], but uses the structural similarity as the distance measure between two nodes instead of Euclidean distance. With the notions of density-based clustering, CHRONICLE$_{1st}$ finds a high density subset of nodes, *i.e.*, a *topologically*

dense subgraph like a quasi-clique, as a cluster in each G_i. Figure 3 shows an example network and four local clusters found by CHRONICLE$_{1st}$ over the network. After clustering, there remain some nodes that have relatively low similarity with its adjacent nodes, and thus, do not belong to any cluster. Such nodes are considered *noises* by the notions of density-based clustering. We skip the explanation of how to determine two density parameters μ and ε since it is not a core part of this paper and there are some methods for it.

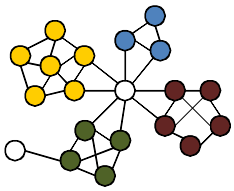


Fig. 3. An example network and four local clusters.

3.2 Constructing t-Partite Graph

CHRONICLE constructs a t-partite graph from the 1st-stage clustering result. This is performed by connecting between two local clusters c_{ij} and $c_{i'j'}$ $(i' < i)$ that have a non-zero similarity (or affinity). For a similarity measure, there are many candidates, and Jaccard coefficient $Jaccard(c_{ij}, c_{i'j'})$ would be a good candidate. For example, if $|c_{ij}| = 6$, $|c_{i'j'}| = 8$, and c_{ij} and $c_{i'j'}$ have four common nodes, then $Jaccard(c_{ij}, c_{i'j'}) = \frac{4}{6+8-4} = 0.4$. We might be able to connect between two local clusters in non-adjacent timestamp networks $G_{i'}$ and G_i $(|i' - i| > 1)$, where we call $g = |i' - i|$ as a *gap*, but we only deal with the case of $g = 1$ in this paper due to space limit. Each link between a pair of nodes of t-partite graph has its own weight based on the similarity between the corresponding local clusters. Hereafter, we call a connection between two different partites in the t-partite graph as a *link* for discriminating it from normal edges. We also denote the t-partite graph as T.

Besides links, CHRONICLE also connects two local clusters in the same partite that have *inter cluster edges* (simply, *ic-edges*) between them. Let ν_j be the number of edges with one endpoint in c_{ij} and the other endpoint in $\overline{c_{ij}}$, where $\overline{c_{ij}}$ denotes the complement of c_{ij}, and let ω_j be the number of edges with both endpoints in c_{ij}. We denote the number of edges with one endpoint in c_{ij} and the other endpoint in $c_{ij'}$ as $\nu_{jj'}$ and denote $\nu_j + \omega_j = \tau_j$, which is equal to the sum of degrees of nodes in c_{ij}. Then, we define a similarity measure $InterEdge(c_{ij}, c_{ij'}) = \frac{\nu_{jj'}}{\tau_j + \tau_{j'}}$ for two local clusters in the same network. As two local clusters c_{ij} and $c_{ij'}$ are more tightly connected with more ic-edges, the value of $InterEdge(c_{ij}, c_{ij'})$ increases. CHRONICLE uses these ic-edges for the 2nd-stage clustering. In Figure 2, the dashed line between c_{12} and c_{13} and that between c_{23} and c_{24} represent ic-edges. We denote the nodes, links, and ic-edges of T as V_T, L_T, and E_T, respectively.

3.3 The 2nd-Stage Clustering

Similarity measure. In this section, we present the similarity measure for the 2nd-stage clustering method CHRONICLE$_{2nd}$. Different from a timestamp

network G_i, the t-partite graph T has weights on each link/ic-edge, and has time semantics from T_1 to T_t. Thus, our similarity measure, *general similarity (GS)*, for CHRONICLE$_{2nd}$ considers those two features of T.

The key concept of *GS* is the integration of the *structural affinity (SA)* and *weight affinity (WA)* between two nodes so as to discover both single path clusters and path group clusters. *GS* is defined as in Eq. 1. We just use the structural similarity $\sigma(v, w)$ as $SA(v, w)$. If we only use $SA(v, w)$ instead of *GS* as a measure for T, the clustering result could be awkward. For example, in Figure 2, $\sigma(v, w)$ would identify $c_{22}c_{32}$ as a very strong cluster since $\sigma(c_{22}, c_{32}) = 1$ even though its weight is just 0.4.

$$GS(v, w) = SA(v, w) \times WA(v, w) \tag{1}$$

For considering time semantics of t-partite graph, CHRONICLE$_{2nd}$ restricts the scope of measuring similarity to each bipartite within time interval $[i, i + 1]$. Figure 4 shows the general case of the relationship between v in T_i and w in T_{i+1} in T. When measuring the similarity between v and w, CHRONICLE$_{2nd}$ only takes account of nodes and links/ci-edges within time interval $[i, i + 1]$ except the links within time interval $[i - 1, i]$ and those within time interval $[i + 1, i + 2]$. This restriction prevents that the links of such outside timestamps affect the similarity of v and w, and at the same time, allows CHRONICLE$_{2nd}$ to find clusters in an on-line fashion, *i.e.*, perform incremental clustering for a new timestamp network G_{t+1}. Unless having such restriction, the nodes in two extreme partites of T, *i.e.*, T_1 and T_t would always have some distorted similarity values due to the imbalanced number of neighborhood nodes.

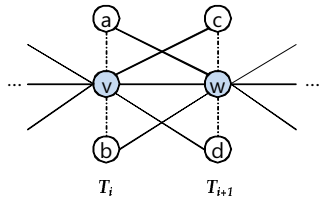


Fig. 4. The general case of the relationship between v of T_i and w of T_{i+1}.

The sub-similarity measure *SA* gives a cohesive path group a chance to be found as a cluster. In Figure 4, every pair of v and w would have zero or more neighborhood nodes. If there is no neighborhood, v and w forms a simplest bi-clique, and so, the weight itself between v and w determines the overall similarity between them. However, if there are some neighborhood nodes, we consider that not only the weights on links or ic-edges, but also the structural cohesion between them affects the overall similarity. That is, the more common neighborhood nodes v and w share compared with all their neighborhood nodes, v and w receive a high score of structural affinity. For example, in Figure 2, four links of $c_{12}c_{23}$, $c_{12}c_{24}$, $c_{13}c_{23}$, and $c_{13}c_{24}$ belong to the bi-clique of $\{c_{12}, c_{13}, c_{23}, c_{24}\}$, and so, all their structural similarities become 1.0 just like a single path. This is reasonable since it gives a chance to a group of paths that have weak similarities, but are highly intertwined, to be found as one strong cluster of path group type.

The sub-similarity measure weight affinity *WA* considers the weights of links/ic-edges between v and w including common neighborhood nodes. *WA* is defined as in Definitions 3∼4. Intuitively, $\Phi(v, w)$ in Definition 3 represents the sum of weights conveyed from v to w through their common neighborhood nodes. *WA* is a weighted combination of a direct affinity between v and w, *i.e.*, $\phi(v, w)$, and a indirect affinity between v and w, *i.e.*, $\Phi(v, w)$. This combination indicates the expected maximum possible affinity between two consolidations of v and w with their common neighborhood nodes (*e.g.*, affinity between $\{a, b, v\}$ and $\{c, d, w\}$ in Figure 4). The parameter α allows user to control the weights of the direct and indirect affinities.

Definition 3. *Let $\phi(v, w)$ be a similarity weight between v and w, and $\Omega(v, w)$ common neighborhood nodes of v and w, i.e., $\Omega(v, w) = (N(v) \cap N(w) - \{v, w\})$. The common neighborhood weights $\Phi(v, w)$ of between $v \in V_i$ and $w \in V_{i+1}$ is defined by*

$$\Phi(v, w) = \sum_{x \in \Omega(v,w)} (\phi(v, x) + \phi(x, w)).$$

Definition 4. *The weight affinity $WA(v, w)$ of a node pair $(v, w) \in V_i \times V_{i+1}$ is defined by*

$$WA(v, w) = \alpha \cdot \phi(v, w) + (1 - \alpha) \cdot \Phi(v, w), \quad for\ 0 \leq \alpha \leq 1.$$

Finally, CHRONICLE$_{2nd}$ gives a high general similarity *GS* score to a pair of nodes whose both *SA* and *WA* are high. For example, in Figure 2, $SA(c_{12}, c_{23}) = 1$ since c_{12} and c_{13} are a part of the bi-clique, and, when $\alpha = 0.5$, $WA(c_{12}, c_{23}) = 0.5 \times 0.6 + 0.5 \times (0.5 + 0.5) = 0.8$. Thus, $GS(c_{12}, c_{23}) = 1 \times 0.8 = 0.8$. Likewise, $GS(c_{12}, c_{24}) = 1 \times 0.85 = 0.85$, $GS(c_{13}, c_{23}) = 1 \times 0.9 = 0.9$, and $GS(c_{13}, c_{24}) = 1 \times 0.75 = 0.75$.

Notions of density-based clustering. We summarize the notions of density-based clustering using the *GS* measure for t-partite graph through Definitions 5∼11. The similar notions are used in other density-based clustering methods such as DBSCAN [5], SCAN [15], and TRACLUS [8] for points, static network, and trajectory data, respectively. However, different from those methods, our notions consider time semantics existing in T. The CHRONICLE$_{2nd}$ using this notions takes two density parameters, μ and ε, and discovers all subgraphs with high *GS* scores as clusters (*i.e.*, communities) on T.

Definition 5. *The ε-neighborhood $N_\varepsilon(v)$ of a node $v \in V_T$ is defined by $N_\varepsilon(v) = \{x \in N(v) \mid \langle v, x \rangle \in L_T \ \wedge \ GS(v, x) \geq \varepsilon\}$.*

Definition 6. *A node $v \in V_T$ is called a core node w.r.t. ε and μ if $|N_\varepsilon(v)| \geq \mu$.*

We note that CHRONICLE$_{2nd}$ considers only the nodes connected with v not by *ic-edges* but by *links* as the candidates of $N_\varepsilon(v)$ in order to expand a cluster in the direction of time. We also note that $v \in N(v)$ and $GS(v, v) = 1$ in Definition 5, and so, the value of $|N_\varepsilon(v)|$ is always at least 1 in Definition 6.

Definition 7. *A node $x \in V_T$ is gs-direct reachable from a node $v \in V_T$ w.r.t. ε and μ if (1) v is a core node and (2) $x \in N_\varepsilon(v)$.*

Definition 8. *A node $v_j \in V_T$ is gs-reachable from a node $v_i \in V_T$ w.r.t. ε and μ if there is a chain of nodes $v_i, v_{i+1}, \ldots, v_{j-1}, v_j \in V_T$ such that v_{i+1} is gs-direct reachable from v_i $(i < j)$ w.r.t. ε and μ.*

Definition 9. *A node $v \in V_T$ is gs-connected to a node $w \in V_T$ w.r.t. ε and μ if there is a node $x \in V_T$ such that both v and w are gs-reachable from x w.r.t. ε and μ.*

Definition 10. *A non-empty subset $S \subseteq V_T$ is called a gs-connected cluster w.r.t. ε and μ if S satisfies the following two conditions:*

(1) Connectivity: $\forall v, w \in S$, v is gs-connected to w w.r.t. ε and μ

(2) Maximality: $\forall v, w \in V_T$, if $v \in S$ and w is gs-reachable from v w.r.t. ε and μ, then $w \in S$.

We note that the gs-reachability is the transitive closure of direct gs-reachability, and it is asymmetric. It is only symmetric for a pair of core nodes. However, the gs-connectivity is a symmetric relation, which is an important property for incremental clustering because it guarantees the consistency of the clustering result regardless of whether performing batch clustering for the whole T or performing incremental clustering for every bipartite of T,*i.e.*, $\{\langle T_1, T_2\rangle, \ldots, \langle T_{t-1}, T_t\rangle\}$.

Definition 11. *Let P be a set of gs-connected clusters found by Definition 10. A node $v \in V_T$ is a noise if v is not contained in any cluster of P.*

CHRONICLE$_{2nd}$ algorithm. Algorithm 1 outlines the pseudo-code of the CHRONICLE$_{2nd}$ algorithm for batch clustering. CHRONICLE$_{2nd}$ makes one scan over a t-partite graph $T = \langle V_T, L_T, E_T\rangle$ and finds a set of gs-connected clusters $\mathcal{CR} = \{C_i\}$ w.r.t. ε and μ. Since there could exist many small-sized clusters composed of only two nodes like $c_{22}c_{32}$ in Figure 2, we set $\mu = 2$ in most cases for not missing them.

For each node of T, there are two kinds of labels: UNCLASSIFIED and NON_MEMBER. At first, all nodes are labeled as UNCLASSIFIED (line 2). If there is a node that is not classified yet (line 3), CHRONICLE$_{2nd}$ checks whether the node is a core node (line 4). If the node v is a core node, CHRONICLE$_{2nd}$ finds a gs-connected cluster containing v and adds the cluster to $\mathcal{CR}$ (line 5). Otherwise, CHRONICLE$_{2nd}$ labels the node as NON_MEMBER (line 7). After finding all clusters, the NON_MEMBER nodes can be further classified into *outliers* or *hubs* by whether the node has edges to only one cluster or multiple clusters, respectively, although the corresponding codes are not presented in the algorithm.

Algorithm 2 outlines the pseudo-code of the *FindCluster* algorithm, a subroutine of CHRONICLE$_{2nd}$. FindCluster finds all nodes that are gs-reachable from a given seed node v. It starts with inserting v into a queue Q (line 2). Then, FindCluster searches those nodes by repeating the following steps until Q is empty: (1) calculating the direct gs-reachable nodes R from the front node x of Q; (2) inserting the part of R into Q; and (3) deleting x from Q (lines

Algorithm 1 CHRONICLE$_{2nd}$

Input: (1) t-partite graph $T = \langle V_T, L_T, E_T \rangle$,
(2) Minimum number of nodes μ,
(3) Similarity threshold ε.
Output: Communities $\mathcal{CR} = \{C_i\}$.
1: $\mathcal{CR} \leftarrow \emptyset$;
2: $\forall v \in V : v \leftarrow$ UNCLASSIFIED;
3: **for each** UNCLASSIFIED node $v \in V$ **do**
4: **if** v is a core node **then**
5: $\mathcal{CR} \leftarrow \mathcal{CR} \cup findCluster(v)$;
6: **else**
7: $v \leftarrow$ NON_MEMBER;
8: **return** $\mathcal{CR}$;

Algorithm 2 FindCluster

Input: (1) A core node v,
(2) T, μ, ε.
Output: A gs-connected cluster C.
1: $C \leftarrow \emptyset$;
2: $Q.push(v)$; // Q is a queue
3: **while** $Q.empty() =$ **false do**
4: $x \leftarrow Q.front()$;
5: $R \leftarrow \{y \in V \mid y$ is direct gs-reachable from $x\}$;
6: **for each** $y \in R$ **do**
7: **if** y is UNCLASSIFIED **then**
8: $C \leftarrow C \cup \{y\}$;
9: $Q.push(y)$;
10: **if** y is NON_MEMBER **then**
11: $C \leftarrow C \cup \{y\}$;
12: $Q.pop()$;
13: **return** C;

3∼5, 9, 12). For each node y of R, FindCluster inserts y into a result cluster C and a queue Q if y is UNCLASSIFIED (lines 8∼9), or inserts y only into C if y is NON_MEMBER (line 11). Here, that y is NON_MEMBER means that y is visited before and is not a core node.

The time complexity of the CHRONICLE$_{2nd}$ algorithm is $O(2 \cdot |L_T| + 2 \cdot |E_T|)$. It is because the algorithm visits each node $v \in V_T$ only once and checks the GS scores between v and its neighborhood nodes. We note that this complexity is not affected by the number of clusters or any parameters such as μ, ϵ, and α.

Example 1. Consider performing CHRONICLE$_{2nd}$ for the t-partite graph in Figure 2 with the parameter setting of $\alpha = 0.5$, $\mu = 2$, and $\varepsilon = 0.7$. First, the single path of $c_{11}c_{21}c_{31}$ is easily found as a cluster since those three nodes form a gs-connected cluster, where they all are core nodes satisfying $GS(c_{11}, c_{21}) = 0.9 \geq \varepsilon$ and $GS(c_{21}, c_{31}) = 1.0 \geq \varepsilon$. Next, the path group of $(c_{12}c_{13})(c_{23}c_{24})$ is also found

as another cluster since those four nodes form a gs-connected cluster with satisfying $GS(c_{12}, c_{23}) = 0.8 \geq \varepsilon$, $GS(c_{12}, c_{24}) = 0.85 \geq \varepsilon$, $GS(c_{13}, c_{23}) = 0.9 \geq \varepsilon$, and $GS(c_{13}, c_{24}) = 0.75 \geq \varepsilon$. The remaining nodes $\{c_{22}, c_{32}, c_{33}\}$ are identified as noises due to their low GS scores with their neighborhood. However, if we loosen the threshold ε to 0.6, the path group of $(c_{12}c_{13})(c_{23}c_{24})c_{33}$ is found as a cluster instead of $(c_{12}c_{13})(c_{23}c_{24})$ because $GS(c_{23}, c_{23}) = 0.6 \geq \varepsilon$ and $GS(c_{24}, c_{33}) = 0.6 \geq \varepsilon$. Here, the nodes $\{c_{22}, c_{32}\}$ are still identified as noises, which would be found as a cluster when ε decreases into 0.4. □

Online version of CHRONICLE$_{2nd}$. Although we present CHRONICLE$_{2nd}$ for batch clustering in Algorithm 1, the online version of CHRONICLE$_{2nd}$, *i.e.*, incremental clustering can be easily performed under the same concept. When a new timestamp network G_{t+1} arrives, we perform the 1st-stage clustering CHRONICLE$_{1st}$ for G_{t+1}, and obtain a new partite of T, *i.e.*, T_{t+1} by calculating links and ic-edges within time interval $[t, t+1]$. Then, we perform CHRONICLE$_{2nd}$ on the bipartite graph $\{T_t, T_{t+1}\}$ while maintaining the community ID of each node in T_t. Since CHRONICLE$_{2nd}$ only takes account of nodes and links/ci-edges within time interval $[i, i+1]$ for calculating the similarity, and at the same time, the gs-connectivity is a symmetric relation, this incremental clustering does not hurt the consistency of a new clustering result compared with the past clustering result for $\{T_1, \ldots, T_t\}$. In Example 1, suppose that we perform the initial clustering on the first bipartite $\{T_1, T_2\}$, then incremental clustering on the second bipartite $\{T_2, T_3\}$ under the parameter setting of $\alpha = 0.5$, $\mu = 2$, and $\varepsilon = 0.6$. As a result of the first clustering for $\{T_1, T_2\}$, two clusters $c_{11}c_{21}$ and $(c_{12}c_{13})(c_{23}c_{24})$ are found, which we assign community ID 1 and 2, respectively. As a result of the second clustering for $\{T_2, T_3\}$, two clusters $c_{21}c_{23}$ and $(c_{23}c_{24})c_{33}$ are found, and we already know the community IDs of c_{21} and $(c_{23}c_{24})$ are 1 and 2, respectively, and thus, we finally obtain the same clustering result with that in Example 1 by assigning the community IDs 1 and 2 to c_{23} and c_{33}, respectively.

4 EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness and efficiency of our algorithm CHRONICLE compared with the previous state-of-art t-partite graph based method, the BFS method. We describe the experimental data and environment in Section 4.1, and present the comparison results in Sections 4.2 and 4.3.

4.1 Experimental Setting

We use real dynamic network data set, the DBLP data[3]. We regard authors as nodes, co-authorships as edges, and years as timestamps. We extract the bibliographic information of all journals and conferences of the years from 1993 to 2007 (*i.e.*, 15 years). By filtering authors of low number of publications, we generate four data sets of 15,000 authors, 30,000 authors, 60,000 authors, and 120,000 authors. We call each data sets as DBLP-15K, DBLP-30K, DBLP-60K, and DBLP-120K, respectively.

[3] http://www.informatik.uni-trier.de/~ley/db

In both the CHRONICLE algorithm and the BFS method, the 1st-stage clustering and the 2nd-stage clustering are independent with each other, and the key part of them is the 2nd-stage clustering. Since the quality of the 1st-stage clustering results of the BFS method (*i.e.*, biconnected components) are not as good as those of CHRONICLE$_{1st}$, and at the same time, in order to be fair in our comparison, we use the t-partite graph constructed by CHRONICLE$_{1st}$ as an input t-partite graph for the BFS method. For all the 1st-stage clustering, we use $\mu = 3$ and $\varepsilon = 0.6$.

To compare the efficiency, we measure the elapsed time and memory usage of both CHRONICLE$_{2nd}$ and the BFS method while varying the data size and the number of timestamps, n_{ts}. In case of CHRONICLE$_{2nd}$, the elapsed time and memory usage are not affected by the parameters μ, ε or α. On the contrary, in case of the BFS method, the performance is largely affected by the parameters k or l_{min}. Thus, we also measure the elapsed time and memory usage of the BFS method while varying k and l_{min}. For CHRONICLE$_{2nd}$, we set $\alpha = 0.5$, $\mu = 2$, and $\varepsilon = 0.5$.

The BFS method and CHRONICLE$_{2nd}$ performs some *different mining task* with different purpose for t-partite graph. Moreover, there are no explicit ground truth answers for communities in the DBLP data. Thus, it is very difficult to evaluate the effectiveness by using common measures such as precision/recall. Instead, we measure how many clusters of the BFS method are overlapped with those of CHRONICLE$_{2nd}$, *i.e.*, how much the result of CHRONICLE$_{2nd}$ contains the result of the BFS method. We note that, while the BFS method discovers the exact top-k clusters based on a DP algorithm, CHRONICLE$_{2nd}$ discovers less exact but more various clusters, where the variousness is controlled by ε. Let $\mathcal{CR}_{BFS}$ be a set of clusters of the BFS method, and $\mathcal{CR}_{common}$ a set of clusters of the BFS method overlapped with those of CHRONICLE$_{2nd}$. We measure the ratio between the sizes of two sets, $\frac{|\mathcal{CR}_{common}|}{|\mathcal{CR}_{BFS}|}$ while varying ε and k. In order to show the effectiveness of path group type clusters of CHRONICLE$_{2nd}$, we also measure the average similarity of clusters across time by using Jaccard coefficient for both single path type and path group type while varying ε.

We conduct all the experiments on a Pentium Core2 Duo 2.0 GHz PC with 2 GBytes of main memory, running on Windows XP. We implement our algorithm andthe BFS method in C++ using Microsoft Visual Studio 2005.

4.2 Results of Efficiency

Figures 5∼7 show the result of efficiency evaluation of the BFS method and CHRONICLE$_{2nd}$. In Figure 5, CHRONICLE$_{2nd}$ has the better efficiency than the BFS method as the number of nodes and the number of timestamps increase. Especially, since the BFS method needs to maintain paths of all lengths greater than l_{min}, its performance gets much worse as n_{ts} gets longer as in Figure 5(a). In Figure 6, the performance of the BFS method is much more degraded as k and l_{min} increase due to heavy computation for heaps, whereas that of CHRONICLE$_{2nd}$ is the same regardless of the parameters μ and ε. In Figure 7, the amount of memory usage of the BFS method also largely increases as k and l_{min} get larger due to the increased heap usage, whereas that

of CHRONICLE$_{2nd}$ is the same regardless of μ and ε since it uses only two fixed-size data structures: t-partite graph itself and status list for checking UNCLASSIFIED/NON_MEMBER.

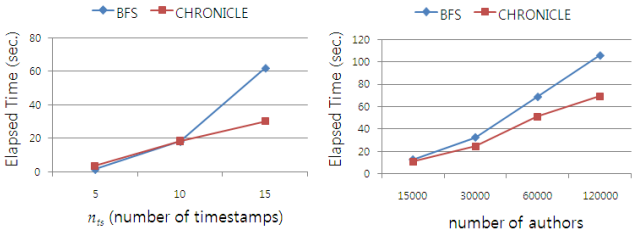


Fig. 5. Elapsed time of varying n_{ts} and the data size ($k = 20$, $l_{min} = 3$).

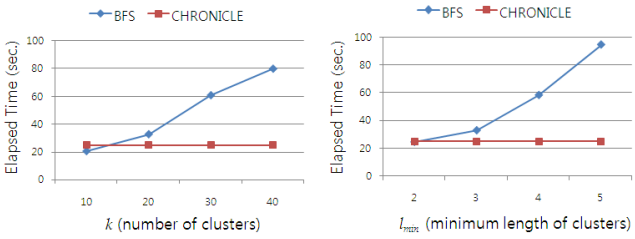


Fig. 6. Elapsed time of varying k and l_{min} (data set = DBLP-30K, $n_{ts} = 10$).

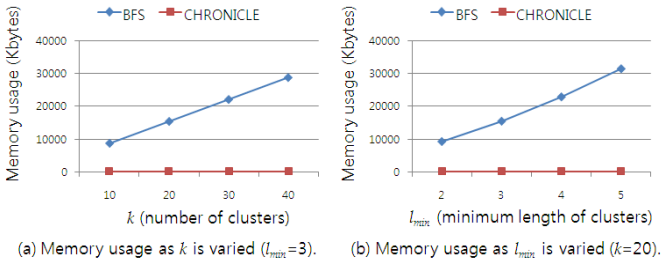
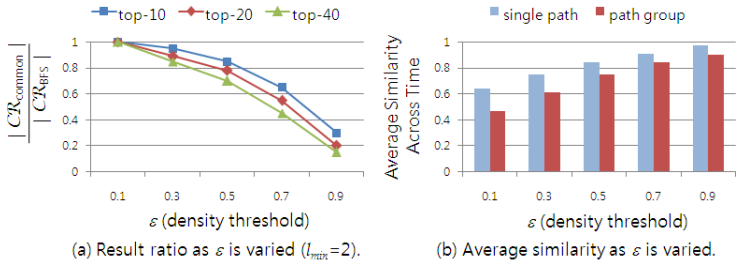


Fig. 7. Memory usage of of varying k and l_{min} (data set = DBLP-30K, $n_{ts} = 10$).

4.3 Results of Effectiveness

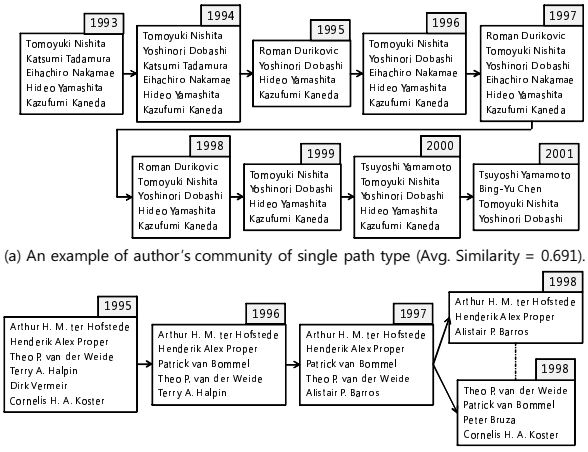
Figure 8(a) shows $\frac{|\mathcal{CR}_{common}|}{|\mathcal{CR}_{BFS}|}$ while varying k of the BFS method and ε of CHRONICLE$_{2nd}$. Even though CHRONICLE$_{2nd}$ discovers a wide range clusters instead of only the most stable clusters, its clustering result includes a fairly high percentage (about 0.7∼0.9) of the clustering result of the BFS method at the moderate density threshold $\varepsilon = 0.5$. Thus, a user can obtain the approximated top-k clusters by sorting the clusters found by CHRONICLE$_{2nd}$. Figure 8(b) shows that not only the clusters of single path type but also the clusters of path group type have a high similarity across time with the similarity becoming stronger as ε increases. This means that the dynamicity of both type clusters is controlled by ε.

Figure 9 shows examples of author's communities of single path type and path group type. Our CHRONICLE algorithm finds dynamic communities of single path type as in Figure 9(a) as well as communities of path group type as in Figure 9(b), whereas the BFS method cannot find both communities.



(a) Result ratio as ε is varied (l_{min}=2). (b) Average similarity as ε is varied.

Fig. 8. The result ratio of $\frac{|\mathcal{CR}_{common}|}{|\mathcal{CR}_{BFS}|}$ and the average similarity across time of clusters of CHRONICLE$_{2nd}$ (data set = DBLP-30K, $n_{ts} = 10$).



(a) An example of author's community of single path type (Avg. Similarity = 0.691).

(b) An example of author's community of path group type (Avg. Similarity = 0.650).

Fig. 9. Examples of author's communities (data set = DBLP-30K).

5 CONCLUSIONS

In this paper, we have proposed a two-stage density-based clustering algorithm, CHRONICLE, that efficiently discovers time-evolving communities over large-scale dynamic networks. By performing the density-based clustering in the 2nd-stage for t-partite graph, CHRONICLE$_{2nd}$ finds both single path clusters and path group clusters, and at the same time, achieves a very high performance compared with the previous BFS method. Through extensive experiments over the DBLP data set, we have shown the effectiveness and efficiency of CHRONICLE, especially, that CHRONICLE$_{2nd}$ is far more scalable than the BFS method with respect to the length of dynamic network and the parameter values.

6 ACKNOWLEDGEMENT

The work was supported in part by the Korea Research Foundation grant funded by the Korean Government (MOEHRD), KRF-2007-357-D00203, by the U.S. National Science Foundation grants IIS-08-42769 and BDI-05-15813, and the Air Force Office of Scientific Research MURI award FA9550-08-1-0265. Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

References

1. S. Asur, S. Parthasarathy, and D. Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. In *Proc. KDD 2007*, pages 913–921.
2. N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa. Seeking stable clusters in the blogosphere. In *Proc. VLDB 2007*, pages 806–817.
3. D. Chakrabarti, R. Kumar, and A. Tomkins. Evolutionary clustering. In *Proc. KDD 2006*, pages 554–560.
4. Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proc. KDD 2007*, pages 153–162.
5. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. KDD 1996*, pages 226–231.
6. T. Falkowski, J. Bartelheimer, and M. Spiliopoulou. Mining and visualizing the evolution of subgroups in social networks. In *Proc. IEEE/WIC/ACM Web Intelligence 2006*, pages 52–58.
7. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, second edition, 2006.
8. J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: A partition-and-group framework. In *Proc. SIGMOD 2007*, pages 593–604.
9. E. A. Leicht, P. Holme, and M. E. J. Newman. Vertex similarity in networks. *Physical Review*, E73:026120, 2006.
10. Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng. FacetNet: A framework for analyzing communities and their evolutions in dynamic networks. In *Proc. WWW 2008*, pages 685–694.
11. Q. Mei and C. Zhai. Discovering evolutionary theme patterns from text - an exploration of temporal text mining. In *Proc. KDD 2005*, pages 198–207.
12. J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. GraphScope: Parameter-free mining of large time-evolving graphs. In *Proc. KDD 2007*, pages 687–696.
13. L. Tang, H. Liu, J. Zhang, and Z. Nazeri. Community evolution in dynamic multimode networks. In *Proc. KDD 2008*, pages 677–685.
14. C. Tantipathananandh, T. Y. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. In *Proc. KDD 2007*, pages 717–726.
15. X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. SCAN: A structural clustering algorithm for networks. In *Proc. KDD 2007*, pages 824–833.
16. L. Zhao and M. J. Zaki. Tricluster: An effective algorithm for mining coherent clusters in 3d microarray data. In *Proc. SIGMOD 2005*, pages 694–705.