Dynamic Community Detection in Weighted Graph Streams*

Chang-Dong Wang†  Jian-Huang Lai‡  Philip S. Yu§

Abstract
In this paper, we aim to tackle the problem of discovering dynamic communities in weighted graph streams, especially when the underlying social behavior of individuals varies considerably over different graph regions. To tackle this problem, a novel structure termed Local Weighted-Edge-based Pattern (LWEP) Summary is proposed to describe a local homogeneous region. To efficiently compute LWEPs, some statistics need to be maintained according to the principle of preserving maximum weighted neighbor information with limited memory storage. To this end, the proposed approach is divided into online and offline components. During the online phase, we introduce some statistics, termed top-$k$ neighbor lists and top-$k$ candidate lists, to track. The key is to maintain only the top-$k$ neighbors with the largest link weights for each node. To allow for less active neighbors to transition into top-$k$ neighbors, an auxiliary data structure termed top-$k$ candidate list is used to identify emerging active neighbors. The statistics can be efficiently maintained in the online component. In the offline component, these statistics are used at each snapshot to efficiently compute LWEPs. Clustering is then performed to consolidate LWEPs into high level clusters. Finally, mapping is made between clusters of consecutive snapshots to generate temporally smooth communities. Experimental results are presented to illustrate the effectiveness and efficiency of the proposed approach.

1 Introduction
In the past few years, a huge amount of graph stream data has been generated, such as social networks, citation networks, and web graph streams. Analyzing these evolving graph streams is a significant research task [3, 2, 5]. Especially, detecting dynamic communities in graph streams enables one to capture natural and social structures present in graph streams [5, 11, 9]. Recently, a growing number of research work have been made on detecting dynamic communities from various perspectives [11, 9, 10, 6, 16, 14]. The temporal smoothness framework that trades off the history quality with the snapshot quality has been investigated in [11, 9]. To address the questions concerning community membership, growth and evolution, Backstrom et al. conducted a case study on two large sources of data [6]. Focusing on identifying communities in a multi-mode network, where both actor membership and interactions can evolve, Tang et al. proposed an evolutionary multi-mode clustering by using the temporal information [16].

In this paper, from a different perspective, we identify the following unaddressed problems associated with dynamic community detection.

1. Weighted graph stream problem: The relationship between a pair of nodes having a large number of interactions is much stronger than the relationship between another pair of nodes having a small number of interactions [4]. This is especially important in dynamic community detection as interactions vary over time. Most of the existing work [9, 10, 6, 1] only consider unweighted edges by ignoring the number of interactions between linked nodes. Additionally, to capture up-to-date community structures, the time-decaying property of the interactions also needs to be considered in the stream scenario. This is because the currently occurred interactions are more informative than the past ones. Therefore, the challenge is that the techniques designed for the unweighted case will not work for the weighted graph problem.

2. Local heterogeneity in spatial-temporal dimensions: The graph structure at each timestamp is influenced by the underlying social behavior over individuals, which vary considerably over different graph regions. Apart from that, the local heterogeneity is also exhibited in temporal dimension due to the evolutionary behavior over the graph stream. Existing approaches attempt to provide a global analysis to determine global communities assuming uniform behavior over the network. In spatial dimension, although a local pattern based method has been proposed in [1] to overcome this, it is restricted to the static graph. In temporal dimension, there is no scheme proposed to adapt to the local heterogeneity. The challenge is that it is non-trivial to extend the method in [1] into graph streams due to the prohibitive computational complexity of constructing local patterns from scratch.

To address the above two challenging problems, we propose a new dynamic community detection algorithm. The basic idea lies in proposing a novel structure termed Local Weighted-Edge-based Pattern (LWEP) Summary to describe a local homogeneous region, which is effective in dealing with locally heterogeneous weighted graph streams. However, the computation of LWEPs requires solving a multiple minimum support frequent pattern mining problem [12].

---

*This work is done during the first author’s internship at University of Illinois at Chicago.
†Sun Yat-sen University, Email: changdongwang@hotmail.com.
‡Sun Yat-sen University, Email: stsljh@mail.sysu.edu.cn.
§University of Illinois at Chicago, Email: psyu@cs.uic.edu.
which is time-consuming. In the weighted graph stream scenario, the problem worsens if we cannot maintain suitable statistics. The principle is that, the statistics should preserve maximum weighted neighbor information with limited memory storage. The min-hash table used in [1] will not work for our weighted graph problem, since it is a uniform sampling, which would not get the neighbors with high weights. Even the weighted min-hash table will not work. This is because the relationship between a projected accumulated decay weight is proposed to estimate the weight score for top-k candidate selection. The rationale is that in order for a previously less active neighbor to move into the top-k neighbor list, it must first show some bursty behavior to move to the front of the top-k candidate list. In the offline component, these statistics are used at each snapshot to create local patterns, e.g., LWEPs. Clustering is then performed to consolidate LWEPs into high level clusters at each snapshot. Finally, mapping is made between clusters of consecutive snapshots to generate temporally smooth communities.

To summarize, our main contributions in this work are:

1. We for the first time identify the problem of dynamic community detection in locally heterogeneous weighted graph streams and propose a novel local pattern structure termed Local Weighted-Edge-based Pattern (LWEP) Summary to deal with such a problem.

2. We propose top-k neighbor lists and top-k candidate lists as statistics to efficiently compute LWEPs. Such statistics which can preserve maximum weighted neighbor information are shown to be very effective in information preservation. Based on these statistics, an effective and efficient approach, termed Top-k ranking, is proposed for dynamic community detection in locally heterogeneous weighted graph streams.

3. We validate our approach via a thorough experimental evaluation on both real and synthetic graph streams.

This paper is organized as follows. The motivation and framework is described in Section 2. In Section 3, the local pattern structure is defined. Section 4 and Section 5 describe respectively the online and offline components. Experimental results are reported in Section 6. This paper is concluded in Section 7.

2 Motivation and Framework

In weighted graph stream, the underlying social behavior of individuals would vary considerably over different graph regions throughout the stream. That is, communities spreading over different graph regions may undergo different amounts of activities. This phenomenon is termed local heterogeneity and graph stream exhibiting local heterogeneity is termed locally heterogeneous weighted graph stream. Since social activities are the major contribution to the forming of communities, the local heterogeneity requires discovering communities of varying densities to reflect the different amounts of activities. This situation is aggravated in graph streams with weighted edges. This is because the relationship between a pair of nodes having a large number of interactions is much stronger than the relationship between another pair of nodes having a small number of interactions. Additionally, to capture up-to-date community structures, the time-decaying property of the interactions also needs to be considered in the stream scenario. For instance, in the example shown in Figure 1(a), there are two communities \( \{1, 2, 3, 4, 5\} \) and \( \{6, 7, 8, 9\} \). The first one is as dense as being a clique, meanwhile the second one is not so dense. The example becomes more challenging if the edge weights are also taken into ac-
count in the dynamic case. In consequence, this paper for the first time identifies the problem of dynamic community detection in locally heterogeneous weighted graph streams.

To adapt to the locally heterogeneous graph stream with weighted edges, we propose the Local Weighted-Edge-based Pattern (LWEP) Summary, which is a set of nodes describing a local homogeneous region. To efficiently compute LWEPs, some statistics need to be maintained according to the principle of preserving maximum weighted neighbor information with limited memory storage. To this end, the proposed approach is divided into online and offline components. In the offline component, the LWEPs are efficiently computed based on top-k neighbor lists and top-k candidate lists. In the offline component, the LWEPs are efficiently computed based on top-k neighbor lists. A local pattern decaying mechanism is introduced to phase out the old LWEPs so as to adapt to the concept drift appearing in graph streams. The LWEPs are then consolidated into high level clusters. Finally, mapping is made between clusters of consecutive snapshots to form communities, which allows anytime community emergence, growth, dissolving, merging and split. In the offline component, snapshots can be taken anytime according to the given task. Like in [11, 9], this paper takes one snapshot between any two consecutive timestamps, and the terms snapshot and timestamp are used alternately.

3 Local Weighted-Edge-Based Patterns

To adapt to the locally heterogeneous graph stream with weighted edges, we propose a novel structure termed Local Weighted-Edge-based Pattern (LWEP) Summary, which is a set of nodes describing a local homogeneous region.

3.1 Definition of LWEP

We start by introducing the incremental representation of weighted graph stream.

**Definition 3.1.** The incremental representation of a graph stream is \( \mathcal{G}_t = (V_t, E_t) \) at timestamp \( t \) consists of the newly attached nodes, \( \mathcal{V}_t = \{v_t|v_t \in \mathcal{V}_0 \cup \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_t\} \) with \( \gamma_{i,j}^{t} = \gamma_{i,j}^{t-1} + \lambda_{i,j}^{t} \sum \lambda_{i,j}^{t} e^{-\lambda (t-t_{i,j})} \) denoting the number of interactions between nodes \( v_i \) and \( v_j \) that occur between timestamps \( t-1 \) and \( t \).

To take into account the varying number of interactions and their time-decaying property, an exponential decaying weight is defined for each edge.

**Definition 3.2.** Assume that interactions between nodes \( v_i \) and \( v_j \) are recorded at timestamps \( t_0 < t_1 < \cdots < t_m \), \( t_0 \) is the timestamp at which \( v_i \) and \( v_j \) began to interact and \( t_m = t \) is the current timestamp. The exponential decaying weight \( w_{t_{i,j}}^{i,j} \) of the edge linking \( v_i \) and \( v_j \) at timestamp \( t \) is defined as

\[
w_{t_{i,j}}^{i,j} = \sum_{l=0}^{m} \gamma_{i,j}^{l} e^{-\lambda (t-t_{i,j})}
\]

where \( \lambda \) is a positive number called the decaying constant. Since \( \gamma_{i,j}^{l} = \gamma_{i,j}^{l-1} \), we have \( w_{t_{i,j}}^{i,j} = w_{t_{i,j}}^{i,j} \).

When new interactions are made to an existing edge, its exponential decaying weight is updated as follows.

**Theorem 3.1.** At timestamp \( t \), for an existing edge with new interactions, its exponential decaying weight \( w_{t_{i,j}}^{i,j} \) that was last updated at timestamp \( t_e \) is updated as

\[
w_{t_{i,j}}^{i,j} = w_{t_{i,j}}^{i,j} e^{-\lambda (t-t_e)} + \gamma_{i,j}^{i,j}
\]

Proof. Assume that until timestamp \( t_e \), there are \( m \) incremental graphs recording interactions between nodes \( v_i \) and \( v_j \). The weight at timestamp \( t_e \) is computed as

\[
w_{t_e}^{i,j} = \sum_{l=0}^{m} \gamma_{i,j}^{l} e^{-\lambda (t_e-t_{i,j})}.
\]

Since \( t_e \) is the last update timestamp before the current timestamp \( t \), then till timestamp \( t \), there are \( m+1 \) incremental graphs recording interactions between nodes \( v_i \) and \( v_j \) with \( t_m = t \). From the definition, we have

\[
w_{t}^{i,j} = \sum_{l=0}^{m} \gamma_{i,j}^{l} e^{-\lambda (t-t_{i,j})} = \sum_{l=0}^{m-1} \gamma_{i,j}^{l} e^{-\lambda (t_e-t_{i,j})} + \gamma_{i,j}^{m} e^{-\lambda (t-t_{m})} = w_{t_e}^{i,j} e^{-\lambda (t-t_e)} + \gamma_{i,j}^{i,j}.
\]

The proof ends.

**Definition 3.3.** The accumulated weighted graph \( \mathcal{G}_t = (V_t, \mathcal{WE}_t) \) at timestamp \( t \) contains the accumulated node set \( V_t \) and the weighted edge set \( \mathcal{WE}_t \) defined respectively as \( V_t = \mathcal{V}_0 \cup \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_t \) and \( \mathcal{WE}_t \) where \( \mathcal{WE}_t = \{(v_i, v_j, \lambda_{i,j}^{t})|v_i, v_j \in V_t\} \).

Figure 1 illustrates the incremental graph stream representation and its relation with the accumulated weighted graph.

Given a weighted universal set consisting of \( n \) objects. Let its weighted subset \( A \) be represented by a vector \( \vec{a} = (a_1, a_2, \ldots, a_n) \). The element \( a_j, j = 1, \ldots, n \), denotes the weight of the \( j \)-th object if \( a_j \neq 0 \), otherwise it indicates that the \( j \)-th object is not contained in \( A \). The weighted Jaccard similarity is defined as follows [7, 8].

**Definition 3.4.** Given two weighted subsets \( A \) and \( B \) (represented as \( \vec{a} \) and \( \vec{b} \) respectively) of a weighted universal set consisting of \( n \) objects, the weighted Jaccard similarity is

\[
WJ(A, B) = \frac{\sum_{j=1}^{n} \min(a_j, b_j)}{\sum_{j=1}^{n} \max(a_j, b_j)}.
\]
Similarly, the multi-way weighted Jaccard similarity of a collection of weighted subsets $A^1, \ldots, A^m$ is defined as

$$WJ(A^1, \ldots, A^m) \triangleq \frac{\sum_{j=1}^{m} \min(a^1_j, \ldots, a^m_j)}{\sum_{j=1}^{m} \max(a^1_j, \ldots, a^m_j)}.$$  

**Definition 3.5.** The weighted neighbor set $\mathcal{WN}(v^j)$ of node $v^j \in V$ is defined as

$$\mathcal{WN}(v^j) \triangleq \{(v^i, w^{i,j}) | v^i \in \mathcal{W}(v^j) \land w^{i,j} \in \mathcal{WE} \}.$$  

Using the vector representation, the weighted neighbor vector $w^{i,j}$ representing the weighted neighbors of node $v^j$ is

$$\overrightarrow{w^{i,j}} = (w^{i,j}_1, w^{i,j}_2, \ldots, w^{i,j}_{|V|})$$

with $w^{i,j}_i, \forall i = 1, \ldots, |V|$ being

$$w^{i,j}_i = \begin{cases} w^{i,j} & \text{if } v^i \text{ and } v^j \text{ are linked} \\ 0 & \text{Otherwise} \end{cases}$$

Here we assume that $(v^i, w^{i,j}) \in \mathcal{WN}(v^j)$ with $w^{i,j} = \max_{(v^i, w^{i,j}) \in \mathcal{WN}(v^j)} \{|w^{i,j}|\}$, i.e., each node has the maximum interactions with itself.

**Definition 3.6.** Given a node set $S = \{v^1, v^2, \ldots, v^m\} \subseteq V$ consisting of $m$ nodes, the weighted edge group affinity is defined as the multi-way weighted Jaccard similarity of their weighted neighbor sets,

$$WJN(S) \triangleq WJ(\mathcal{WN}(v^1), \ldots, \mathcal{WN}(v^m))$$

**Definition 3.7.** For a given node $v^j \in V$, the weighted tail threshold $WT(v^j)$ is defined as the average value of all the weighted edge group affinities for the pairwise sets containing node $v^j$ and each of its neighbors,

$$WT(v^j) \triangleq \frac{\sum_{v^i \in \mathcal{WN}(v^j)} WJN(\{v^j, v^i\})}{|\mathcal{WN}(v^j)|}.$$

The LWEP is a set of nodes defined in terms of both the Jaccard coefficient of the nodes’ weighted neighbor sets and the weighted tail thresholds.

**Definition 3.8. (Local Weighted-Edge-based Pattern) A Local Weighted-Edge-based Pattern (LWEP) of the weighted graph $G_t$ at timestamp $t$ is a set of nodes $P \subseteq V_t$ satisfying (1) $WJN(P) \geq \min_{v^i \in P} WT(v^i)$, and (2) there is no superset $P' \supset P$ such that $WJN(P') \geq \min_{v^i \in P'} WT(v^i)$.

### 3.2 Justification of LWEP

According to Definition 3.6, it is not necessarily true that a node set spreading over a region of higher density would have a larger weighted edge group affinity. It is also not necessarily true that a node set containing nodes with a larger number of links would have a larger weighted edge group affinity. Instead, a node set containing nodes with similar weighted neighbors would have a larger weighted edge group affinity.

For instance, for the node set $\{4, 5\}$ shown in Figure 2, we compute its weighted edge group affinities respectively in different cases. By comparing Figure 2(b) with Figure 2(a), $WJN(\{4, 5\})$ decreases when adding two edges $(3, 4, 0.5)$ and $(3, 5, 0.01)$ of different weights. This is contrary to case if we mistakenly consider the weighted edge group affinity as the link similarity within the set. $WJN(\{4, 5\})$ in Figure 2(c) becomes larger compared with that in Figure 2(a), since the similarity of the neighbor sets of $\{4, 5\}$ becomes larger by adding the same weighted neighbor (i.e., node 3 with the same link weight). An extreme case can be found by comparing Figure 2(d) and Figure 2(e). In Figure 2(d), $WJN(\{4, 5\})$ is as small as 0.0050, although nodes 4 and 5 are directly connected to each other. However, in Figure 2(e), $WJN(\{4, 5\}) = 0.3333$. If nodes 4 and 5 are also directly connected (no matter what number of interactions is), then $WJN(\{4, 5\}) = 1$, reaching the maximum. This means that nodes 4 and 5 have the same weighted neighbors.

According to Definition 3.8, the nodes contained in a LWEP have relatively high weighted edge group affinity compared with the minimum weighted tail threshold. This means they should share similar weighted neighbors, which indicates that the nodes contained in a LWEP are spread over a relatively homogeneous region. This homogeneous region can be either dense or sparse. All the LWEPs together are used to represent the locally heterogeneous weighted graph. By further using LWEPs as the basic elements to generate clusters, the proposed method is capable of adapting to locally heterogeneous weighted graph streams.

### 4 Online Component

Directly maintaining LWEPs at each timestamp is computationally infeasible. As the graph stream progresses, the number of nodes and their weighted neighbor sets increase dramatically. The direct computation of LWEPs requires searching all subsets and their weighted neighbor sets over the entire graph. However, according to Definition 3.8, the computation of LWEPs mainly relies on the neighbors with the highest weights. If we can preserve a small set of highly-weighted neighbors, no significant information will be lost. That is, we have the following neighbor selection principle.

**Proposition 4.1. (Neighbor Selection Principle) Given limited memory storage, we should preserve maximum weighted neighbor information to prevent from losing any significant neighbors.**

To this end, the proposed method is divided into online and offline components. In the online component, the major task is to extract some statistics efficiently at each timestamp, based on which LWEPs are generated in the offline component. The challenge is how to efficiently maintain statistics according to the principle stated in Proposition 4.1.
The min-hash table used in [1] will not work, since it is a uniform sampling, which would not get the neighbors with high weights. Even the weighted min-hash table will not work. This is because as the stream progresses, the weighted sampling would result in constantly changing weighted min-hash tables, leading to time-consuming local pattern reconstruction at each timestamp. To this end, we introduce the statistic structures called top-$k$ neighbor lists and top-$k$ candidate lists. For each node, the top-$k$ neighbors with the largest link weights are put into the top-$k$ neighbor list. Each entry of the top-$k$ candidate list stores one candidate neighbor that is not put into the top-$k$ neighbor list but with potential to reach it later on.

Based on top-$k$ neighbor lists, LWEPS can be efficiently computed in the offline component. However, only maintaining a top-$k$ neighbor list for each node $v$ is not sufficient to capture newly emerged neighbors. It is difficult for a node which is not on the top-$k$ neighbor list to get onto that list as the criterion is based on the accumulated weights. If a neighbor is not on the top-$k$ neighbor list, its past interactions are not tracked and hence cannot be included in the accumulated weight computation. Such neighbor, e.g., denoted as $x$, is mistakenly taken as a new neighbor of $v$, despite that there are interactions occurred between $x$ and $v$ at some past timestamps. At the current timestamp $t$, the edge weight associated with $x$ and $v$ is defined to be its new interaction number by discarding past interactions. The lost of the past interactions obviously violates the neighbor selection principle stated in Proposition 4.1. Therefore, apart from maintaining the top-$k$ neighbor list, a top-$k$ candidate list is also required for each node to identify neighbors with high burst activities at each timestamp. By tracking the weights of the nodes in the candidate list, newly emerged neighbors with high activities can be brought into the top-$k$ neighbor list.

4.1 Top-$k$ Neighbor List Let the top-$k$ neighbor list of node $v^i$ at timestamp $t$ be denoted as $M_t^i$ and the top-$k$ neighbor list set for all nodes be denoted as $M_t$. In the initialization stage, the initial top-$k$ neighbor list $M_0^i$ is generated for each node $v^i$, which is an array consisting of maximum $k$ 3-tuples. Each tuple $M_0^{i,j} = (v^j, w_0^{i,j}, 0)$ contains three entries with the first entry $v^j$ storing the neighbor, the second entry $w_0^{i,j}$ storing the weight of the link edge and the third entry $t = 0$ storing the last timestamp at which the weight is computed.

When a new incremental graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, $t \geq 1$ arrives, for the new nodes $\mathcal{V}_t$, we compute their top-$k$ neighbor lists and append them to $M_{t-1}$ to obtain a temporary top-$k$ neighbor list set $M_t$ of size $|\mathcal{V}_t|$. According to the new interactions, the existing nodes $\mathcal{V}_{t-1}$ are classified into active nodes, semi-active nodes and inactive nodes.

DEFINITION 4.1. The active nodes $\Delta \mathcal{V}_{t-1} \subset \mathcal{V}_{t-1}$ are the existing nodes associated with the new interactions, i.e., $\Delta \mathcal{V}_{t-1} \triangleq \{v^i \in \mathcal{V}_{t-1} \mid \exists v^j \in \mathcal{V}_t, (v^i, v^j, \gamma_{i,j}) \in \mathcal{E}_t\}$.

DEFINITION 4.2. The semi-active nodes $\Theta \mathcal{V}_{t-1} \subset \mathcal{V}_{t-1}$ are the existing nodes, of which some of the top-$k$ neighbors are active, i.e., $\Theta \mathcal{V}_{t-1} \triangleq \{v^i \in \mathcal{V}_{t-1} \mid \exists (v^i, w_{i,j}^{i,j}, t_e) \in M_{t-1}^i, v^j \in \Delta \mathcal{V}_{t-1}\}$.

The remaining nodes $\mathcal{V}_{t-1} \setminus (\Theta \mathcal{V}_{t-1} \cup \Delta \mathcal{V}_{t-1})$ are inactive nodes. For the semi-active and inactive nodes, their top-$k$ neighbor lists are kept unchanged.

We only need to update the top-$k$ neighbor lists of the active nodes $\Delta \mathcal{V}_{t-1}$ as follows. For each active node $v \in \Delta \mathcal{V}_{t-1}$, we first generate the new weighted edges for the new interactions made between a pair of previously unlinked nodes. According to Definition 3.2, the weights for these new edges are defined to be the interaction numbers. For the existing weighted edges linking $v$ and its neighbors, there are two cases. If there are no new interactions made between the pair of nodes associated with the existing edge, the edge’s weight is updated according to Theorem 3.1. If there is no new interaction made between the pair of nodes associated with the existing edge, the edge’s weight is kept unchanged. Finally, the top-$k$ neighbors with the largest edge weights are brought into the top-$k$ neighbor list for this active node.

4.2 Top-$k$ Candidate List Before describing top-$k$ candidate list, let’s describe a more general case—candidate list without memory storage limitation. Each entry of the candidate list stores one candidate neighbor. At timestamp $t$, assume that a new neighbor $x$ for the first time interacts with $v$. If this new neighbor $x$ is not added to the top-$k$ neighbor list of $v$ due to its relatively low link weight, this new neighbor $x$ becomes a candidate. That is, the node $x$, the corresponding link weight and the timestamp $t$, form a 3-tuple which
is added to the candidate list. For the rest of the neighbors that are not added to the top-$k$ neighbor list, we update their exponential decaying weights according to Theorem 3.1 and add the corresponding 3-tuple entries to the candidate list. Or we update the corresponding 3-tuple entries in the candidate list if they are already in the list. At later timestamps, for a "new" weighted neighbor $x$ that is not in the previous top-$k$ neighbor list but in the candidate list, its link weight preserved in the candidate list is used in accumulated weight calculation. In this way, we realize a fair competition in constructing top-$k$ neighbor lists such that no weight is lost.

However, for each node, maintaining such a candidate list for each node would require a huge amount of memory storage. This is because the union of such candidate list neighbor lists is actually equal to storing all the neighbors. This also violates the neighbor selection principle, because we are given limited memory storage. To this end, we introduce a sliding window model with the window parameter $H$ to realize an approximate maintenance of the candidate list. The basic idea is to put only top-$k$ candidates into the candidate list. Therefore, we need a score to rank the candidates in the candidate list according to the neighbor selection principle—preserving maximum weighted neighbor information with limited memory storage. The accumulated decaying weight cannot serve this purpose because the different nodes are put into the candidate list at different timestamps, hence include different number of terms in the accumulated weight calculation. To make it fair, we need to create another measure, referred to as the projected accumulated decaying weight.

The basic idea is that, for the missing term, we take the corresponding term from the bottom node in the top-$k$ neighbor list. Let’s denote the accumulated infinity exponential decaying value as $\theta = \sum_{i=1}^{\infty} e^{-i\lambda} = 1/(e^\lambda - 1)$, and denote the lowest edge weight in the top-$k$ neighbor list at timestamp $t$ as $\omega_j^t$. Assume that a node in the candidate list was put into the list $s(< H)$ timestamps ago, i.e., it only contains at most $s+1$ terms in the accumulated weight calculation (from $t-s$ to $t$). We will assume all missing ($H-s-1$) terms to be $\omega_j^t / \theta$, so the projected accumulated decaying weight is defined to be the current accumulated decaying weight $\omega_j^t / \theta \times \sum_{i=s+1}^{H-s-1} e^{-\lambda(t-H+i)}$. For each node, the candidates that are not put into the top-$k$ neighbor list would be put into the top-$k$ candidate list if their projected accumulated decaying weights are among the top-$k$ ones.

According to the above description, the proposed scheme for maintaining statistics obeys the neighbor selection principle—given the limited memory storage, e.g. $2 \times k$ 3-tuples for each node, maintaining top-$k$ neighbor list and top-$k$ candidate list for each node preserves the maximum weighted neighbor information. We refer to the proposed approach using this ranking scheme to maintain candidate lists as the Top-$k$ ranking approach.

5 Offline Component
In the offline component, the LWEps are created at each snapshot based on the top-$k$ neighbor lists. These LWEps are then consolidated into high level clusters. Finally, mapping is made between clusters of two consecutive snapshots to form communities.

5.1 LWEp Creation At the first timestamp, an initial transaction set $T_0$ is computed by removing duplication and subsets from the initial top-$k$ neighbor list set $M_0$. The initial weighted tail threshold $WT_0(v^i)$ for each node $v^i$ is approximately computed from the top-$k$ neighbor list set $M_0$. After computing the initial transaction set $T_0$ and weighted tail thresholds $WT_0(v^i), \forall v^i \in V_0$, we prove that searching the Local Weighted-Edge-based Patterns (LWEps) can be formalized into the multiple minimum support frequent pattern mining problem [12] restricted to the transaction set.

Theorem 5.1. At timestamp $t$, the Local Weighted-Edge-based Patterns (LWEps), denoted as $L_t$, can be generated by finding any maximum subset $P$ from each transaction $T^i_t \in T_t$ such that (1) $WJN(P^i) \geq \min_{v \in P} WT(v^i)$, and (2) there is no superset $T^i_t \supset P^i \supset P$ such that $WJN(P^i) \geq \min_{v \in P} WT(v^i)$.

Proof. According to the definition of LWEp, one subset $P$ needs to satisfy two conditions in order to be a LWEp. Since the weighted tail threshold of any node is larger than 0, from the first condition, i.e., $WJN(P^i) \geq \min_{v \in P} WT(v^i)$, the precondition for a subset $P$ to be a LWEp is that $WJN(P)$ must be larger than 0. That is, the nodes in $P$ must share at least one common neighbors. According to the definition of transaction, only the subset drawn from transactions shares at least one common neighbors. Therefore, the LWEps can only be selected from the subsets of each transaction. The second condition in this theorem is actually equal to that in the definition. The proof ends.

At later snapshots, the existing LWEps containing only inactive nodes $\nabla_t \setminus (\Theta \nabla_{t-1} \cup \Delta \nabla_{t-1})$ will keep unchanged. That is, given a LWEp $P \in L_{t-1}$ of the previous snapshot, if $P \subset \nabla_t \setminus (\Theta \nabla_{t-1} \cup \Delta \nabla_{t-1})$, then $P$ will remain unchanged. This is because, the computations of the weighted edge group affinity of one local pattern and the nodes’ weighted tail thresholds are only limited to the weighted neighbor sets. These unchanged LWEps are restored in $L_t$ with their timestamps kept unchanged. We only need to recompute the remaining LWEps containing active and semi-active nodes.

For the active nodes $\Delta \nabla_{t-1}$, semi-active nodes $\Theta \nabla_{t-1}$ and the new nodes $\nabla_t$, we first compute new transactions and weighted tail thresholds from the top-$k$ neighbor lists of their union $\Delta \nabla_{t-1} \cup \Theta \nabla_{t-1} \cup \nabla_t$. Consequently, a set of new LWEps $\Delta L_t$ is computed. These newly constructed
LWEPS $\mathcal{L}_t$ are also stored in $\mathcal{L}_t$ with their timestamps set as the current timestamp $t$.

For the existing LWEPs, if they have decayed for a long time that is larger than a prespecified threshold, termed LWEP decaying threshold $\eta$, then they are phased out. The underlying reason is as follows. If one LWEP has not been activated for a long time, the structure of the corresponding graph region is out-of-date. In order to adapt to the change of graph streams, we need to phase out it to allow new LWEPs to take more effect.

### 5.2 Cluster Generation

The consolidation of LWEPS to generate clusters is similar to that in [1], which is proved to be effective for static heterogeneous graphs. It is a two-phase approach. The first phase pieces together local communities in order to create the cores of the locally relevant communities. The second phase then re-constructs these cores in a more comprehensive way with an iterative approach. The major difference is that, rather than fixing the number of clusters as the prespecified number, we adaptively adjust the cluster number in both phases to find appropriate number of clusters. In the first phase, we remove the cores that are completely contained in other cores. In the second phase, a breadth first search is utilized to construct clusters more efficiently, and new clusters are added for the nodes that are not connected to any existing cluster. For completeness, we summarize the cluster generation procedure in Algorithm 1. Like in [1], a small $f_{\text{min}}$ (e.g., 0.02) is used. For detailed discussion, please refer to [1].

As a result, a set of non-overlapping clusters are generated, denoted as $C_t = \{C^1_t, C^2_t, \ldots, C^k_t\}$. Next, we will make mappings between clusters of two consecutive snapshots to form communities.

### 5.3 Community Evolving

Since the main contribution made to form a community is the evolving interactions between pairs of nodes, we realize the mapping by taking advantage of the weighted edges. For any two clusters generated at snapshots $t-1$ and $t$ respectively, we first compute their similarity based on their weighted edges. That is, $\forall C_{t-1}^i \in \mathcal{C}_{t-1}, C_t^j \in \mathcal{C}_t$, their similarity is computed as $\text{sim}(C_{t-1}^i, C_t^j) = WJ(WE_{t-1}^i, WE_t^j)$ where $WE_{t-1}^i$ and $WE_t^j$ are the weighted edge sets of the subgraphs corresponding to the clusters $C_{t-1}^i$ and $C_t^j$ respectively. If the similarity $\text{sim}(C_{t-1}^i, C_t^j)$ is larger than some prespecified threshold (e.g., 0.25, which is used in our experiments), $C_{t-1}^i$ and $C_t^j$ are said to be connected. Otherwise, they are unconnected. In this way, we generate a connection matrix of size $|C_{t-1}| \times |C_t|$ for the cluster sets $C_{t-1}$ and $C_t$.

Based on the connection matrix, we can detect communities that are arbitrarily forming, evolving, and dissolving. Additionally, the evolving of communities may also involve the case where more than one communities (or part of them) are merged to form a new community or one large community splits into more than one communities. Therefore, in the offline component, apart from outputting the clusters generated at each snapshot, we also output one connection matrix reflecting the mapping between the previous cluster set $C_{t-1}$ and the current cluster set $C_t$, according to which the community ID can be correctly assigned.

### 6 Experimental Evaluation

Several experiments are conducted to evaluate the proposed algorithm from three perspectives. The first one is to investigate the effectiveness of the top-$k$ neighbor list in information preservation. Then we compare the clustering performances with the weighted and unweighted edges. Finally, to show the ability of dealing with locally heterogeneous weighted graph streams, we compare the proposed approach with two state-of-the-art algorithms. All the experiments are implemented in Matlab R2009a 64bit edition on a workstation (4 Intel 2.00GHz processors, 4GB of RAM).

#### 6.1 Data Sets and Experimental Settings

Three real-world graph streams are used in our experimental eval-

---

**Algorithm 1 Cluster generation**

1. **Input:** LWEP set $\mathcal{L}_t$, Node set $\mathcal{V}_t$, Top-$k$ neighbor list set $\mathcal{M}_t$, Initial cluster number $n_r$.
2. Take $n_g$ largest LWEPS to form an initial core set $\mathcal{R}$.

**Phase I:**

3. **repeat**
   4. Assign each local pattern $P \in \mathcal{L}_t$ to the core $S \in \mathcal{R}$ with the largest Jaccard similarity $J(P, S)$.
   5. Let $\mathcal{Q}_i \subset \mathcal{L}_t$ be the set of LWEPS assigned to the $i$-th core, $\forall i = 1, \ldots, |\mathcal{R}|$.
   6. Reset the $i$-th core as the nodes occurring in at least $f_{\text{min}} \times |\mathcal{Q}_i|$ patterns of $\mathcal{Q}_i$, $\forall i = 1, \ldots, |\mathcal{R}|$.
   7. **until** $\mathcal{R}$ is unchanged.
   8. Remove overlapping nodes between cores in $\mathcal{R}$, and eliminate the resulting empty cores.

**Phase II:**

9. Initialize clusters as the cores in $\mathcal{R}$.
10. **repeat**
11. Get the unassigned neighbors of the assigned nodes, denoted as $\text{ToBeAssigned}$.
12. **if** $\text{ToBeAssigned} = \emptyset$ **then**
13. Add a new cluster by randomly sampling an unassigned node as the core.
14. **else**
15. Assign each node of $\text{ToBeAssigned}$ to the cluster that shares the maximum number of neighbors.
16. **end if**
17. **until** There is no unassigned node.
18. **Output:** Clusters.
Table 1: Summary of the testing datasets: the overall number of nodes, edges, interactions and timestamps are listed.

<table>
<thead>
<tr>
<th>Name</th>
<th>#Node</th>
<th>#Edge</th>
<th>#Interaction</th>
<th>#Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP</td>
<td>35978</td>
<td>121448</td>
<td>135524</td>
<td>40</td>
</tr>
<tr>
<td>ACM</td>
<td>16839</td>
<td>48562</td>
<td>51887</td>
<td>43</td>
</tr>
<tr>
<td>IBM</td>
<td>2553</td>
<td>5528</td>
<td>10463</td>
<td>35</td>
</tr>
</tbody>
</table>

The first data set is a subset of the DBLP data consisting of 20 conferences from 1969 to 2011, which are AAAI, CIKM, CVPR, ECIR, ECML, EDBT, ICDE, ICDM, ICML, IJCAI, KDD, PAKDD, PKDD, PODS, SDM, SIGIR, SIGMOD, VLDB, WSDM, and WWW. We name it the ACM graph stream. The DBLP graph stream models co-author relationship among researchers, by taking authors as nodes and the number of co-authored papers as the interaction number. Each year is taken as a timestamp. There are 40 timestamps since these conferences were not held in 1970, 1972 and 1974. There are overall 35978 authors and 135524 interactions distributed in 121448 edges.

The second data set is a subset of ACM publication data consisting of 14 top ACM conferences from 1967 to 2010, which are KDD, SIGMOD, WWW, SIGIR, CIKM, SODA, STOC, SOSP, SPAA, SIGCOMM, MobiCOMM, ICML, COLT and VLDB. We name it the ACM graph stream. Similar to the DBLP graph stream, the ACM graph stream also models co-author relationship among researchers. There are in total 43 timestamps since these conferences were not held in 1968. There are overall 16839 authors and 51887 interactions distributed in 48562 edges.

The third data set is another type of data, namely, the IBM sensor data, which is a subset of the sens1 data used in [3]. This data set contains information about local traffic in a sensor network consisting of IP addresses and traffics between pair-wise IP addresses. The IP addresses are taken as nodes, and the traffics between pairs of IP addresses are taken as interactions. The subset contains 2553 nodes and 10463 interactions distributed over 5528 edges. The subset is partitioned into 35 timestamps according to the time-order of the traffics.

The parameters of the proposed approach are set as follows. Top-$k$ neighbor list parameter $K = 25$ in DBLP and ACM graph streams, and $K = 50$ in IBM graph stream, LWEPE decaying threshold $\eta = 3$, window parameter $H = 5$, weight exponential decaying constant $\lambda = 0.5$, and initial cluster number $n_g = 100$. The clustering results generated at each timestamp are compared using an evaluation measure named Normalized mutual information [15].

Normalized mutual information (NMI) is one of the most widely used measures of clustering quality [15, 17, 18]. Given a data set $D$ of size $n$, the clustering labels $\beta$ of $c$ clusters and the other clustering labels $\theta$ of $\hat{c}$ classes, we build a confusion matrix where entry $(i,j)$ defines the number $n_{ij}$ of points in the $i$-th cluster of $\beta$ and the $j$-th cluster of $\theta$. Then NMI can be computed from the confusion matrix [15]

$$NMI = \frac{2 \sum_{i=1}^{c} \sum_{h=1}^{\hat{c}} n_{ih} \log \frac{n_{ih}}{n_{i}^{(h)} \sum_{c=1}^{c} n_{ic}^{(h)}}}{H(\beta) + H(\theta)}$$

where $H(\beta) = -\sum_{j=1}^{\hat{c}} \frac{n_{j\cdot}}{n} \log \frac{n_{j\cdot}}{n}$ and $H(\theta) = -\sum_{j=1}^{c} \frac{n_{\cdot j}}{n} \log \frac{n_{\cdot j}}{n}$ are the Shannon entropy of cluster labels $\beta$ and $\theta$ respectively, with $n_{ij}$ and $n_{ij}^{(h)}$ denoting the number of points in the $i$-th cluster of $\beta$ and in the $j$-th cluster of $\theta$ respectively. A high NMI value indicates that the two clustering labels match well.

6.2 Effectiveness of the Top-$k$ Neighbor List. We first investigate the effectiveness of the top-$k$ neighbor list in information preservation. To this end, the bruteforce approach is taken as the baseline, i.e., using all weighted neighbors rather than top-$k$ weighted neighbors as the statistics.

We run Top-$k$ ranking with $k = 10, 25, 50$ respectively on DBLP and ACM data, and $k = 25, 50, 75$ respectively on IBM data. Figure 3 plots the NMI values (using bruteforce as the baseline) of the clustering results generated by Top-$k$ ranking. Due to the extremely large computational complexity of the bruteforce approach at later timestamps, we only perform bruteforce till timestamp as late as possible. The overall results show a relatively slow performance deterioration caused by using only top-$k$ neighbors. In DBLP graph stream, setting $k = 10$ generates very accurate clustering results with NMI as high as 0.96 at timestamp 12, although the maximum neighbor size is as large as 26 and there are more than 10% of nodes whose neighbor sizes exceed 10. In IBM graph stream, setting $k = 25$ also generates accurate clustering results with NMI as high as 0.95 at timestamp 15, although the maximum neighbor size is larger than 50 and there are more than 20% of nodes whose neighbor sizes exceed 50. Similar results are obtained in ACM graph stream.

As the parameter $k$ increases, i.e., a larger number of weighted neighbors are maintained in statistics, more accurate clustering results can be obtained. For instance, in ACM graph stream, setting $k = 25$ generates almost the same clustering results as that by bruteforce, which is better than setting $k = 10$. Despite that, the clustering results based on top-$k$ neighbor lists with $k$ being much smaller than the maximum neighbor size are acceptable. Additionally, due to the extremely large computational complexity of
bruteforce, the Top-k ranking approach is more attractive than bruteforce. That is, for computing LWEPs at timestamp \( t \), bruteforce needs to solve the multiple minimum support frequent pattern mining problem of scale as large as \( 2^{N_t} \) (i.e., the maximum transaction size is \( N_t \)); meanwhile Top-k ranking only needs to solve problem of scale \( 2^k \) with \( k < N_t \) and \( N_t \) would be as large as 100, leading to \( 2^k \ll 2^{N_t} \). The experimental results validate that the proposed method obeys the neighbor selection principle—preserving maximum weighted neighbor information with limited memory storage.

6.3 Weighted Version vs. Unweighted Version In this subsection, we compare the dynamic community detection approaches with weighted edges and unweighted edges. To this end, we implement a type of Top-k method with unweighted edges, which is called Unweighted top-k. That is, the number of interactions as well as their time-decaying property are discarded in Unweighted top-k. With unweighted edges, the construction of top-k neighbor lists is based on random sampling of the neighbors, although we call it Unweighted top-k.

We first investigate the information loss of the Unweighted top-k approach by using Unweighted bruteforce as the baseline. By Unweighted bruteforce, we mean taking all the unweighted neighbors. Figure 4 plots the clustering accuracy of Unweighted top-k by using Unweighted bruteforce as the baseline. By comparing Figure 3 and Figure 4, it is clear that when setting the parameter \( k \) to the same value, the unweighted version relatively loses more information than the weighted version. When setting \( k = 10 \) in DBLP and ACM graph streams, the weighted version outperforms the unweighted version by getting at least 0.1 higher accuracy. To obtain the accuracy as high as 0.9, the unweighted version requires setting \( k \) as large as 50. From the viewpoint of information preservation, the weighted version outperforms the unweighted version as it can better capture the level of interactions over time.

6.4 Comparison with Other Approaches In this subsection, to show the ability of dealing with locally heterogeneous weighted graph streams, we conduct experiments to compare the proposed approach with the existing algorithms such as FacetNet [11] and the Particle-and-Density based Evolutionary Clustering method (abbr. PDEC) [9].
Under the assumption of local heterogeneity in weighted graph streams, we first compare the clustering accuracy of FacetNet, PDEC and Top-\(k\) ranking in the three graph streams by using brute-force as the baseline. The comparison results are shown in Figures 5(a), 5(b) and 5(c). From the figures, it can be seen that the proposed Top-\(k\) ranking approach consistently generates better clustering results than the compared methods.

Two synthetic graph streams with ground-truth labeling are generated to compare Top-\(k\) ranking with FacetNet and PDEC. We use the data generating method proposed by Newman et al. [13], which has also been used in FacetNet and PDEC. The two graph streams are generated in the same way as [9] except that 1) the duplicated edges are not removed but rather taken as the number of interactions and 2) communities at different regions and timestamps are of varying densities, i.e., exhibiting spatial-temporal heterogeneity throughout the graph stream. Using the same notation as that in [9], the two synthetic graph streams are termed respectively SYN-FIX (dynamic network of a fixed number of communities) and SYN-VAR (dynamic network of a variable number of communities). Please refer to [9] for detailed synthetic data generation.

Figures 5(d) and 5(e) compare the clustering accuracy in the two synthetic graph streams \(w.r.t.\) the ground-truth labeling. It can be seen that the three methods obtain quite stable accuracy with the progression of graph streams. In these two graph streams, the proposed method outperforms the compared methods. Especially, in the SYN-VAR graph stream, the Top-\(k\) ranking method obtains clustering accuracy at least 0.2 higher than the second winner. The reason is that, the two graph streams have varying interaction numbers and exhibit local heterogeneity, and the proposed Top-\(k\) ranking method is capable of dealing with such cases; meanwhile the compared methods are not designed for such cases.

Finally, Figure 6 shows the graph stream processing rate (the number of nodes processed per second) with progression of the five graph streams. Since Top-\(k\) ranking requires some time to initialize top-\(k\) neighbor lists, its processing rate is a little bit lower at initial timestamps. However, once steady state is reached, the processing rate of Top-\(k\) ranking is much higher than PDEC and FacetNet. The reason is that, the online component of Top-\(k\) ranking only needs to update the top-\(k\) neighbor lists and top-\(k\) candidate lists at each timestamp; meanwhile PDEC and FacetNet take some iterations to find clusters trading-off the snapshot cost and the temporal cost.

7 Conclusion and Summary

In this paper, we have presented the first algorithm for dynamic community detection in locally heterogeneous weighted graph streams. A novel structure termed Local Weighted-Edge-based Pattern (LWEP) Summary is proposed to deal with the local heterogeneity in weighted graph streams. To solve the computational challenge in local pattern creation, top-\(k\) neighbor lists are maintained as the statistics in the online component, based on which local patterns can be efficiently created. An auxiliary data structure
termed top-$k$ candidate list is used to allow for less active neighbors to transition into top-$k$ neighbors. Experimental results show that, as the statistics, the top-$k$ neighbor lists and top-$k$ candidate lists are very effective in information preservation. We also compare the results by the weighted and unweighted versions, which shows that the weighted version preserves more information than the unweighted version and generates more meaningful results. When compared with two recent methods, the proposed method is more effective in dealing with locally heterogeneous weighted graph streams. Furthermore, it is also extremely efficient in terms of stream processing rate.

8 Acknowledgments
This work is supported in part by NSFC (Grant No. 61173084 and 61128009), US NSF through grants IIS-0905215, CNS-1115234, IIS-0914934, DBI-0960443, and OISE-1129076, US Department of Army through grant W911NF-12-1-0066, Google Mobile 2014 Program, Huawei and KAU grants.

References


Downloaded from knowledgecenter.siam.org