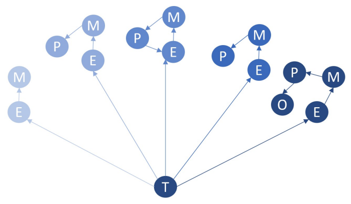
TEMPO

Management and Processing of Temporal Networks

H.F.R.I. Project No. 03480

D4.3: Library for Community detection in MAGMA and Experimental Verification



Computer Engineering & Informatics Department
University of Patras
Greece
30/11/2025

D4.3: Library for Community detection in MAGMA and Experimental Verification

Konstantinos Christopoulos and Kostas Tsichlas

November 30, 2025

Abstract

This report contains the description of our distributed community detection for static historical graphs. They are static, in the sense that the history of the graph is given to us in advance and there is no way to change it. The proposed algorithms are implemented in Scala, and have been tested thorougly on a SPARK platform over a small cluster. The algorithms can be straightforwardly be applied to a dynamic historical graph, provided that they are stored and maintained on our proposed distributed graph database system. We still (until the time of writing) do not have any related experimental results on dynamic historical graphs. Currently, the Computer Engineering and Informatics Department of University of Patras builds a cluster of 28 high-end PCs that will be used for extensive experimentation of the distributed graph database and community detection (as well as, for other algorithms). The library for distributed community detection can be found in [1]. The material of this report is currently under review in a journal.

1 Introduction

Networks are widely used as a powerful tool for data analysis across numerous scientific fields, including social sciences, transportation, and biology. A static network is typically represented as $G = (V, E)$, where V denotes the set of vertices (entities) and E denotes the set of edges (relationships or interactions between entities). Edges may be either directed – such as one person sending an email to another – or undirected – like a collaborative relationship between two colleagues. By incorporating the element of time, we arrive at the concept of static historical graphs, where each node, edge, or attribute is associated with a set of valid time intervals. These graphs are considered static because their structure remains fixed – no nodes or edges are added or removed over time – but the activity of nodes and edges is constrained to specific temporal intervals, hence the term historical. On the other hand, dynamic historical graphs support changes of nodes and edges and their respective time intervals.

Community Detection (hereafter referred to as CD) is a process rooted in graph partitioning [24, 12], aimed at identifying groups of nodes that exhibit high internal connectivity – commonly referred to as communities. The goal is to uncover tightly-knit clusters within a network, with prominent examples including the discovery of user groups in social networks or functional protein complexes in biological networks. Most existing community detection algorithms have been developed for networks consisting of only a few million nodes and edges [28]. However, in recent years, there has been a steady surge in data generated across various sectors such as social media, the Internet of Things (IoT), healthcare, and retail [27]. Representing such vast datasets as networks leads to extremely large-scale graphs, often comprising billions of nodes and connections. For example, as of January 2024, Facebook reported approximately 3 billion active users worldwide[2]. As data volumes continue to grow, so does the demand for scalable computational resources capable of handling these large-scale networks.

Most of the existing community detection algorithms are designed to be used on a single machine. This means that they cannot find communities in large-scale networks, as such networks demand high processing power and memory in order to be analyzed. To deal with these issues, centralized

[1] `https://github.com/kostasada7/Community-Detection-in-Historical-Graphs`

[2] https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

systems are shifted to distributed, decentralized systems, e.g., a peer-to-peer network. An apparent advantage of adopting a distributed system is eradicating the single point of failure compared to a centralized system. For instance, a peer can fail while the service is still available. This means that other peers can inform the requesting entities that a particular peer failed and/or take over the task of the failed peer. One more advantage of utilizing distributed systems is that we can add extra computational resources to the entities in the system in order to scale-up and/or add new entities in order to provide new services in the system.

In many real-world applications, it has often been observed that communities in large networks evolve over time, such that nodes (or communities) exhibit relatively high interaction probabilities during specific time intervals, while interactions across different periods are much less frequent. One such example is the airline route network, which contains flight connections among airports across the globe. Instead of focusing on geographical regions, we consider temporal patterns – such as seasonal schedules or global events – that cause dense connections between certain airports during specific intervals (e.g., holidays, summer travel season). Within each interval, airports may form temporal communities based on synchronized flight schedules, demand surges, or airline partnerships. These time-sensitive communities can change in different periods due to shifts in demand or policy.

One more motivating example could be an IoT network. The IoT is defined as a network of connected devices and end systems that directly interact with each other to collect, share, and analyze important data via the cloud [40]. In such networks, the connections between nodes are not stable but change over time due to factors like mobility, energy constraints, or usage patterns. Community detection over defined time intervals (e.g., hourly, daily) can help uncover persistent communication patterns or emerging structures. For instance, analyzing community membership across time can highlight how clusters of devices collaborate during peak hours or adapt in response to failures or updates. Applying temporal community detection in this way supports better resource management, fault detection, and adaptive routing strategies. Another motivating example is related to tracking potential COVID-19 cases emanating from a single infected person who recently traveled from overseas. In this scenario, community detection can be used to identify a group of individuals who may have been exposed, either directly or indirectly, based on their interactions. We assume the existence of a contact temporal network, which can be constructed using appropriate mobile applications [18]. By applying community detection over specific time intervals – such as daily or hourly contact logs – one can trace how the exposure risk propagates through the network and identify evolving clusters of potential cases.

1.1 Contributions

In this work, we propose distributed methods for community detection in large-scale static historical graphs, focusing on the identification of communities within a user-defined query time interval [3]. Our methods improve upon existing approaches by incorporating temporal information from historical graphs, where each edge is annotated with a validity interval. Below, we summarize the main contributions of our work:

- We are the first to consider the community detection problem in a static historical graph setting, where the history of nodes/edges is represented by time intervals.
- We introduce a temporal extension of the Weighted Clustering Coefficient (WCC) method [29], where key graph concepts are redefined to account for time-dependent activity. A central concept in this extension is the local temporal Clustering Coefficient (ltCC), which generalizes the classical clustering coefficient to capture dynamic neighborhood connectivity.
- Additional graph metrics are revised to maintain consistency with the semantics of temporal graphs.
- We propose two WCC-based variants: One variant explicitly preserves the time intervals of edges (t-iWCC). The other encodes time intervals as edge weights (t-wWCC).
- We develop a second method based on the distributed Label Propagation Algorithm (LPA) [3], with two analogous variants: one that directly processes edge time intervals (t-iLPA), and another that incorporates temporal information through edge weights (t-wLPA).

[3]Code available at `https://github.com/kostasada7/Community-Detection-in-Historical-Graphs`

- We experimentally explore the efficiency and the effectiveness of the proposed methods.

These four algorithms (two WCC-based and two LPA-based) combine structural and temporal information in complementary ways. We also include an existing distributed implementation of the Louvain method [26], which supports weighted edges. All four algorithms are designed to scale efficiently and are capable of processing large-scale temporal graphs, making them suitable for time-aware community detection in both synthetic and real-world datasets.

The structure of the paper is as follows. Section 2 reviews the literature on distributed community detection algorithms while Section 3 introduces definitions and preliminaries. Section 4 presents a temporal CD method based on counting triangles, while Section 5 presents another method for temporal CD based on label propagation. In Section 6 we present and discuss the experimental evaluation of these algorithms.

2 Related Work

We discuss separately distributed community detection algorithms and distributed triangle counting/reporting, since the latter is a crucial component of the algorithms in this paper.

Large-Scale Community Detection The existing literature contains only a few references concerning the distributed detection of communities in dynamic or static graphs. Notably, in the field of historical graphs, a temporal or historical graph is conceptualized as a structure wherein the identification of communities necessitates the discernment of snapshots at specific temporal instances. More precisely, the process of CD in historical graphs is more closely related to what is described in [33] as *instant optimal*. In this case, the scope of algorithms includes those that apply community detection on individual snapshots and strive to identify corresponding communities across these snapshots. Subsequently, we proceed to present literature addressing the distributed aspect and methodologies closely aligned with historical graphs, whether distributed or not.

In [34], a distributed CD method is proposed based on the WCC metric [29]. This method consists of three phases: 1) Preprocessing, which computes the number of triangles for each vertex and removes the edges that do not participate in any triangle, 2) Community Initialization, which resembles a decentralized process of computing the maximal independent set, and 3) WCC Iteration, that iterates over all nodes deciding whether to stay in their current community, transfer to another community or remove themselves from the community creating a singleton community. Iteration ends after a pre-specified number of iterations or when there is no significant change in the WCC score. They implemented the algorithm in Giraph with Hadoop. These results were enhanced in [13], by providing heuristics to count the triangles faster. An extension of this approach concerned an incremental CD algorithm in a distributed environment [1]. They implemented their algorithm using Apache Spark and GraphX in Scala on a multi-cluster environment.

In [25], the authors present an alternative version of the PHASR (Prune-Hash-Refine) method, in a way that is consistent with known distributed models. For this reason, the algorithm is executed using the big data analysis engine Apache Spark. Although most distributed local community detection algorithms aim to discover a subset of edges within the network using some metric, the technique presented in this paper attempts to discover local communities that have the lowest temporal conductivity in a distributed way. The algorithm uses notions like temporal conductivity in order to identify the local communities. Then, they use the Personalized PageRank metric in the refinement step of the algorithm, which is based on random interactions between the nodes. The study in [21] introduces the novel concept of isolated sets to partition graph networks and proposes a parallel version of the Louvain algorithm based on these sets. The approach enables vertex updates and parallel computation without incurring synchronization delays or requiring the exchange of community labels. Another distributed approach is discussed in [16], where they apply static and dynamic community detection on combined datasets, aiming at identifying noteworthy clusters and tracking their evolution over time. To achieve this goal, they examine community detection algorithms specifically tailored for heterogeneous information networks. The historical data undergoes conversion (in snapshots), and the algorithms are subsequently applied to the dataset. More information about the conversion of a temporal graph into a series of snapshots can be found in [2].

An approach that uses local modularity in the LOCAL model is presented in [23]. The main steps of the method are the following: 1) Create a one-node first community, 2) create multiple single-node communities 3) iteratively and distributively extend each community, 4) create new communities for unassigned nodes, and 5) repeat steps (3) and (4) until all nodes have been assigned to a community. A very interesting TLAV (Thinking-Like-A-Vertex) approach is shown in [11]. This is a random walk approach that in a nutshell works as follows: given a node s, it finds its community with local random walks and then iterates over all nodes. They claim that this is a lightweight algorithm. A similar spectral algorithm can be found in [38]. A distributed memory implementation with message passing between processes is given in [15]. Heuristics are provided to speed up the execution time. Label propagation is also used for distributed CD [31]. An example of dynamics that resembles label propagation for distributed community detection is presented in [5]. Finally, DyG-DPCD [35] is a very recent parallel algorithm designed for community detection in dynamic networks, implemented within the Message Passing Interface (MPI) framework. It adopts a vertex-centric model that enables community discovery through local optimization at each node. In addition to the core method, the authors introduce three heuristic enhancements that significantly boost performance without compromising the quality of the detected communities.

A recent survey that contains distributed algorithms can be found in [4]. Based on their taxonomy, the focus of this paper is on self-aggregation and self-organization CD algorithms. An example of such an algorithm is given in [9]. Each node iteratively computes locally an entropy metric and changes its community based on this metric.

Large-Scale Triangle Counting We review relevant literature on triangle counting in both distributed and temporal graph settings. This is particularly important, as one of our methods rely on triangle counting as a core component of its design. By examining prior work in this area, we aim to highlight the challenges and advancements in efficiently counting triangles across large-scale, dynamic, and temporally-evolving networks.

In [17], the authors propose a cloud-edge collaborative framework for distributed triangle counting in graph streams that gathers edges from multiple domains and tags each edge with its respective domain. The master node then performs clustering based on the collected edges and, using the clustering results, distributes the edges to different workers. This enables triangle counting to be carried out in a distributed manner. In [41], a Locally Differentially Private (LDP) method for real-time counting of k-triangles in dynamic social graphs is proposed, marking the first approach to offer edge-level LDP while also ensuring a provably tight upper bound on the estimation error. To further reduce this error, they introduce two enhanced variants that operate without requiring extensive user coordination or synchronization. The core idea behind these methods is to sample one or two disjoint users in the real-time social graph to form wedge structures – key components of k-triangles. This strategy enables accurate triangle estimation while effectively preserving user privacy. Authors in [37] present STEP, a scalable and efficient algorithm designed to approximate temporal triangle counts from a stream of temporal edges. Each temporal edge is valid in a single time instance. STEP integrates a predictive model – estimating the number of triangles each incoming edge may participate in – with a lightweight sampling strategy. This combination enables high scalability and accuracy while efficiently approximating all eight types of temporal triangles in a unified manner.

Finally, the authors of this paper [7] present a distributed algorithm for triangle-based community analysis in historical graphs, focusing on counting triangles within a specified query time interval. Temporal edges are characterized by time intervals (not single time instances). The proposed method introduces a novel approach by incorporating temporal information into triangle counting, enabling the detection of time-aware interaction patterns in dynamic networks. Unlike previous methods, it specifically targets the temporal dimension by counting only those triangles that are active within user-defined time windows. Experimental results on real-world historical datasets demonstrate the algorithm's effectiveness in capturing temporal structures, highlighting its potential to advance temporal graph analysis.

3 Definitions

Let $G = (V_T, E_T)$ be a static historical network. The set of historical nodes V_T consists of nodes paired with their corresponding time intervals, that is, $V_T \subset V \times \mathbb{N}^2$. Similarly, the set of historical edges E_T contains edges together with their time intervals, $E_T \subset E \times \mathbb{N}^2$, where E includes all possible $\binom{|V|}{2}$ undirected edges. We consider the case where each node and each edge has a single valid time interval, although it is straightforward to extend this to multiple valid time intervals. Specifically, every node $v \in V$ (and every edge $e \in E$) has an associated time interval $[t_v^{(s)}, t_v^{(f)}]$ and similarly $[t_e^{(s)}, t_e^{(f)}]$, where (s) and (f) denote the start and finish times, respectively. This interval specifies the exact time range during which the node v (or the edge e) exists. Consequently, if $t \notin [t_v^{(s)}, t_v^{(f)}]$, then v does not exist at time t.

The set $V_{ij} \subseteq V$ contains all nodes whose time interval spans the query interval $[t_i, t_j]$ (analogously, E_{ij} can be defined for edges). By definition, the time interval of each edge is contained within the time intervals of its incident vertices. In cases where multiple time intervals are used, the boundaries of each interval must be defined carefully to avoid overlaps. A common approach is to make intervals open on the left and closed on the right. Following our convention, a single time point t is represented by $(t, t]$.

Assume that $\mathcal{N}_{ij}(v)$ denotes the neighborhood of node v within the query time interval $[t_i, t_j]$. It is possible that $\mathcal{N}_{ij}(v)$ is the empty set for certain query intervals. The routing table $r(v)$ of a node v contains all historical edges from v to other nodes in G. We define $r_{ij}(v)$ as the subset of this routing table that includes only those historical edges whose time intervals intersect the query time interval $[t_i, t_j]$. Equivalently, $r_{ij}(v)$ contains all edges connecting v to nodes that are members of $\mathcal{N}_{ij}(v)$.

Our algorithms are designed having in mind the Bulk Synchronous Parallel (BSP) model, which is a model for designing parallel algorithms that structures computation into a sequence of supersteps. Each superstep consists of three phases: local computation performed independently by each processor (corresponding to any computational entity), communication where processors exchange data, and a global synchronization barrier that ensures all processors have completed their work before the next superstep begins. The model abstracts away low-level details of parallel/distributed hardware, enabling portable and predictable performance analysis based on parameters such as computation time, communication cost, and synchronization overhead.

3.1 Problem Formulation - Methods

We consider the historical graph to be stored in a vertex-centric system [19], which offers space efficiency and enables improvements in both update and query operations. Thus, the complete history of each node, together with its adjacent edges, is maintained within the node itself. Our objective is to support the following query:

CD($G, [t_i, t_j]$): Identify the aggregate communities in graph G in the time interval $[t_i, t_j]$. If $t_i = t_j$, then identify the communities in this time instance.

In case the query is about a particular time instance t_i, then the community detection degenerates to a distributed community detection algorithm [34] on a single instance (snapshot). When an edge or a node is not valid at t_i, then it does not exist in the retrieved snapshot.

Given a query time interval $[t_i, t_j]$ with $t_i < t_j$, one possible approach is to detect aggregated communities within this interval. Alternatively, one may track the evolution of all (or a subset of) these communities over the same period. In the latter case, in addition to identifying the communities and their temporal changes, further challenges emerge regarding the efficient reporting of both the communities and their evolution. Focusing on aggregated communities, which is the scope of this paper, it is essential to account for the contribution of each node and edge according to its overlap with the query time interval. In other words, we must consider that certain nodes or edges may not be valid throughout the entire duration of $[t_i, t_j]$.

For instance, consider a user query requesting the graph partition for the time interval $T = [2, 6]$. This interval contains 5 discrete time instances: $\{2, 3, 4, 5, 6\}$. Within this range, some nodes or edges may be valid for only $1, 2, 3$, or 4 of these instances, rather than for the entire interval. More specifically, suppose an edge e is valid during the time interval $[1, 5]$. Since e is active for 4 out of the 5 time instances in T, we assign it a weight of $\frac{4}{5}$. One of the methods proposed in this paper focuses on how such weight definitions are incorporated when partitioning the graph into communities for the given query interval[4]. The above discussion leads to the following definition.

[4]Our definitions and methods apply also straightforwardly in the case of continuous time intervals.

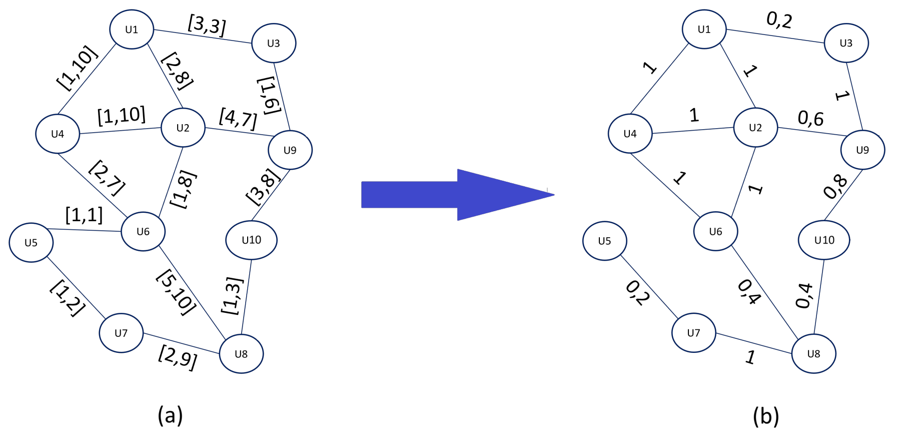


Figure 1: Illustration of graph transformation for a certain time interval. We show only the edges, and we omit the nodes. a) The initial network G with time intervals for each edge, b) The transformed graph G for the query time interval $[2, 6]$, with the corresponding weights on each active edge. If edge weight is equal to 0, then the edge is not shown, e.g., $e(v_5, v_6)$.

Definition 1. *The* ***observed interval*** *of a node/edge is the intersection between the query time interval and the valid time interval of this node/edge.*

If a node or edge is not valid at any time instance within the query time interval, it is considered non-existent for that interval. Under this approach, the unweighted historical graph is converted into a weighted static graph corresponding to the specific query interval. In this weighted representation, the weight assigned to each node or edge is given by the ratio between the length of its observed interval and the length of the query interval.

Definition 2. *The* ***observation ratio*** *of an object (node/edge/triangle) is the ratio between the size of the observed interval of the object and the size of the query time interval defined by the user.*

The observation ratio (weight) reaches its highest value of 1 when the temporal span of a node or edge completely overlaps with the query interval. In contrast, the ratio is 0 when the temporal span of the object does not intersect the query interval at all. Figure 1 presents a temporal network together with the associated observation ratios, only for its edges.

In this work, we propose two variants for modeling and evaluating community structures in temporal graphs: the Interval-based Temporal graph approach and the Weighted-based Temporal Graph approach. The Interval-based Temporal Graph variant is grounded in estimating the contribution of edges and nodes, based on the observed intervals. On the other hand, the Weighted-based Temporal Graph variant is based on the observation ratio by assigning to each edge a weight that reflects its relative importance or strength within the temporal context. Initially, the edge intervals are filtered by checking whether an edge is active within the query time interval. Thus, each edge in the Interval-based approach retains its observed interval, and each edge in the Weighted-based approach retains its observation ratio. If the observed intervals are null, or the observation ratio is equal to 0, then the edge is not considered for community detection. In what follows, we provide necessary metrics for the two proposed variants of the triangle based community detection: the interval-based approach t-iWCC, and the weight-based approach t-wWCC.

Preliminary findings related to the approaches in Sections 3.2 and 4 have been accepted, as short paper, at an upcoming conference [8]; however, the work has not yet been published. The current article substantially extends and formalizes that preliminary version, providing a comprehensive description, analysis, and evaluation.

3.2 Metrics for t-iWCC Variant for Interval-based Temporal-Historical Graphs

The Weighted Clustering Coefficient (hereafter WCC) metric [34] is based on the principle that, within a community, vertices tend to form a denser concentration of triangles among themselves than with vertices outside the community. In this study, we modify and generalize this metric to operate on temporal-historical graphs rather than unweighted static ones. For a given query time interval, the adapted WCC accounts for the specific contribution of each vertex and edge of the historical graph within that period. In other words, we quantify how much each node and edge contributes in the WCC calculation for the specified interval. Following this concept, for a historical graph $G(V_T, E_T)$, we define the cohesion of a vertex x with respect to a set of vertices S over the query interval $[t_i, t_j]$ as follows:

$$WCC_{ij}(x,S) = \begin{cases} \frac{t_{ij}(x,S)}{t_{ij}(x,V)} \times \frac{vt_{ij}(x,V)}{|S\backslash\{x\}|_{ij} + vt_{ij}(x,V\backslash S)} & \text{if } t_{ij}(x,V) \neq 0 \\ 0 & \text{if } t_{ij}(x,V) = 0 \end{cases} \tag{1}$$

The function $t_{ij}(x,S)$ represents the total contribution of edges involved in triangles formed by vertex x with vertices from the set S during the time interval $[t_i, t_j]$. Specifically, consider a triangle consisting of vertices u, v, and x, with edges $e_1 = \left(x, u, t_{e_1}^{(s)}, t_{e_1}^{(f)}\right)$, $e_2 = \left(x, v, t_{e_2}^{(s)}, t_{e_2}^{(f)}\right)$, and $e_3 = \left(u, v, t_{e_3}^{(s)}, t_{e_3}^{(f)}\right)$. The observation ratio for the triangle involving x is defined in Equation (2) as:

$$C_e(x,u,v) = \begin{cases} \frac{|[t_{e_1}^{(s)}, t_{e_1}^{(f)}] \cap [t_{e_2}^{(s)}, t_{e_2}^{(f)}] \cap [t_{e_3}^{(s)}, t_{e_3}^{(f)}] \cap [t_i, t_j]|}{|[t_i, t_j]|} & \text{if } x,u,v \text{ forms triangles} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$t_{ij}(x,S) = \sum_{u,v \in S} C_e(x,u,v) \tag{3}$$

This ratio corresponds to the number of time instances in which the intersection of the edges' temporal intervals and the query interval occurs, divided by the total number of time instances in the query interval. The term $t_{ij}(x,V)$ is defined in the same manner as $t_{ij}(x,S)$. Furthermore, the function $vt_{ij}(x,S)$ measures the total contribution of vertices participating in all such triangles within S. The specific contribution of a vertex u to triangles formed by x in S is given by:

$$C_{vert}(x,u,S) = \frac{\left| [t_i, t_j] \cap \bigcup_{\substack{v \in S \\ (x,u,v) \text{ is a triangle}}} \left([t_{(x,v)}^{(s)}, t_{(x,v)}^{(f)}] \cap [t_{(x,u)}^{(s)}, t_{(x,u)}^{(f)}] \cap [t_{(u,v)}^{(s)}, t_{(u,v)}^{(f)}] \right) \right|}{|[t_i, t_j]|} \tag{4}$$

$$vt_{ij}(x,S) = \sum_{u \in S} C_{vert}(x,u) \tag{5}$$

This value represents the ratio between the size of the union of the time intervals for all triangles—where each triangle's interval is obtained from the intersection of the temporal intervals of its three edges—formed by x and u with vertices $v \in S$ and the query interval, and the size of the query interval itself. Equation (5) then specifies the overall contribution $vt_{ij}(x,S)$ of each vertex in triangles closed by x. The quantity $vt_{ij}(x, V \setminus S)$ is defined in an analogous manner.

Finally, $|S \setminus \{x\}|_{ij}$ is computed using the observation ratio of each vertex in S except x. For a vertex $w \in S$ with valid time interval $[t_w^{(s)}, t_w^{(f)}]$, the contribution of w is defined as:

$$C_S(w) = \frac{\left| [t_w^{(s)}, t_w^{(f)}] \cap [t_i, t_j] \right|}{|[t_i, t_j]|} \tag{6}$$

$$|S_{ij} \setminus \{x\}| = \sum_{w \in S} C_S(w) \tag{7}$$

Equation (7) computes the sum of the observation ratios of all nodes in S, except from x.

In this way, the proposed WCC cohesion metric assigns a higher score when the proportion of closed triangles within the community is large compared to those spanning outside the community (left-hand term), while penalizing vertices in the community that fail to participate in any closed triangle within it (right-hand term).

For a given partition $P_{ij} = \{C_1, C_2, \ldots, C_n\}$ of the vertex set V in G over the query interval $[t_i, t_j]$, the overall WCC_{ij} score corresponding to this partition P is defined as:

$$WCC_{ij}(P) = \frac{1}{|V_{ij}|} \sum_{C \in P} \sum_{x \in C} WCC_{ij}(x, C) \tag{8}$$

Here, V_{ij} denotes the set of nodes whose time intervals overlap with the query interval. For a given partition P, the $WCC_{ij}(P)$ score is computed as the weighted average of the $WCC_{ij}(C)$ scores across all communities C within the partition.

For the purpose of community initialization, we employ the temporal local Clustering Coefficient (tlCC). Although local clustering coefficients have been proposed for temporal graphs, these definitions are often tailored to specific applications (e.g., [10]). We define the tlCC of a node u over the query interval $[t_i, t_j]$ as the ratio of the sum of observation ratios of all triangles involving u to the maximum possible sum of observation ratios that u could achieve. This maximum is determined by retaining all existing edges among u's neighbors with their original temporal intervals, while adding every possible missing edge among these neighbors with a time interval equal to the query interval $[t_i, t_j]$. Since these newly added edges span the entire query interval, their observation ratios are set to 1. The function $t'_{ij}(x, V)$ is defined similarly to $t_{ij}(x, V)$ but refers to this augmented graph including the additional edges. Consequently, the tlCC is formally defined as:

$$tlCC_{ij}(u) = \begin{cases} \frac{t_{ij}(u,V)}{t'_{ij}(u,V)} & \text{if } d_{ij}(u) > 0 \\ \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where $d_{ij}(u)$ represents the weighted degree of node u within the time interval $[t_i, t_j]$, defined as the sum of the observation ratios in $\mathcal{N}_{ij}(u)$.

3.3 Metrics for t-wWCC Variant for Weighted-based Temporal-Historical Graphs

The weighted-based approach is a simplified alternative to the interval-based method. Rather than computing explicit time interval intersections, this method assumes that each edge is annotated with a scalar score and is valid during a specific time interval. For a given query interval $[t_i, t_j]$, we first filter the set of active edges (i.e., those whose validity intervals intersect $[t_i, t_j]$) and then use their associated scores to estimate triangle and vertex contributions in community cohesion.

Let $G(V_T, E_T)$ be a static historical graph, and let s_e be the score associated with edge $e \in E_T$ (s_e is the observation ratio of edge e for the query time interval). The cohesion of a node x to a set of nodes S is defined as:

$$\widehat{WCC}_{ij}(x, S) = \begin{cases} \frac{\widehat{t}_{ij}(x,S)}{\widehat{t}_{ij}(x,V)} \times \frac{\widehat{vt}_{ij}(x,V)}{|S \backslash \{x\}|_{ij} + \widehat{vt}_{ij}(x, V \backslash S)} & \text{if } \widehat{t}_{ij}(x, V) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The function $\widehat{t}_{ij}(x, S)$ denotes the sum of the average scores of the edges to triangles closed by x, with vertices in S in the time interval $[t_i, t_j]$. More precisely, given a triangle with vertices u, v, x, and their edges $e_1 = (x, u, s_{e_1})$, $e_2 = (x, v, s_{e_2})$ and $e_3 = (u, v, s_{e_3})$, the score of the triangle closed by x is defined in Equation (11) as follows:

$$\widehat{C}_e(x, u, v) = \begin{cases} \frac{s_{e_1} + s_{e_2} + s_{e_3}}{3} & \text{if } x, u, v \text{ forms triangles} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$\widehat{t}_{ij}(x, S) = \sum_{u,v \in S} \widehat{C}_e(x, u, v) \tag{12}$$

$\widehat{t}_{ij}(x, V)$ is defined accordingly to $\widehat{t}_{ij}(x, S)$. The function $\widehat{vt}_{ij}(x, S)$ estimates the sum of the scores of the vertices contained in all such triangles in S. The score of a vertex u to triangles closed by x in S, is defined in Equation (13) as follows:

$$\widehat{C}_{\text{vert}}(x,u,S) = \sum_{\substack{v \in S \\ (x,u,v) \text{ is a triangle}}} \left(\frac{s_{e_1} + s_{e_2} + s_{e_3}}{3} \right) \tag{13}$$

$$\widehat{vt}_{ij}(x,S) = \sum_{u \in S} \widehat{C}_{vert}(x,u,S) \tag{14}$$

This is the sum of the average scores of all triangles of x and u with nodes $v \in S$. Then, in Equation (14), the total score $\widehat{vt}_{ij}(x,S)$ of each vertex in triangles closed by x is given. The function $\widehat{vt}_{ij}(x, V \setminus S)$ is defined similarly.

Similarly, the $|\widehat{S} \setminus \{x\}|_{ij}$ is estimated based on the score of each vertex excluding x, in S. Thus, given the vertex $v \in S$, the total score $\forall v \in S$ is defined as follows:

$$|\widehat{S} \setminus \{x\}|_{ij} = \sum_{v \in S} W_v, \quad \text{where } W_v \text{ is the score of vertex } v \tag{15}$$

In the same manner, the $\widehat{WCC}_{ij}$ of a partition P is defined as follows:

$$\widehat{WCC}_{ij}(P) = \frac{1}{|V|} \sum_{C \in P} \sum_{x \in C} \widehat{WCC}_{ij}(x,C), \quad V \text{ is the sum of all score vertices in } G \tag{16}$$

In this case, the temporal local Clustering Coefficient is defined as follows:

$$\widehat{tlCC}_{ij}(u) = \begin{cases} \frac{\widehat{t}_{ij}(u,V)}{\widehat{t'}_{ij}(u,V)} & \text{if } d_{ij}(u) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

$\widehat{t'}_{ij}(u,V)$ denotes the maximum triangle score of node u, where all existing edges retain their actual scores, and all missing edges are assigned a score of 1. This score-based variant provides a natural generalization of temporal clustering and cohesion that accommodates weighted or scored historical graphs, where edge scores may encode strength, frequency, trust, or other relevance metrics over time.

In the following two sections, we present our two proposed methods and their variants. Without loss of generality and for the simplicity of the exposition of the algorithms, we assume that nodes do not carry time intervals. Thus, in what follows, we consider that nodes are active in the entire query time interval, and in this case, the contribution of each node is equal to 1. This is justified, since the valid interval of a node must contain the valid intervals of all adjacent edges.

4 The Distributed Algorithm for Triangle-based Community Detection

In this section, we discuss the two variants of the triangle-based CD method for temporal-historical graphs, t-wWCC and t-iWCC, corresponding to weight-based and interval intersection based semantics, respectively. As the steps are identical for both variants, we present only the interval-based variant. In what follows, we describe the three steps of the proposed distributed community detection algorithms: preprocessing, community initialization, and community refinement via WCC_{ij} iteration. This algorithm is based on the algorithms presented in [30, 34]. In the pre-processing step, we calculate $t_{ij}(x,V)$, $vt_{ij}(x,V)$ and $C_V(x)$ $\forall x \in V_{ij}$, since these quantities do not change throughout the whole computation. In the next step, an initial graph partition is formed, estimating the local temporal Clustering Coefficient (*tlCC*) $\forall u \in V_{ij}$. Based on the initial partition, we improve the WCC_{ij} metric repeatedly in order to find the best vertex movements that can optimize the initial communities. A depiction of the distributed algorithm is shown in Figure 2.

4.1 Preprocessing

During the preprocessing phase, several functions are computed to generate an initial partition of the historical graph and support subsequent steps. Specifically, the functions $t_{ij}(x,V)$, $vt_{ij}(x,V)$,

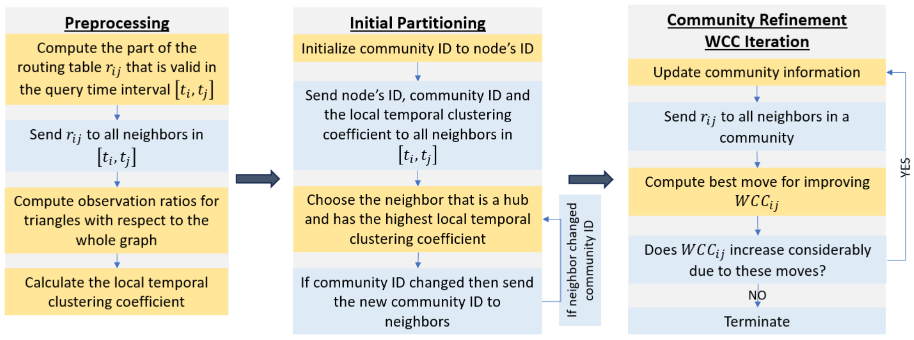

Preprocessing
Compute the part of the routing table r_{ij} that is valid in the query time interval $[t_i, t_j]$
Send r_{ij} to all neighbors in $[t_i, t_j]$
Compute observation ratios for triangles with respect to the whole graph
Calculate the local temporal clustering coefficient
Initial Partitioning
Initialize community ID to node's ID
Send node's ID, community ID and the local temporal clustering coefficient to all neighbors in $[t_i, t_j]$
Choose the neighbor that is a hub and has the highest local temporal clustering coefficient
If community ID changed then send the new community ID to neighbors
If neighbor changed community ID
Community Refinement
WCC Iteration
Update community information
Send r_{ij} to all neighbors in a community
Compute best move for improving WCC_{ij}
Does WCC_{ij} increase considerably due to these moves?
YES
NO
Terminate

Figure 2: A depiction of the three phases of the triangle-based CD method. Yellow boxes correspond to local computation within each box, while blue boxes require some sort of communication between nodes to carry out the related computation.

and $C_V(x)$ are evaluated for every node $x \in V_{ij}$. The value of $vt_{ij}(x, V)$ can be efficiently derived from $t_{ij}(x, V)$, as the latter provides the necessary information to estimate $ut_{ij}(x, v)$. Since these quantities remain constant throughout the process, they only need to be computed once. It is also worth mentioning that prior to preprocessing, the graph is filtered according to the query time interval. This filtering step is necessary to compute either the observation ratio (used in t-wWCC) or the observed interval (used t-iWCC), depending on the specific variant of the algorithm being applied.

Through message passing, every vertex x shares its value $r_{ij}(x)$ with its neighbors in the set $\mathcal{N}_{ij}(x)$. Subsequently, each node determines the intersection of these neighborhoods to identify which nodes form triangles with it. Since the corresponding time intervals are also exchanged, Equations (2) – (5) can be applied to calculate $t_{ij}(x, V)$, $vt_{ij}(x, V)$, and $C_V(x)$ for all nodes $x \in V_{ij}$.

Algorithm 1 Preprocessing Phase. Computation of $t_{ij}(v, V)$, $vt_{ij}(v, V)$ and $tlCC_{ij}(v)$ at node v.

```
1: *Communication 1:*
2:     for all $u \in \mathcal{N}_{ij}(v)$ do
3:         if $\deg(v) < \deg(u)$ then
4:             Send $r_{ij}(v)$ to neighbor $u$
5:         end if
6:     end for
7: *Computation 1:*
8:     Compute $t_{ij}(v, V)$, $vt_{ij}(v, V)$ and $tlCC_{ij}(v)$
```

In the case of large-scale graphs, transmitting $r_{ij}(x)$ from a node to all its neighbors in a single superstep can cause significant delays in communication or even memory overloads that may lead to worker failures. Specifically, vertices with high degrees generate substantially higher communication overheads when sharing their routing tables compared to those with lower degrees. To address this challenge and reduce communication costs, nodes send their routing tables only to neighboring vertices that have a higher degree. This strategy promotes one-way communication from lower-degree vertices to higher-degree ones. As a result, all high-degree neighbors of a node u calculate the total contribution of edges to triangles involving u and then send this aggregated information back to u. Using this approach, the values of $t_{ij}(x, V)$ and $vt_{ij}(x, V)$ for all $x \in V_{ij}$ are efficiently computed.

Algorithm 1 first makes each node send its routing table for the time interval $[t_i, t_j]$ to all its neighbors. As soon as node v has all routing tables of its neighbors, it computes $t_{ij}(v, V)$, $vt_{ij}(v, V)$ and $tlCC_{ij}(v)$.

Algorithm 2 Community initialization process.

```
1: Initialization:
2:     $S \leftarrow V$        ▷ $S$ is used for explanatory purposes to test the termination condition
3: Communication 1:
4:     Each node $v \in S$ sends its {ID, community ID, $tlCC_{ij}$} to its neighbors in $S$
5: Computation 1:
6:     if $v$ has the greatest $tlCC_{ij}$ among its neighbors in $S$ then
7:         $v$ becomes the hub of a new community $C$
8:         $S \leftarrow S \setminus \{v\}$
9:     else
10:        $v$ joins the community $C$ of the neighbor with the highest $tlCC_{ij}$
11:    end if
12: while $S \neq \emptyset$ do
13:     Communication 2:
14:         Each node $v \in S$ sends its {community ID} to neighbors in $S$ with lower $tlCC_{ij}$
15:     Computation 2:
16:         if $v$ has neighbors in $C$ and $v$ is not a hub then
17:             if $v$ has higher $tlCC_{ij}$ than its neighbors in $C$ then
18:                 $v$ remains in $C$
19:                 $S \leftarrow S \setminus \{v\}$
20:             else
21:                 $v$ leaves $C$        ▷ $v$ starts a new singleton community
22:             end if
23:         end if
24: end while
```

4.2 Community Initialization

In this stage, we focus on initializing communities using the local temporal clustering coefficient, $tlCC_{ij}$. The underlying assumption is that a vertex with a higher $tlCC_{ij}$ indicates a stronger likelihood that its neighbors belong to the same community, reflecting the tight connectivity among these nodes.

To obtain the initial partitioning, we adopt an approach inspired by a distributed Maximal Independent Set (MIS) algorithm [14]. The initial partitioning follows these criteria: (a) each community forms a star structure, composed of a central node v (the hub) and a subset of its neighbors $\mathcal{N}_{ij}(v)$ as the periphery; (b) the hub is selected as the node with the highest *tlCC* within the community; and (c) every peripheral node is linked to the hub with the highest *tlCC* among its neighbors. Alternatively, one might simplify initialization by omitting conditions (b) and (c) and directly applying the distributed MIS algorithm. Although this reduces computation time, it can negatively affect the quality of the resulting initial partition.

Initially, each vertex forms its own community using as community ID its own node ID (hub ID). Then, each node sends via message passing its ID, its temporal local clustering coefficient, and its community ID to its neighbors. Consequently, vertices with lower $tlCC_{ij}$ change their community ID and adopt the ID from a vertex that is a hub and has the highest $tlCC_{ij}$ among all its neighbors. To check that a neighbor is a hub, each node simply compares the particular vertex ID with its community ID, and if they are the same then the node is a hub. To impose rule (c), in the next steps, the nodes that changed community ID send their new community ID only to neighbors with lower $tlCC_{ij}$. When adjacent nodes receive the new community ID, they define their communities again based on which of their neighbors have become hubs or belong to the periphery of a hub. For example, suppose that a vertex u changed its community ID in step $i-1$. In step i, node u communicates its updated community ID to its neighbor v, whose $tlCC_{ij}$ is lower. At step $i-1$, v identified its community using the node ID of u. However, by step i, node u no longer belongs to this community, as it resides on the periphery of another community. Consequently, vertex v adjusts its community ID to match its own ID, acknowledging this change. This iterative process persists until no further community adjustments occur within a step.

4.3 Partition Optimization via WCC_{ij}

During this phase, the community partition is refined through an iterative process aimed at maximizing the $WCC_{ij}(P)$ score for a given partition P. The process continues until a user-specified threshold θ is reached, balancing solution quality against computational efficiency. At each iteration, every node evaluates three possible moves regarding its membership in the current community structure. It selects the move that leads to an increase in the overall $WCC_{ij}(P)$ score. The possible moves include:

1. *Stay:* Vertex remains at the current community.
2. *Remove:* Vertex is removed from the current community and forms a new singleton community.
3. *Transfer:* Vertex is removed from its current community and joins another community.

To determine the optimal move for every node $x \in V$, all calculations are performed in parallel. Below, we outline the three scenarios that must be examined to identify the best move for each vertex. Equations (18) to (20), originally established and validated in [30], have been adapted here to suit the specific requirements of temporal (historical) graphs.

1. **Case 1:** Let $P = \{C_1, ..., C_n, \{x\}\}$ and $P' = \{C'_1, C_2, ...C_{n-1}, C_n\}$, be two partitions of the historical graph $G = (V_T, E_T)$, and let $C'_1 = C_1 \cup \{x\}$.

   If $WCC_{ij}(P') - WCC_{ij}(P) > 0$, then the vertex x leaves its singleton community and is inserted into C_1, creating the new community C'_1. Otherwise, vertex x remains **isolated**, as a singleton community. The equation presented above can be expressed in the following manner:

$$WCC_{ij}(P') - WCC_{ij}(P) = \frac{1}{|V_{ij}|}\left(\sum_{y \in C'_1} WCC_{ij}(y, C'_1) - \sum_{y \in C_1} WCC_{ij}(y, C_1)\right) =$$
$$= \frac{1}{|V_{ij}|}\left(\sum_{y \in C_1} WCC_{ij}(y, C'_1) + WCC_{ij}(x, C'_1) - \left(\sum_{y \in C_1} WCC_{ij}(y, C_1) + \cancelto{0}{WCC_{ij}(x, \{x\})}\right)\right) =$$
$$= \frac{1}{|V_{ij}|}\left(\sum_{y \in C_1} WCC_{ij}(y, C'_1) + WCC_{ij}(x, C'_1) - \sum_{y \in C_1} WCC_{ij}(y, C_1)\right) = W_{\mathbf{iso}}(x, C_1) \tag{18}$$

   where $W_{\mathbf{iso}}(x, C_1)$ represents the change in the score when node x that constituted a one-node community moves to community C_1. Analogously to the preceding proof, the subsequent two cases are established in a similar manner. Therefore, for the sake of brevity, we state only the results.

2. **Case 2:** Let $P = \{C_1, ..., C_n\}$ and $P' = \{C'_1, C_2, ...C_{n-1}, C_n, \{x\}\}$, be two partitions of the historical graph $G = (V_T, E_T)$, and $C_1 = C'_1 \cup \{x\}$.

   If $WCC_{ij}(P') - WCC_{ij}(P) > 0$, then the vertex x is removed from its current community and forms its own singleton community. Otherwise, vertex x remains at its current community C_1. The equation presented above can be expressed in the following manner:

$$WCC_{ij}(P') - WCC_{ij}(P) = -W_{\mathbf{iso}}(x, C'_1) \tag{19}$$

3. **Case 3:** Let $P = \{C_1, ...C_{n-1}, C_n\}$ and $P' = \{C'_1, C_2, ...C_{n-1}, C'_n\}$, be two partitions of the historical graph $G = (V_T, E_T)$, where $C_1 = C'_1 \cup \{x\}$ and $C'_n = C_n \cup \{x\}$

If $WCC_{ij}(P') - WCC_{ij}(P) > 0$, then the vertex x is moved from community C_1 to community C_n. Otherwise, vertex x remains in the current community. In a manner analogous to **Cases** 1 and 2, we have:

$$WCC_{ij}(P') - WCC_{ij}(P) = -W_{\mathbf{iso}}(x, C'_1) + W_{\mathbf{iso}}(x, C_n) \tag{20}$$

Algorithm 3 Partition optimization process. Note: Any global aggregation (e.g., $WCC_{ij}(P')$) is described using a conceptual master for explanatory purposes and as a termination check; the process is otherwise fully distributed. We assume a termination threshold θ. In Line 4, the mixed step is an abstract description of the process of each node for choosing the best movement. It contains a communication step and a computation step (mixed step).

```
1:  *Initialization:*
2:      Initialize $converged \leftarrow$ false
3:  **while** not $converged$ **do**
4:      *Mixed 1: Local movement selection*
5:          Each node evaluates possible movements: *Stay*, *Remove*, or *Transfer* , stores its best
        movement in *b_ move* and update community memberships.
6:          Each node computes $WCC_{ij}(y, C)$ for its (best movement) current community $C$
7:      *Communication 2: Local community quality*
8:          Each node sends $WCC_{ij}(y, C)$ to the workers responsible for $C$
9:      *Computation 2: Community aggregation*
10:         For each community $C$, aggregate $\sum_{y \in C} WCC_{ij}(y, C)$
11:     *Communication 3: Global quality estimation*
12:         Aggregated community values are exchanged to compute $WCC_{ij}(P')$ for the entire graph
        partition
13:     *Computation 3: Termination check*
14:         **if** $\frac{WCC_{ij}(P') - WCC_{ij}(P)}{WCC_{ij}(P)} < \theta$ **then**
15:             $converged \leftarrow$ true
16:         **else**
17:             Update $P \leftarrow P'$ and continue
18:         **end if**
19: **end while**
```

As can be easily inferred from the above cases, we do not compute $WCC_{ij}(P')$ and $WCC_{ij}(P)$ from scratch. In contrast, we should calculate the $WCC_{ij}(y, C)$ and $WCC_{ij}(y, C')$ (utilizing Algorithm 1) for a limited number of communities, focusing on those communities to which the neighbors of y belong but differ from the community of y. In addition, all the above cases must be calculated for all nodes in V_{ij}, and all the best moves for all vertices are stored within each node and applied simultaneously. Given the new partition after one iteration, we should check whether the termination condition is valid. If the improvement in global quality exceeds a predefined threshold θ, $\frac{WCC_{ij}(P') - WCC_{ij}(P)}{WCC_{ij}(P)} \geq \theta$, indicating a significant change in the community structure, the process continues with another iteration. Otherwise, if the improvement is less than θ, the process terminates, and each node retains the community ID to which it belongs.

Calculating the exact value of WCC_{ij} is computationally intensive, as it requires triangle counting for every vertex – an especially costly operation for high-degree nodes. Since this computation is repeatedly performed during the algorithm's iterations, it becomes a significant performance bottleneck. To mitigate this, we adopt an approximation proposed in [30] (see Equation 21), which estimates the WCC_{ij} gain in constant time using simple community-level statistics: size (r), density (δ), boundary edges (b), and a fixed global clustering coefficient (ω). This approach significantly reduces computation time without sacrificing accuracy.

$$WCC_{ij}(P') - WCC_{ij}(P) = WCC'_{ij}I(v, C) = \frac{1}{V} \cdot (d_{in} \cdot \Theta_1 + (r - d_{in}) \cdot \Theta_2 + \Theta_3), \tag{21}$$

where,

$$\Theta_1 = \frac{\big((r-1)\,\delta + 1 + q\big)\,(d_{\text{in}}-1)\,\delta}{(r+q)\,\Big((r-1)(r-2)\,\delta^3 + (d_{\text{in}}-1)\,\delta + q(q-1)\,\delta\,\omega + q(q-1)\,\omega + d_{\text{out}}\,\omega\Big)}, \tag{22}$$

$$\Theta_2 = -\,\frac{(r-1)(r-2)\,\delta^3}{(r-1)(r-2)\,\delta^3 + q(q-1)\,\omega + q(r-1)\,\delta\,\omega} \cdot \frac{(r-1)\,\delta + q}{(r+q)\,(r-1+q)}, \tag{23}$$

$$\Theta_3 = \frac{d_{\text{in}}\,(d_{\text{in}}-1)\,\delta}{d_{\text{in}}\,(d_{\text{in}}-1)\,\delta + d_{\text{out}}\,(d_{\text{out}}-1)\,\omega + d_{\text{out}}\,d_{\text{in}}\,\omega} \cdot \frac{d_{\text{in}} + d_{\text{out}}}{r + d_{\text{out}}}, \tag{24}$$

$$q = \frac{b - d_{\text{in}}}{r}. \tag{25}$$

In this context, the variables have been adapted to accommodate both the interval-based and weighted-based variants. As previously mentioned, in the interval-based approach each edge's contribution score is obtained by dividing its valid interval by the query time interval, whereas in the weighted-based approach, each edge has a score. Accordingly, the internal and external degrees, d_{in} and d_{out}, are computed for each node based on the edge scores. Specifically, d_{in} refers to the number (or total score) of edges that connect the node to other nodes within the same community, while d_{out} represents the edges that connect the node to nodes outside its community.

The parameter b denotes the community's total external-edge weight score (i.e. edges connecting community members to nodes outside the community). When a vertex v is removed, its outgoing-edge weight score is subtracted from b and its internal-edge weight score is reclassified, since they no longer contribute to the internal total.

Since we do not assume time intervals on the graph nodes, r denotes the total number of nodes in the current community. The edge-density δ measures how tightly the nodes with total score r are interconnected by normalizing the total internal-edge weight a against the maximum possible number of ordered node pairs $\delta = \frac{2a}{r^2}$. Finally, after computing the temporal local clustering coefficient tlCC(v) for each vertex v, the global clustering coefficient ω is obtained as the average of these values.

4.4 Discussion on the Efficiency

Regarding the efficiency of the three-phase distributed algorithm, we provide a crude estimate in the BSP model, aiming at highlighting its bottlenecks. In [30], although the method is not inherently centralized, the authors provide a rough estimate of its efficiency by assuming a centralized setting. Assume that $n = |V_T|$ and $m = |E_T|$ are the total number of nodes and edges, respectively, in G. Let $n_{ij} = |V_{ij}|$ be the number of nodes that have at least one edge active in the query time interval $[t_i, t_j]$, and similarly, $m_{ij} = |E_{ij}|$ be the number of edges in this interval. Assume that $d_{ij} = m_{ij}/n_{ij}$ is the average degree of each vertex in G_{ij}. Assume also that the number of processors (executors) is p. The graph G is partitioned into p processors. For the query time interval $[t_i, t_j]$, assume that the edge cut imposed by this partitioning of G_{ij} between the p processors is c_{ij} and its size (in terms of number of edges) is $k_{ij} = |c_{ij}|$. Finally, assume that g is the cost for sending or receiving a message. For simplicity, we assume, similarly to the LOCAL distributed model, that g is the cost for communicating a message irrespective of its size. g is larger than 1 and is defined in terms of the unit computation time. This may cause some considerations regarding the communication of routing tables, which are not of constant size, but it will be enough for our crude analysis. We will not consider the cost of the barrier synchronization that depends on various parameters of the network.

The filtering phase is performed in a single computation superstep and takes $O\left(\frac{n+m}{p}\right)$ time, as it does not involve any communication between processes. In the preprocessing phase, we calculate the contribution of each edge and vertex to the formed triangles. Initially, all nodes v communicate their routing tables $r_{ij}(v)$. Summing over all nodes in V_{ij}, the total communication cost is proportional to the edge cut of the partitioning of G_{ij} of the p processors, which is $O(k_{ij})$. The local computation per node needed to intersect the sorted routing tables in order to find the triangles is $O(d_{ij} \cdot d_{ij})$, since each node has an average degree of d_{ij} and its routing table has this size as well. Thus, the aggregate local computation cost per processor is $O\left(d_{ij}^2 \frac{n_{ij}}{p}\right)$. Consequently, we need $O(n_{ij}/p)$

per processor to estimate the contribution of edges and vertices for each triangle and compute all quantities. Thus, the total complexity for the preprocessing phase, consisting of a single superstep, is $O\left(k_{ij} + d_{ij}^2 \frac{n_{ij}}{p}\right)$.

For the community initialization, the number of supersteps required is $O(\log n_{ij})$, assuming that the efficiency of the proposed initialization method resembles that of MIS. In fact, the proposed method is a biased MIS that tends to choose higher weight ($ltCC$) nodes for the independent set. Initially, we need $O(d_{ij})$ time to compute $ltCC_{ij}$ $\forall v \in V_{ij}$, since at the preprocessing step all necessary calculations have been made. This contributes an $O(m_{ij}/p)$ local computation cost per processor only once at the first iteration. Then, in each iteration, each node sends its community ID along with other $O(1)$ information to all neighbors, which contributes a communication cost of $O(k_{ij}g)$ since it depends on the edge cut between the subgraphs stored in each processor. The local computation within each node is then equal to $O(d_{ij})$, since the community ID with the largest $ltCC$ must be chosen. Thus, the aggregate cost of the community initialization phase is $O\left(\log n_{ij}\left(\frac{m_{ij}}{p} + k_{ij}g\right)\right)$.

Lastly, in the partition refinement phase, we assume that the number of iterations is a complexity parameter ℓ – it doesn't seem easy to express ℓ as a function of properties of the graph, and it has not been done even in a centralized setting. A similar assumption was made also by [30], which was experimentally backed up as it is in our case. In each iteration, $WCC_{ij}(C)$ must be computed for all possible movements and for all nodes in V_{ij}. In the worst case, for a vertex v, the $WCC_{ij}(C)$ could be estimated for $3d_{ij}(v)$ potential movements, assuming that each of its incident edges connects to different communities. Thus, for n_{ij} vertices we need $O(k_{ij}g)$ total communication in the worst case between processors, and $O(m_{ij})$ local computation distributed to the p processors, leading to a total complexity of $O\left(\ell\left(\frac{m_{ij}}{p} + k_{ij}g\right)\right)$.

These rather informal but informative arguments lead to a total complexity of

$$O\left(\frac{n+m}{p} + k_{ij} + d_{ij}^2\frac{n_{ij}}{p} + \log n_{ij}\left(\frac{m_{ij}}{p} + k_{ij}g\right) + \ell\left(\frac{m_{ij}}{p} + k_{ij}g\right)\right)$$

simplified based on the definition of d_{ij} to $O\left(\frac{m_{ij}^2}{pn_{ij}} + (\log n_{ij} + \ell)\left(\frac{m_{ij}}{p} + k_{ij}g\right)\right)$. The main difference between the two variants is located in the first term, where in the case of t-wWCC it is $O(m_{ij}/p)$.

5 Temporal Label Propagation Algorithm

The temporal Label Propagation Algorithm is an efficient, distributed method for detecting community structures in graphs. Similarly to the previous section, we present two variants, one based on weights and one based on the intersections of the intervals. The former generates edge weights that reflect the temporal overlap, guiding the propagation process based on total weights. The latter preserves and uses the overlapping intervals by employing their intersection to guide the propagation process.

5.1 temporal weighted LPA: t-wLPA

In this temporal label propagation variant, we first filter the edges based on the query time interval $[t_i, t_j]$. For each edge e with valid time interval $[t_e^{(s)}, t_e^{(f)}]$, we compute its observation ratio, and only edges with a non-zero observation ratio are retained. Then, the algorithm follows the standard process of a label propagation algorithm. Initially, each node in the graph is assigned a unique label, typically its own identifier. These labels represent tentative community memberships. After initialization, we move to the iterative process of label propagation. Based on experimental evaluation, we determined a fixed number of iterations that provides stable label assignments while maintaining low computational cost. Over this fixed number of iterations, labels are updated in parallel according a simple rule applied on the observation ratio of the edges as discussed below.

In each communication round, nodes exchange only their current labels. However, the label is transmitted together with the corresponding edge weight to explicitly associate the received label with the specific edge-weight it came from. This ensures correct computation during label updates. In this way, each node collects its incoming messages and sums the weights per received label. Let

$w_v(l)$ denote the total weight contributed by neighbors of v holding label l. Node v updates its label to the one with the maximum weight. In case of a tie, the label is selected randomly.

This process leverages a Pregel-style bulk-synchronous parallel model: at each superstep, nodes exchange messages, aggregate them, and update labels simultaneously. After the final iteration, nodes sharing the same label form a community. While LPA does not guarantee convergence to a unique partition, in practice, it rapidly identifies dense subgraphs.

5.2 temporal LPA with Interval Intersection: t-iLPA

In this variant of temporal Label Propagation, rather than collapsing all edge intervals into a single score (the observation ratio), we retain the full set of edge intervals that overlap with a user-specified query time window $[t_i, t_j]$. More precisely, we keep only the edges whose *observed interval* (Definition 1) is non-empty; that is, there exists a non-zero intersection between the edge's active lifespan and the query interval. This constitutes the filtering phase of the algorithm.

After preprocessing, the label propagation proceeds for a fixed number of iterations. Initially, each node is assigned a unique community label, typically equal to its own vertex ID. During each iteration, each node sends its current label to all neighbors, along with the *observed interval* that corresponds to the edge over which the label is being propagated. This allows the recipient to associate the label not only with its identity but also with the specific time span in which the relationship was active.

Upon receiving label-interval pairs from its neighbors, each node aggregates the incoming messages by community label and collects the list of corresponding observed intervals for each label. The aggregation step collects these messages by grouping the received time intervals according to each community label. In other words, for each vertex, we create a map where the keys are the different labels observed among its neighbors, and the values are the lists of time intervals corresponding to edges connecting to vertices in those labels. Thus, for each distinct label received, the node computes the *sum of the sizes of all pairwise intersections* among the associated intervals. That is, for a given label, all pairs of intervals are considered. This pairwise intersection sum serves as the label's temporal score.

The node then updates its own label to the one with the highest total pairwise intersection score. In case of a tie between multiple labels, a random tie-breaking rule is applied. This process is repeated for a fixed number of iterations. After thorough experimentation, in both LPA variants, and evaluation of the trade-off between computational cost and efficiency, the optimal number of iterations was determined to be four.

Algorithm 4 Label propagation with interval intersection (t-iLPA).

```
1: Initialization:
2:     Assign each node $v \in V$ an initial community ID $c_v \leftarrow v$
3:     Set iteration counter $iter \leftarrow 0$
4: while $iter < \ell$ do
5:     $iter \leftarrow iter + 1$
6:     Communication 1: Label messages
7:         Each node $v$ sends to each neighbor $u$ the pair $(c_v, I_{vu})$, where $I_{vu}$ is the interval of edge $(v, u)$
8:     Computation 1: Interval-based label scoring
9:         Each node $v$ groups received intervals by label: $\{(c_k, [I_1, I_2, \ldots])\}$
10:        for all labels $c_k$ do
11:            Compute total pairwise intersection $score(c_k)$
12:        end for
13:    Computation 2: Label update
14:        $c_v \leftarrow \arg\max_{c_k} score(c_k)$        ▷ Each node $v$ updates its label breaking ties randomly
15: end while
```

5.3 Comparison of LPA Variants

The foundational paper [31] introduced the label propagation method for community detection, claiming a near-linear time complexity based only on extensive experimental evaluation. We are going to discuss succinctly the complexity of the proposed variants of LPA in the BSP model using the same definitions as in 4.4.

Initially, the filtering step requires one computation superstep, which is carried out in $O\left(\frac{n+m}{p}\right)$ time, since no communication is necessary. The following supersteps correspond to iterations of the LPA algorithm. In our case, the number of iterations is fixed; assume it is equal to ℓ. In each iteration, each node sends its label to all its neighbors along with the weight or the interval of the corresponding edge. The cost of this is equal to the size k_{ij} of the edge cut c_{ij} of the graph G_{ij}, which is $O(k_{ij}g)$. During each iteration, each processor computes the new labels of its assigned nodes. t-iLPA has a different behavior than t-wLPA in the worst case. In t-wLPA, the sum of all weights related to each label is required. This is carried out in time proportional to the degree of each node, contributing a total of $O(m_{ij}/p)$. For t-iLPA, all pairwise overlaps of the intervals corresponding to the same label must be computed within each node. This means that each node v with degree $d_{ij}(v)$, requires $O(d_{ij}^2(v))$ computation time. In the worst case, this may lead to $O(m_{ij}^2/p)$ local computation time.

In total, the worst-case complexity of t-wLPA is $O\left(\frac{n+m+\ell m_{ij}}{p} + \ell k_{ij}g\right)$, while for t-iLPA is $O\left(\frac{n+m+\ell m_{ij}^2}{p} + \ell k_{ij}g\right)$. The two variants differ asymptotically only on the computation side, while the communication cost is similar. Finally, regarding the fixed number ℓ of iterations, it is well known that LPA may never converge to a steady state since periodicity may come up (e.g., in the case of bipartite graphs). Analysis related to the number of iterations, require assumptions that are beyond the scope and aim of this paper.

6 Experiments

We conducted three sets of experiments to evaluate our methods for different scenarios. In the first set, we imposed a uniform, fixed time interval on the edges of four real-world networks – each lacking native temporal annotations – and assessed community detection accuracy against available ground-truth partitions. Our goal in this set was to assess the quality of the results of the algorithms and, at the same time, look at their efficiency in an extreme scenario where the whole graph is queried. The second set augmented a single large real graph dataset with synthetic timestamps to create meaningful intervals and focused exclusively on measuring runtime, as no ground truth existed for the simulated temporal graph. Our goal was to showcase that although the graph is massive, a small query time interval allows the efficient identification of communities. Finally, the third set employed a synthetic graph generator to produce three networks with known community assignments at discrete snapshots (but without continuous intervals); here, we applied some modifications and evaluated their performance on these instantaneous views, aiming at evaluating the effectiveness of our methods.

To evaluate how good the community detection results are, we use the *Normalized Mutual Information* (NMI) [20], which is grounded in information theory. NMI quantifies the similarity between two partitions of a dataset and yields a value in the range $[0, 1]$, where a score of 1 indicates perfect alignment between the two partitions. It has been extensively used in the community detection literature [22, 33, 36].

The experimental platform consisted of an Apache Spark 3.5.3 environment deployed on a small-scale Kubernetes cluster [6]. To create a robust and challenging test-bed, the cluster's hardware was intentionally heterogeneous, comprising several smaller commodity PCs alongside a single, more powerful machine with server-grade components. For processing, the Spark application was configured with 4 executor pods, to which Kubernetes allocated 6 CPU cores and 30 GB of RAM each. This mixed-hardware architecture leveraged Kubernetes for dynamic resource orchestration, load balancing, and fault tolerance, thereby enabling a thorough assessment of the performance and adaptability of both Spark and the implemented algorithms in a non-uniform environment.

6.1 Description of Datasets

We describe the generation process of the temporal graph datasets. As previously mentioned, there are no available temporal graph datasets with time intervals that also contain ground truth temporal communities. This is also true for synthetic generators.

Real Datasets

Four real-world datasets [39] were utilized to evaluate the effectiveness of distributed community detection algorithms. Due to the absence of temporal information, all edges in the datasets were assigned the same time interval. This means that the query time interval spans all edges. As a result, this set of experiments involves the whole graph, meaning that the results of our algorithms do not differ when compared to a CD algorithm on the initial static graph. In this manner, we are able to test the effectiveness of our algorithms regarding the quality of the community partition since we know the ground truth communities of the static graph. Additionally, we look at the efficiency of our methods related to a worst-case scenario where the whole graph must go through the whole pipeline of our methods, and the filtering is useless.

- **com-Amazon:** This dataset captures the Amazon product network, containing $334,863$ nodes and $925,872$ edges. Nodes represent products, and edges indicate co-purchase relationships, offering a scenario with a retail-based network structure.
- **com-DBLP:** The DBLP collaboration network dataset consists of $317,080$ nodes and $1,049,866$ edges. This graph represents a collaboration network where researchers are linked through co-authorship, making it ideal for testing triangle counting in academic and professional networks with a moderate community structure.
- **com-Youtube:** Representing the YouTube online social network; this dataset has $1,134,890$ nodes and $2,987,624$ edges. This dataset allows us to explore the algorithms' efficiency in sparse networks.
- **com-LiveJournal:** Representing the LiveJournal blogging and friendship network; this dataset contains $3,997,962$ nodes and $34,681,189$ edges. All nodes and edges belong to the largest connected component.

Orkut Synthetic Datasets

Our second set of experiments employs Orkut [39] extended with time intervals on edges based on dynamics that resemble its genesis, development, and decline. We first introduce the underlying network data and then detail our temporal modeling procedure. The Orkut dataset captures a snapshot from the early years of Orkut (around 2005), encompassing $3,072,441$ nodes and $117,185,083$ edges in its largest connected component. Ground-truth communities are derived from user-defined groups, with each connected component within a group treated as a distinct community. However, we cannot use it to argue about the community quality due to the temporal augmentation of the dataset. Building on this static structure, we partition the network's lifecycle into three phases and apply phase-specific rules for edge creation and dissolution, thereby transforming the snapshot into a (as much as possible) realistically evolving temporal graph. Regarding the timeline, we use discrete time that is based on weeks, spanning 11 years for a total of $11 \times 52 = 572$ time steps. The goal of this set of experiments is to explore how the query time interval impacts the efficiency of the methods.

Phase 1: Genesis and Viral Growth $(2004 - 2006)$ The initial phase covers Orkut's launch by Google and its explosive, invitation-driven growth, particularly in Brazil and India. The network was expanding rapidly as early adopters brought their social circles onto the platform. New friendships during this era were primarily driven by influence and existing social structures. We simulate this phase by considering only edge creation rules:

- *Preferential Attachment (60% probability):* New users are most likely to connect with existing, popular users (early adopters). This "rich-get-richer" mechanism accurately models the influence of initial hubs in a growing network.

- *Triadic Closure (*40% *probability):* The "friend-of-a-friend" effect is crucial. This is where a user befriends someone they have a mutual connection with, forming a triangular relationship. This mechanism is responsible for building dense local clusters.

Phase 2: The Community Era and Peak Maturity $(2007 - 2010)$ During this period, Orkut was the dominant social network in its key markets. The "Communities" feature was a central part of the user experience, driving interactions and new connections based on shared interests (hobbies, alumni groups, etc.). The drivers for new friendships shift from pure growth to deepening connections within established social spheres. In this case, we need both edge creation and dissolution rules:

- *Community-Driven Closure (*50% *probability):* This is the most important and Orkut-specific mechanism. It simulates users meeting and befriending each other through shared communities. Since the dataset doesn't explicitly define communities, we can approximate them. A "community" can be defined as the ego-network of a high-degree, high-clustering node. The rule is: (1) Select one of these "community hubs", (2) pick two random, unconnected nodes within that hub's neighborhood, and (3) form a friendship edge between them.

- *Triadic Closure (*30% *probability):* The standard "friend-of-a-friend" mechanism continues to operate for connections made outside of specific community contexts.

- *Preferential Attachment (*20% *probability):* The influence of global hubs diminishes as the network becomes more saturated and activity becomes more localized within communities.

During this mature phase, friendships also begin to dissolve. We can model their duration (lifespan) with a skewed distribution, as not all online friendships are equal, as follows:

- *Short-Lived Ties (60% of newly formed edges):* Lifespans are drawn from an exponential distribution (mean 1.5–2 years), modeling the transient nature of casual acquaintances.

- *Stable Ties (40% of newly formed edges):* Lifespans are sampled from a normal distribution with mean 4–5 years and low variance, reflecting durable, meaningful friendships.

Phase 3: The Great Migration and Decline $(2011 - 2014)$ This final phase captures Orkut's decline as its user base migrated to Facebook, which offered a more global reach and a different user experience. Engagement dropped, leading to the eventual shutdown of the service in 2014. The rate of new friendship formation slows to a trickle. The probabilities for the creation mechanisms remain, but the overall frequency of new edge creation is drastically reduced compared to the peak phase. The rate of dissolving edges accelerates significantly as users abandon the platform. Edges created during this phase are assigned considerably shorter lifespans compared to earlier phases. We explicitly model two categories of ties:

- *Short-Lived Ties (60% of edges):* Lifespans drawn from a normal distribution with a mean of 75 ticks (corresponding to roughly 6–9 months).

- *Stable Ties (40% of edges):* Lifespans drawn from a normal distribution with a mean of 240 ticks (approximately 2 years). This captures the increasing instability of connections as users began to leave the platform.

Synthetic Datasets

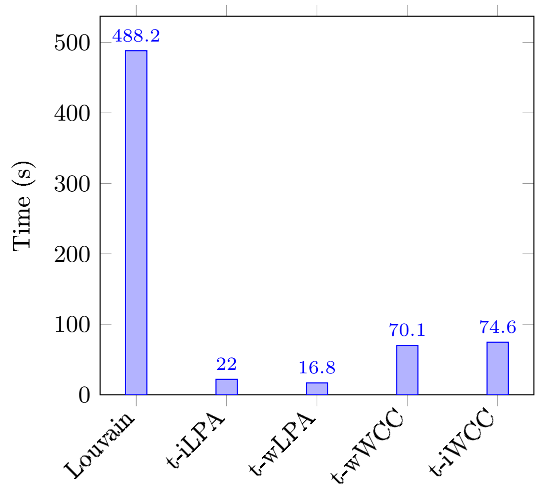
In our experiments, we employ synthetic datasets generated using RDyn [32], a framework designed to create dynamic networks that reflect key structural properties of real-world graphs. RDyn also provides time-evolving ground-truth communities with configurable quality, supporting events such as community merging and splitting. The generator relies on three main user-defined parameters: the total number of nodes, the number of iterations, and the average degree of nodes. Each iteration represents a set of edge actions, insertions, and deletions, where the number of actions may vary across iterations.

Each action in the RDyn-generated stream carries a timestamp indicating either an edge insertion or deletion. We record the insertion timestamp and the deletion timestamp for each edge; since edges are inserted and removed exactly once by the generator, these two values define each edge's active interval. As a result, we obtain edges alongside their corresponding active time intervals.

Table 1: Amazon Network

Method	NMI
Louvain	0.527
t-iLPA	0.685
t-wLPA	0.685
t-wWCC	0.697
t-iWCC	0.697

Figure 3: Execution Time on Amazon



6.2 Experimental Results

We first consider the experimental evaluation of the four real-world datasets. In each case, every edge was assigned the same fixed time interval, and we assessed the detected communities both in terms of quality – using the Normalized Mutual Information (NMI) metric against the ground truth – and computational efficiency, as measured by end-to-end runtime. We first report the results obtained on the Amazon network dataset.

Analysis of the Amazon Network. In Table 1 and Figure 3, we observe a clear trade-off between detection quality (NMI) and computational cost. More precisely, Louvain achieves the lowest alignment with ground truth (NMI = 0.527) and is the slowest method, requiring nearly 488s. Label-Propagation variants (t-iLPA and t-wLPA) both reach an NMI of 0.685, with t-wLPA converging faster (16.8s vs. 22s) due to the use of edge weights. WCC-based methods (t-wWCC and t-iWCC) attain the highest NMI (0.697); t-wWCC completes in 70.1s, while t-iWCC takes 74.6s. Overall, t-wWCC delivers the highest NMI at moderate cost (0.697 at 70.1s), t-wLPA offers the fastest runtime with a small drop in NMI (0.685 at 16.8s), and Louvain is dominated on both fronts.

Analysis of the DBLP Network. Table 2 and Figure 4 jointly present the noteworthy performance of all methods applied to the DBLP dataset. Louvain produces a modest NMI of 0.31 yet demands 545s, underscoring its substantial computation and relatively poor fit. Label-Propagation variants (t-iLPA and t-wLPA) both register an NMI of 0.65. By incorporating edge weights, t-wLPA slashes the execution time from 38s down to 28.2s. WCC-based methods (t-wWCC and t-iWCC) attain the highest NMI of 0.70. Among them, t-wWCC completes in 74.5s, while t-iWCC requires 79.7s. In brief, t-wWCC delivers the strongest alignment in a moderate timespan, W-LPA achieves the quickest turnaround with only a minor dip in NMI, and Louvain is surpassed on both fronts under our fixed-interval setup.

Analysis of the YouTube Network. Table 3 together with Figure 5 highlights the way various algorithms trade off between detection accuracy and computational efficiency. Louvain records the lowest NMI (0.473) and is the slowest method, taking over 1139s, reflecting its heavy modularity optimization on this large graph. Label-Propagation methods (t-iLPA and t-wLPA) achieve an NMI of 0.76. Moreover, t-wLPA cuts runtime to 120.7s from t-iLPA's 142.3s. WCC-based approaches, (t-wWCC and t-iWCC) reach the highest NMI of 0.86. Between them, t-wWCC completes in 256.8s, while t-iWCC requires 278.3s. t-wWCC attains the strongest concordance in approximately 257s, t-wLPA offers the quickest turnaround at 120.7s with only a modest NMI reduction, and Louvain trails on both fronts under the fixed-interval framework.

Table 2: DBLP Network

Method	NMI
Louvain	0.31
t-iLPA	0.65
t-wLPA	0.65
t-wWCC	0.7
t-iWCC	0.7

Figure 4: Execution Time on DBLP

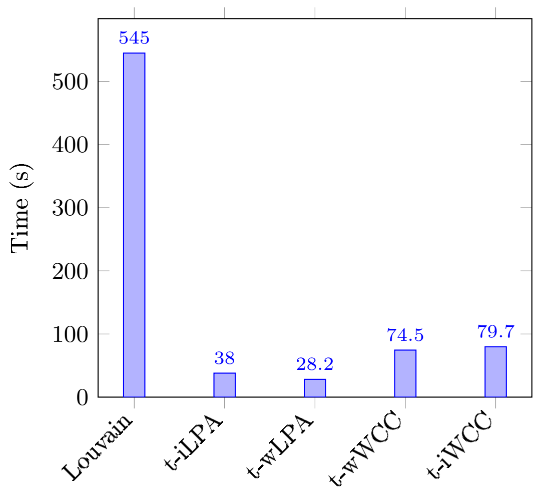


Table 3: YouTube Network

Method	NMI
Louvain	0.473
t-iLPA	0.76
t-wLPA	0.76
t-wWCC	0.86
t-iWCC	0.86

Figure 5: Execution Time on YouTube

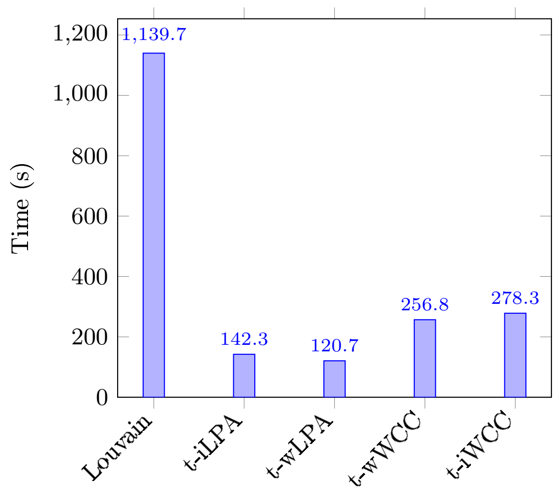
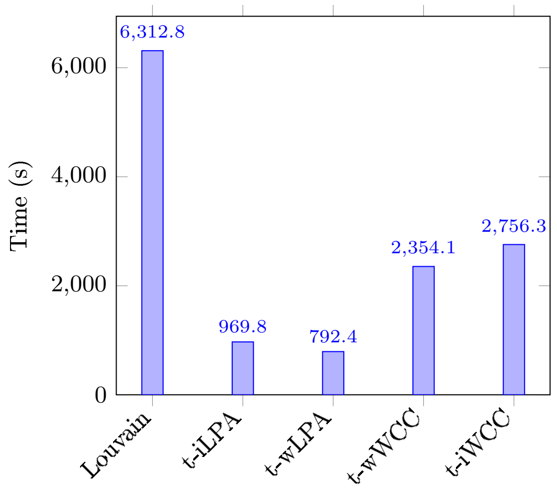


Table 4: Live Journal Network

Method	NMI
Louvain	0.338
t-iLPA	0.653
t-wLPA	0.653
t-wWCC	0.716
t-iWCC	0.716

Figure 6: Execution Time on Live Journal



Analysis of the Live Journal Network. Table 4 and Figure 6 collectively illustrate the relationship between the accuracy of community detection and the time required for computation. Louvain yields the lowest NMI (0.338) and incurs the greatest cost in time (6312.8s), highlighting its extensive computation with limited correspondence to the ground truth. Label-Propagation approaches (t-iLPA and t-wLPA) both reach an NMI of 0.653. In addition, t-wLPA trims the runtime from 969.8s down to 792.4s. WCC-based techniques (t-wWCC and t-iWCC) achieve the top NMI of 0.716. t-wWCC completes in 2354.1s, compared to 2756.3s for t-iWCC. t-wWCC thus secures the strongest match to the ground truth in a moderate timeframe, t-wLPA offers the quickest completion with only a slight drop in NMI, and Louvain falls short on both measures under the fixed-interval configuration.

Discussion on the Results of the Real-World Datasets. In this series of experiments, the full network was utilized because all edges shared identical time intervals, removing the need for temporal filtering. The outcomes clearly confirm the anticipated trend: Label Propagation Algorithm (LPA) methods outperform others in terms of execution speed, which aligns with their inherent propagation-based mechanism that facilitates rapid convergence. However, while LPA offers a time-efficient solution, it often does so at the expense of result quality.

Conversely, WCC approaches, though comparatively slower, demonstrate superior performance in terms of community detection accuracy. This trade-off between speed and accuracy becomes particularly evident in the YouTube dataset, where WCC-based algorithms consistently surpass the other three methods in clustering effectiveness. These findings highlight the classic tension in network analysis between computational efficiency and detection precision, emphasizing that method selection should be guided by the specific demands of the application – whether that be speed or quality.

Experiments on the Orkut Synthetic Network

Before presenting the results, it is important to first provide a brief overview of the implementation. What follows is a description of its main components.

1. *Timeline and Graph Loading:*The notion of discrete time in this implementation is based on weeks, resulting in a total of $11 \times 52 = 572$ time steps. The full Orkut edge list is loaded into a graph structure.

2. *Model Node Births:* To ensure a realistic simulation, nodes are not assumed to exist from the initial time step $t = 0$. We create a birth schedule for nodes that follows an S-curve: slow initial growth, followed by exponential growth in Phase 1, and leveling off in Phase 2. Each node's birth time marks the moment it becomes eligible to form connections. While this information could also define a node's valid time interval, this detail is not required in our setting.

3. *Iterate and Activate Edges:* Loop through the number of predefined time steps. In each time step, a certain number of edges is activated from the static dataset, based on the edge creation rules for the current phase. This does not involve creating new edges; rather, it determines when the pre-existing edges from the dataset become active.

4. *Assign Lifespans - Ties:* Upon activation of an edge at time t, the lifespan function determines its active duration, producing the interval $[t, t + lifespan]$. The lifespan parameter is drawn according to the short-lived and stable tie distributions specified in Phases 2 and 3.

By applying the lifecycle framework to the static Orkut graph, we generate a semi-realistic temporal social network. We make two different experiments on the Orkut dataset based on the query time interval as shown in Table 5.

Table 5: Time-augmented Orkut experiment setups reporting the counts of active nodes and edges for each query time interval.

Dataset	Nodes	Edges	Query Interval
Orkut-Synthetic 1	$2,457,994$	$21,066,416$	$[250 - 300]$
Orkut-Synthetic 2	$2,815,802$	$47,584,407$	$[250 - 400]$

Runtime Evaluation Figures 7 and 8 compare processing times for two query intervals. In the narrower interval [250,300], both t-iWCC (586.6s) and t-iLPA (591.2s) finish fastest, with t-wLPA close behind at 656.5s. t-wWCC requires 1533.1s, and Louvain takes 4862.3s. For the extended span [250,400], t-iLPA leads at 1150.1s, followed by t-wLPA at 1443.9s. The incremental WCC variant (t-iWCC) completes in 4490s and the weighted WCC (t-wWCC) in 5913.7s, while Louvain fails to produce any measurable result within the given timeframe, as it encounters an Out Of Memory (OOM) error during execution. Overall, t-iLPA consistently yields the quickest runtimes, WCC methods grow in cost as the interval widens, and Louvain becomes computationally prohibitive on larger interval.

This experiment clearly highlights the time efficiency of the methods based on interval intersection. In particular, the t-iWCC approach, which searches for triangles by checking the intersection of edge time intervals, benefits significantly from edge filtering. Many triangles are eliminated – either because some of the respective edges do not exist in the query time interval or their time intervals do not overlap. As a result, fewer triangles are formed, and during the iterative process where triangle recalculations are required within each community, substantial time is saved. The t-iLPA method exhibits a similar advantage. The number of pairwise interval intersections drops sharply after filtering, which directly contributes to reduced computational overhead. Overall, both methods demonstrate strong time efficiency due to a lower number of temporal relationships that need to be evaluated.

On the other hand, after temporal filtering, both methods based on the observation ratio assign a weight score to each edge and must process all filtered edges regardless of whether their weight is low or high. Although all methods apply graph filtering, t-iWCC achieves faster performance because it forms fewer triangles. This reduces the need for repeated triangle recalculations within each community during the iterative process, allowing it to converge significantly faster than t-wWCC.

Experiments on Synthetic Datasets

Table 6 presents a summary of the two synthetic datasets along with their characteristics. We performed extensive experiments using 15 different query time intervals on these synthetic datasets, and the average results from these experiments are reported below. The ground truth communities were derived at the conclusion of each complete iteration. Specifically, each iteration involved the synthetic dataset undergoing a full series of temporal changes, including edge insertions and deletions. The ground truth communities were generated only at the final temporal change of the current iteration. Consequently, we evaluated the entire iteration—including all temporal changes—by comparing it to the ground truth provided by the RDyn generator once the iteration was completed. Therefore, the query time interval was selected to span from the start timestamp to the end timestamp of the current iteration, or to be close to these timestamps. We assume that the community

Figure 7: Execution Time on Orkut for the query time interval [250, 300]

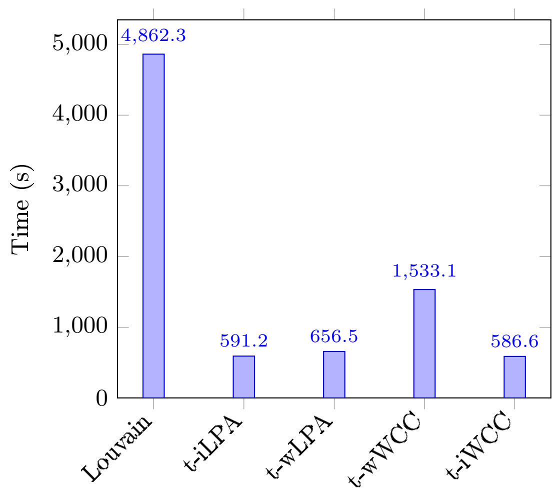
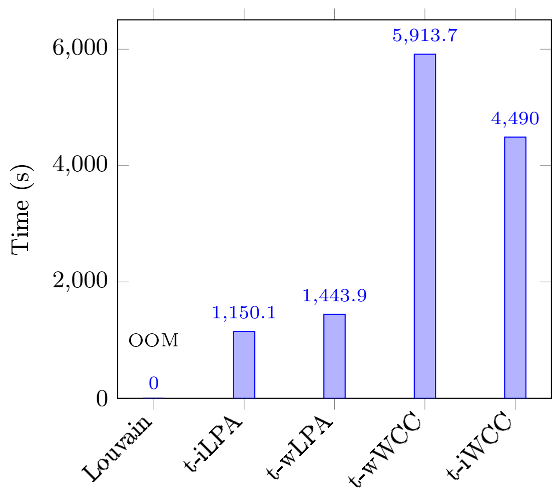


Figure 8: Execution Time on Orkut for the query time interval [250, 400]



structure remains relatively stable near the start and end timestamps, and that the edges that are not valid within this query time interval do not significantly affect the ground truth communities.

Table 6: Two synthetic datasets (SD1 and SD2) with the number of nodes, iterations, actions, and edges.

Dataset	Nodes	Iterations	Actions	Edges
$SD1$	3000	500	15686	13404
$SD2$	5000	1000	251108	138349

Analysis of the SD1 Synthetic Network. Table 7 and Figure 9 illustrate how each algorithm balances NMI performance against running time. Louvain yields a modest NMI of 0.323 in 66.3s, reflecting its lower agreement and moderate computation. Label-Propagation variants (t-iLPA and t-wLPA) achieve NMIs of 0.959 and 0.950, respectively, completing in just 3.4s and 3.5s—demonstrating very fast execution with high accuracy. WCC-based methods (t-wWCC and t-iWCC) secure the top NMIs (0.964 and 0.963) in around 20s (20.4s and 20.6s), trading off a small increase in runtime for peak alignment. In summary, t-wWCC and t-iWCC deliver the best NMI in a moderate timeframe, t-iLPA completes almost instantly with only a minor drop in NMI, and Louvain ranks lowest on both measures in this synthetic setting.

Analysis of the SD2 Synthetic Network. Table 8 and Figure 10 reveal distinct patterns in accuracy versus execution time. Louvain attains an NMI of 0.760 in 167.7s, indicating moderate agreement with substantial processing. Label-Propagation approaches (t-iLPA and t-wLPA) reach NMIs of 0.898 and 0.890, respectively, in under 4s (3.9s and 3.7s), highlighting rapid convergence with high fidelity. WCC-based techniques, (t-wWCC and t-iWCC) achieve the top alignment (0.910 and 0.920 NMI) in roughly 20s (20.4s and 18.6s), trading slightly longer runtimes for the best match. Overall, t-wWCC and t-iWCC offer peak correspondence in a moderate timeframe, label-propagation methods deliver near-instant results with strong accuracy, and Louvain lags behind on both fronts.

Discussion of the Results of the Synthetic Datasets The results from both synthetic datasets highlight clear trends in the trade-off between accuracy and execution time across different algorithms. Louvain consistently performs the weakest, showing lower accuracy and requiring the most processing time. Label Propagation variants (t-iLPA and t-wLPA) stand out for their exceptionally fast execution while still maintaining high accuracy. WCC-based methods (t-wWCC and t-iWCC) achieve the highest accuracy overall, with moderately longer runtimes, striking a strong balance

Table 7: SD1 Network

Method	NMI
Louvain	0.323
t-iLPA	0.959
t-wLPA	0.95
t-wWCC	0.964
t-iWCC	0.963

Figure 9: Execution Time on SD1

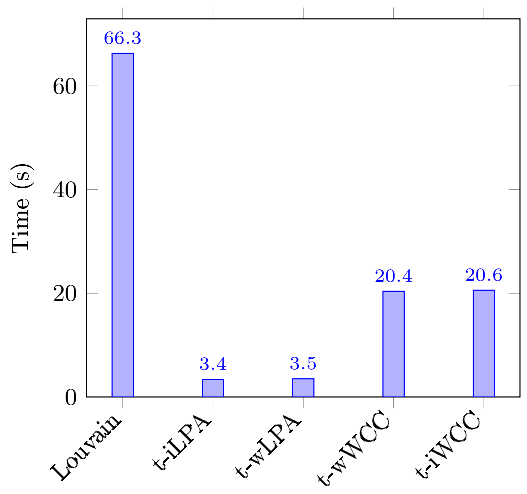
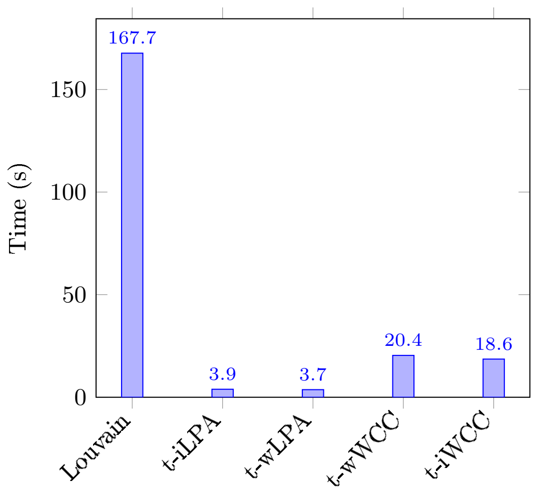


Table 8: SD2 Network

Method	NMI
Louvain	0.76
t-iLPA	0.898
t-wLPA	0.89
t-wWCC	0.91
t-iWCC	0.92

Figure 10: Execution Time on SD2



between precision and efficiency. Overall, WCC methods lead in performance, LPA methods are the fastest, and Louvain trails in both speed and alignment.

References

[1] Tariq Abughofa, Ahmed A. Harby, Haruna Isah, and Farhana Zulkernine. Incremental community detection in distributed dynamic graph. In *2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 50–59, 2021.

[2] HL Advaith, S Adarsh, G Akshay, P Sreeram, and GP Sajeev. A proximity based community detection in temporal graphs. In *2020 IEEE international conference on electronics, computing and communication technologies (CONECCT)*, pages 1–6. IEEE, 2020.

[3] Apache Spark Contributors. Label propagation algorithm in graphx. `https://github.com/apache/spark/blob/master/graphx/src/main/scala/org/apache/spark/graphx/lib/LabelPropagation.scala`, 2025. Accessed: 2025-07-15.

[4] Mehdi Azaouzi, Delel Rhouma, and Lotfi Ben Romdhane. Community detection in large-scale social networks: state-of-the-art and future directions. *Social Network Analysis and Mining*, 9(23), 2019.

[5] Luca Becchetti, Andrea E. Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Find your place: Simple distributed algorithms for community detection. *SIAM Journal on Computing*, 49(4):821–864, 2020.

[6] Agorakis Bompotas, Nikitas-Rigas Kalogeropoulos, and Christos Makris. Commc: A multi-purpose commodity hardware cluster. *Future Internet*, 17(3), 2025.

[7] Konstantinos Christopoulos, Evaggelos Daskalakis, Agorakis Bompotas, and Konstantinos Tsichlas. Triangle counting in large historical graphs. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, pages 432–439, 2025.

[8] Konstantinos Christopoulos and Konstantinos Tsichlas. Distributed community detection in temporal graphs. In *Proceedings of the 19th International Symposium on Spatial and Temporal Data (SSTD 2025)*, Osaka, Japan, August 2025. Accepted.

[9] Ben Collingsworth and Ronaldo Menezes. A self-organized approach for detecting communities in networks. volume 4, 09 2012.

[10] Jing Cui, Yi-Qing Zhang, and Xiang Li. On the clustering coefficients of temporal networks and epidemic dynamics. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2299–2302, 2013.

[11] R. Fathi, A. Rahaman Molla, and G. Pandurangan. Efficient distributed community detection in the stochastic block model. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 409–419, Los Alamitos, CA, USA, jul 2019. IEEE Computer Society.

[12] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[13] Tim Garrels, Athar Khodabakhsh, Bernhard Y Renard, and Katharina Baum. Lazyfox: fast and parallelized overlapping community detection in large graphs. *PeerJ Computer Science*, 9:e1291, 2023.

[14] Mohsen Ghaffari. *An Improved Distributed Algorithm for Maximal Independent Set*, pages 270–277. 2016.

[15] Sayan Ghosh, Mahantesh Halappanavar, Antonino Tumeo, Ananth Kalyanaraman, Hao Lu, Daniel Chavarrià-Miranda, Arif Khan, and Assefaw Gebremedhin. Distributed louvain algorithm for graph community detection. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 885–895, 2018.

[16] Iman Hashemi. Community detection in historical data using knowledge graphs. Master's thesis, 2022.

[17] Ruilin Hu, Chao Song, Jie Wu, and Li Lu. A cloud-edge collaborative framework for distributed triangle counting on graph stream.

[18] Rawan Jalabneh, Haniya Zehra Syed, Sunitha Pillai, Ehsanul Hoque Apu, Molla Rashied Hussein, Russell Kabir, SM Yasir Arafat, Md Anwarul Azim Majumder, and Shailendra K Saxena. Use of mobile phone apps for contact tracing to control the covid-19 pandemic: A literature review. *Applications of artificial intelligence in COVID-19*, pages 389–404, 2021.

[19] Andreas Kosmatopoulos, Kostas Tsichlas, Anastasios Gounaris, Spyros Sioutas, and Evaggelia Pitoura. Hinode: an asymptotically space-optimal storage model for historical queries on graphs. *Distributed Parallel Databases*, 35(3-4):249–285, 2017.

[20] Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New journal of physics*, 11(3):033015, 2009.

[21] Shijie LI, Yang LIU, Jintao TANG, and Hang QIE. An isolated sets based parallel louvain algorithm for community detection. *Computer Engineering & Science*, 47(04):621, 2025.

[22] Xin Liu, Hui-Min Cheng, and Zhong-Yuan Zhang. Evaluation of community detection methods. *IEEE Transactions on Knowledge and Data Engineering*, 32(9):1736–1746, 2019.

[23] Zahra Masdarolomoor, Sadegh Aliakbary, Reza Azmi, and Noshin Riahi. Distributed community detection in complex networks. In *2011 Third International Conference on Computational Intelligence, Communication Systems and Networks*, pages 281–286, 2011.

[24] Peter J Mucha, Thomas Richardson, Kevin Macon, Mason A Porter, and Jukka-Pekka Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *science*, 328(5980):876–878, 2010.

[25] Apostolos N. Papadopoulos and Georgios Tzortzidis. Distributed time-based local community detection. In *Proceedings of the 24th Pan-Hellenic Conference on Informatics*, PCI '20, page 390–393, New York, NY, USA, 2021. Association for Computing Machinery.

[26] Nebula Graph Contributors. Nebula graph java client (v1.0). `https://github.com/vesoft-inc/nebula-java/blob/v1.0/README.md`, 2025. Accessed: 2025-07-15.

[27] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. Big data technologies: A survey. *Journal of King Saud University-Computer and Information Sciences*, 30(4):431–448, 2018.

[28] Michael Ovelgönne. Distributed community detection in web-scale networks. In *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*, pages 66–73. IEEE, 2013.

[29] Arnau Prat-Pérez, David Dominguez-Sal, Josep M. Brunat, and Josep-Lluis Larriba-Pey. Shaping communities out of triangles. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, page 1677–1681, New York, NY, USA, 2012. Association for Computing Machinery.

[30] Arnau Prat-Pérez, David Dominguez-Sal, and Josep-Lluis Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *Proceedings of the 23rd international conference on World wide web*, pages 225–236, 2014.

[31] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106, Sep 2007.

[32] Giulio Rossetti. Rdyn: graph benchmark handling community dynamics. *Journal of Complex Networks*, 5(6):893–912, 2017.

[33] Giulio Rossetti and Rémy Cazabet. Community discovery in dynamic networks: a survey. *ACM Computing Surveys (CSUR)*, 51(2):1–37, 2018.

[34] Matthew Saltz, Arnau Prat-Pérez, and David Dominguez-Sal. Distributed community detection with the wcc metric. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, page 1095–1100, New York, NY, USA, 2015. Association for Computing Machinery.

[35] Naw Safrin Sattar, Khaled Z Ibrahim, Aydin Buluc, and Shaikh Arifuzzaman. Dyg-dpcd: A distributed parallel community detection algorithm for large-scale dynamic graphs. *International Journal of Parallel Programming*, 53(1):4, 2025.

[36] Xing Su, Shan Xue, Fanzhen Liu, Jia Wu, Jian Yang, Chuan Zhou, Wenbin Hu, Cecile Paris, Surya Nepal, Di Jin, et al. A comprehensive survey on community detection with deep learning. *IEEE transactions on neural networks and learning systems*, 35(4):4682–4702, 2022.

[37] Giorgio Venturin, Ilie Sarpe, and Fabio Vandin. Efficient approximate temporal triangle counting in streaming with predictions. *arXiv preprint arXiv:2506.13173*, 2025.

[38] Shihao Wu, Zhe Li, and Xuening Zhu. A distributed community detection algorithm for large scale networks under stochastic block models. *Computational Statistics & Data Analysis*, 187:107794, 2023.

[39] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, pages 1–8. ACM, 2012.

[40] Li Yang and Abdallah Shami. Iot data analytics in dynamic environments: From an automated machine learning perspective. *Engineering Applications of Artificial Intelligence*, 116:105366, 2022.

[41] Ping Zhao, Biyou Wang, and Rong Ye. Locally differentially private k-triangle counting in real-time social graph. *IEEE Transactions on Computational Social Systems*, 2025.