# Distributed Community Detection in Temporal Graphs

Konstantinos Christopoulos*
kchristopou@ac.upatras.gr
University of Patras
Patra, Greece

Konstantinos Tsichlas*
ktsichlas@ceid.upatras.gr
University of Patras
Patra, Greece

## Abstract

Community detection stands as a pivotal process in network analysis, having undergone extensive examination over the past 25 years within static network contexts. This task involves partitioning networks based on structural characteristics, specifically into classes of nodes exhibiting denser connections than the overall network. Moreover, in recent years, an additional layer of complexity has arisen, particularly in networks characterized by a static nature where each node and edge is assigned a valid time interval. Such historical graphs, unlike traditional graphs, incorporate a temporal dimension, allowing for the analysis of how connections between entities evolve and change over different time intervals. In this study, we present a distributed algorithm for static community detection within a query time interval in historical graphs. Specifically, when provided with a designated query time interval, our proposed method identifies communities by assessing the individual contributions of each edge and node within the graph during that specified time interval. To the best of our knowledge, this setting has not been considered before in the literature.

## CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

**ACM Reference Format:**
Konstantinos Christopoulos and Konstantinos Tsichlas. 2025. Distributed Community Detection in Temporal Graphs. In *19th International Symposium on Spatial and Temporal Data (SSTD '25), August 25–27, 2025, Osaka, Japan.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3748777.3748793

## 1 Introduction

Networks are widely used for data analysis in domains such as social sciences, transportation, and biology. A static network is typically represented as $G = (V, E)$, where $V$ denotes entities (nodes) and $E$ denotes interactions (edges), which may be directed or undirected. Extending to temporal settings, historical graphs assign valid time intervals to each node and edge. These graphs are static in structure but dynamic in behavior due to time-based activation of nodes and edges.

Community Detection (CD) aims to identify densely connected node groups[8, 16] (communities) and has applications ranging from social networks to biological systems. Most existing CD algorithms target networks with millions of nodes and edges [19]. However, the rise of massive datasets from sources like social media, IoT, healthcare, and retail [18] has led to networks with billions of entities—e.g., Facebook had 3.51 billion active users as of June 2021[1]—demanding greater computational resources. However, most global CD algorithms run on a single machine, limiting their scalability. This problem can be tackled by moving to a distributed setting for the CD problem.

This paper proposes a distributed algorithm for detecting communities in historical graphs within a given query time interval. Building on the Weighted Clustering Coefficient (WCC) [20], we extend it to support temporal graphs by introducing the local temporal Clustering Coefficient (ltCC), along with other adapted metrics.

The rest of the paper is organized as follows: Section 2 reviews related work, Section 3 defines key concepts, Section 4 presents our algorithm, and Section 5 concludes the paper

## 2 Related Work

The literature on distributed CD, especially in dynamic or historical graphs, remains limited. In historical graphs, community detection typically involves identifying communities in snapshots taken at specific time points, following the instant-optimal paradigm [23]. A foundational distributed CD method based on the WCC metric is presented in [24]. The algorithm runs in three phases: (1) Preprocessing that filters edges not participating in triangles, (2) Initialization that builds communities using clustering coefficients, and (3) WCC Iteration that refines memberships until convergence. An incremental version of this approach using Apache Spark/GraphX is proposed in [1].

LazyFox [9] builds on WCC by optimizing triangle counting. Similarly, [17] adapts PHASR for distributed environments using Spark, focusing on temporal conductivity and refinement via Personalized PageRank. In [12], CD is performed on heterogeneous temporal networks by converting historical data into snapshots, using algorithms tailored for both static and dynamic scenarios. A related preprocessing method is discussed in [2].

A local modularity-based distributed approach is presented in [15], where communities grow from individual nodes through local iterative expansion. Among TLAV (Think Like a Vertex) models, [7] proposes a lightweight random-walk-based algorithm for local CD, while a spectral method with similar objectives is found in [25]. A message-passing implementation appears in [11], and label propagation is employed in both classical [22] and dynamic [4] distributed settings. A comprehensive survey of distributed CD algorithms is provided in [3], categorizing them into self-aggregation

*Both authors contributed equally to this research.


*SSTD '25, Osaka, Japan*

ACM ISBN 979-8-4007-2094-9/25/08
https://doi.org/10.1145/3748777.3748793

[1] https://www.zephoria.com

and self-organization methods. Finally, more related waork we can find in [].

In our work, we address triangle counting in historical graphs which is a topic that has only recently begun to receive attention, as evidenced by two recent studies [5, 26]. Among these, the study in [26] considers a simplified scenario where edge histories are defined by discrete timestamps. In contrast, our focus is on a more general and challenging setting where edge histories are represented as time intervals (i.e., valid intervals). To this end, we build on the approach introduced in [5], which leverages temporal interval intersections to identify triangles—specifically, those where all three constituent edges are simultaneously active during at least one common time instance. We extend this approach by assigning a temporal coherence score to each triangle, aligning closely with the objectives and methodology of our proposed framework.

## 3 Definitions

Let $G = (V_T, E_T)$ be a static historical network. The set of historical nodes $V_T$ consists of a set of nodes along with their time intervals[2], that is $V_T \subset V \times \mathbb{N}^2$. The set of historical edges $E_T$ is a set of edges along with their time intervals, that is $E_T \subset E \times \mathbb{N}^2$, where $E$ contains all possible $\binom{|V|}{2}$ undirected edges. Note that we consider nodes and edges that have a single valid time interval, but it is easy to generalize to a set of valid time intervals. The preceding definitions mean that each node $v \in V$ (and edge $e \in E$) has a time interval attached $[t_v^{(s)}, t_v^{(f)}]$ (similarly $[t_e^{(s)}, t_e^{(f)}]$) (where $(s)$ and $(f)$ stand for start and finish respectively) that dictates the time instances where node $v$ (edge $e$) is existent. Thus, if $t \notin [t_v^{(s)}, t_v^{(f)}]$, then $v$ is not existent at time $t$. $V_{ij} \subseteq V$ contains all nodes that have a time interval that spans the query interval $[t_i, t_j]$. The time interval of each edge is, by definition, a subset of the time interval of the respective vertices. In case of multiple time intervals, we have to define the borders of each interval accordingly, to avoid overlaps. For example, each interval should be open at the left and closed at the right. The convention we make is that a time point $t$ is represented by $(t, t]$.

Assume that by $\mathcal{N}_{ij}(v)$ we represent the neighborhood of node $v$ in the query time interval $[t_i, t_j]$. Note that $\mathcal{N}_{ij}(v)$ may even be the empty set for specific query time intervals. The routing table $r(v)$ of node $v$, contains all historical edges to other nodes in $G$. This means that $r(v)$ contains all edges and nodes along with their time interval. By $r_{ij}(v)$, we represent the part of this table that contains historical edges with time intervals that intersect the query time interval $[t_i, t_j]$, that is all edges that point to nodes that belong to $\mathcal{N}_{ij}(v)$.

Our algorithm is designed having in mind the Local distributed model [14]. In a nutshell, the nodes of the graph have distinct IDs and communicate with the nodes in their neighborhood with messages (of unbounded size). Communication and computation are synchronous, the distributed system is fault-free while the nodes are considered to be infinitely powerful.

[2]The provided definitions allow both for continuous and discrete time. One could just as well assume real numbers to represent time. For simplicity, we assume natural numbers.

### 3.1 Problem Formulation

We assume that the historical graph is stored in a vertex-centric system, [13], which is space efficient and demonstrates the potential for enhancing the efficiency and functionality of both update and query operations. This means that the whole history of a node along with its adjacent edges is stored in the node itself. We want to support the following query:

**CD(**$G, [t_i, t_j]$**):** Find the aggregate communities in graph $G$ in the time interval $[t_i, t_j]$. If $t_i = t_j$, find 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 [24] 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.

However, when the query contains a time interval $[t_i, t_j]$, an approach would be to identify aggregated communities during this interval. The other option is to identify the evolution of all (or some) of these communities during this time interval. In this case, apart from identifying these communities and their evolution, new problems must be handled related to reporting all these communities and their evolution efficiently. The aggregation in this case takes into account the contribution of each node/edge related to its overlap with the query time interval. This means that we wish to take into account that some nodes/edges are not valid in all time instances of the query time interval $[t_i, t_j]$. For example, assume discrete time and assume that the user requests the communities of the graph in the time interval $T = [2, 6]$. The aforementioned time interval consists of 5 different time instances $\{2, 3, 4, 5, 6\}$. Potentially, there are nodes/edges that are valid over a span of 1, 2, 3 or 4 time instances in this time interval instead of spanning it entirely. To be more precise, assume an edge $e$ that is valid in the time interval $[1, 5]$. Since $e$ is valid for 4 time instances out of 5, we will assume that the edge will have a weight equal to $\frac{4}{5}$. Informally, this weight definition will be used to partition the graph into communities for the query time interval (it is straightforward to generalize it to continuous time). The preceding discussion is captured in the following definition.

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.*

When a node/edge is not valid at any time instance of the query time interval, then it is nonexistent during this time interval. Consequently, the unweighted historical graph is transformed into a weighted static graph for a particular time interval, where the weight of each node/edge represents the ratio of the size of its observed interval to the size of the total query interval.

Definition 2. *The **observation ratio** of an object (node, edge or 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 maximum value of the observation ratio is 1 when the time interval of the node/edge spans the query interval entirely. The minimum value of the observation ratio is zero when the time interval of the object and the query time interval are disjoint.

### 3.2 The *WCC* and *tlCC* Metric for Temporal Graphs

The basic idea behind the Weighted Clustering Coefficient (henceforth *WCC*) metric [24], is that within a community, vertices should have a higher concentration of triangles among themselves than with other nodes outside the community. In this work, we extend and adapt this metric to manage historical graphs instead of unweighted static graphs. Given a query time interval, the proposed variant of *WCC* should handle each edge and node of the historical graph in a way that reflects their contribution in that time interval. In other words, we estimate the contribution of each edge and vertex of the graph to *WCC* computation in a given time interval. Based on this idea, given a historical graph $G(V_T, E_T)$, we define the cohesion of a vertex $x$ to a set of vertices $S$ for the query time interval $[t_i, t_j]$ as follows:

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

The function $t_{ij}(x,S)$ denotes the sum of the contributions of the edges to triangles closed by $x$, with vertices in $S$ during the time interval $[t_i, t_j]$. More precisely, given a triangle with vertices $u, v, x$, and their edges $e_1 = (x, u, t_{e_1}^{(s)}, t_{e_1}^{(f)})$, $e_2 = (x, v, t_{e_2}^{(s)}, t_{e_2}^{(f)})$ and $e_3 = (u, v, t_{e_3}^{(s)}, t_{e_3}^{(f)})$, the observation ratio of the triangle closed by $x$ is defined in Equation (2) as follows:

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

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

This is the ratio of the size (number of time instances) contained in the intersection of the edges's time intervals with the query time interval to the size of the query time interval. $t_{ij}(x,V)$ is defined accordingly. The function $vt_{ij}(x,S)$ estimates the sum of the contributions of the vertices contained in all such triangles in $S$. The contribution of a vertex $u$ to triangles closed by $x$ in $S$, is defined as follows:

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

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

This is the ratio of the size of the union of the time intervals of all triangles (the intersection of the time intervals of the three respective edges) of $x$ and $u$ with nodes $v \in S$ and the query time interval, with the size of the query time interval. Then, in Equation (5) the total contribution $vt_{ij}(x,S)$ of each vertex in triangles closed by $x$ is defined. The function $vt_{ij}(x, V \setminus S)$ is defined similarly.

Finally, $|S \setminus \{x\}|_{ij}$ is estimated based on the observation ratio of each vertex excluded $x$, in $S$. Thus, given the vertex $w \in S$ and its time interval $[t_w^{(s)}, t_w^{(f)}]$, the contribution of node $w$ is defined as follows:

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

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

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

Practically, the proposed *WCC* cohesion metric rewards a high ratio of intra-community closed triangles to inter-community closed triangles (the left-hand side term) and awards a penalty to community vertices that do not close any triangle in the community (the right-hand side term).

Given a partitioning $P_{ij} = C_1, C_2, ..., C_n$ during the query time interval $[t_i, t_j]$ of $V$ in $G$, the total $WCC_{ij}$ score of this partitioning $P$ is defined as follows:

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

where $|V|_{ij}$ is defined similarly to Equation (7). Given $P$, the $WCC_{ij}(P)$ score is the weighted average of the $WCC_{ij}(C)$ scores of all the communities in the partition.

To initialize communities we make use of the temporal local Clustering Coefficient (tlCC). Local clustering coefficient has been defined for temporal graphs but in an ad-hoc manner based on its use (e.g., [6]). We define tlCC for node $u$ in the query time interval $[t_i, t_j]$, as the sum of the observation ratios of all triangles containing $u$ divided by the maximum observation ratios that one could get for $u$. This maximum is computed by keeping existing edges between the neighbors of $u$ with their current time intervals and adding all possible non-existent edges during the query time interval $[t_i, t_j]$. The time interval of these new edges is equal to the query time interval $[t_i, t_j]$; thus, the observation ratio of each new edge is 1. We define $t'_{ij}(x,V)$ similarly to $t_{ij}(x,V)$ but for the graph that also contains these new edges. Then, tlCC is defined as follows:

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

where $d_{ij}(u)$ denotes the aggregate degree of node $u$ within the time interval $[t_i, t_j]$.

## 4 The Distributed Algorithm for Community Detection

In what follows, we describe the three steps of the proposed distributed community detection algorithm for historical graphs: preprocessing, community initialization, and community refinement via the $WCC_{ij}$ iteration. This algorithm is based on the algorithms presented in [21, 24].

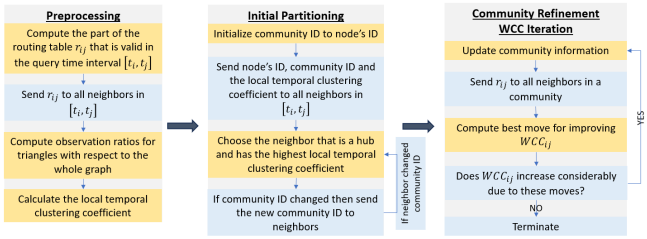


**Figure 1: A depiction of the method's three phases: yellow boxes show local computations, while blue boxes indicate computations requiring inter-node communication.**

*Preprocessing.* In the preprocessing phase, certain functions are computed to obtain an initial partition of our historical graph and facilitate the next phases. In this step, the functions $t_{ij}(x, V)$, $vt_{ij}(x, V)$ and $C_V(x)$ are calculated $\forall x \in V_{ij}$. $vt_{ij}(x, V)$ can be easily estimated based on $t_{ij}(x, V)$, since $t_{ij}(x, V)$ value supplement the $vt_{ij}(x, v)$ value. These values are immutable throughout the computation and therefore only need to be calculated once.

Via message passing, all vertices $x$ communicate their $r_{ij}(x)$ to their neighbors $\mathcal{N}_{ij}(x)$. Then, each node calculates the nodes with which it forms triangles by computing the intersection of their neighborhoods. Due to the fact that the time intervals are also communicated, we can use Equations (2) – (7) to compute the function $t_{ij}(x, V)$, $vt_{ij}(x, V)$ and $C_V(x)$ for all nodes $x \in V_{ij}$.

When dealing with huge graphs, the communication of $r_{ij}(x)$ of a node to all its neighbors in a single superstep, could result in prolonged communication time or, in a worse scenario, memory issues leading to worker failures. More precisely, when vertices with high degrees transmit their routing tables, it leads to notably greater communication costs in contrast to messages from vertices with low degrees. To circumvent this scenario and improve the communication cost, vertices exchange exclusively their routing tables with adjacent vertices that have higher degrees. In this way, we encourage only one-way communication, from low-degree vertices to high-degree vertices. Consequently, all the adjacent high-degree nodes of node $u$ estimate the sum of the contributions of the edges to triangles closed by $u$, and they respond by sending a message, to $u$ containing this information. Subsequently, all the values of $t_{ij}(x, V)$ and $vt_{ij}(x, V)$, $\forall x \in Vij$, are estimated.

*Community Initialization.* This phase involves the initialization of the communities based on the local temporal clustering coefficient $tlCC_{ij}$. This is based on the assumption that the higher the $tlCC_{ij}$ of a vertex, the more likely its neighbors are to belong to its community, considering that a high local temporal clustering coefficient shows how closely connected these vertices are.

To achieve this initial partitioning, we use a strategy that resembles that of a distributed algorithm for Maximal Independent Set (MIS) [10]. We impose the following rules for the initial partitioning: a) Each initial community is a star network, where it consists of a central node $v$ (hub) and its periphery, that is, a subset of $\mathcal{N}_{ij}(v)$, b) the hub is the node with the highest $tlCC$ in the community. and c) each node in the periphery is connected to the hub with the highest tlCC among all its neighbors. For initialization, one could remove rules (b) and (c) and simply apply a distributed algorithm for MIS. Although this is faster, it may degrade the quality of the initial partitioning.

*Partition Optimization via $WCC_{ij}$.* In this phase, we use simple rules to iteratively optimize the community partition by improving repeatedly the $WCC_{ij}(P)$ score for a partitioning $P$ of communities. A user-defined threshold $\theta$ controls when the iterations will terminate and provides a trade-off between quality of solution and speed. In each iteration, each node has three different choices (moves) as to what it will do concerning the current community partition. The move that is chosen by the algorithm is the one that improves $WCC_{ij}(P)$ the most. The allowed moves are the following:

(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 estimate the best move $\forall x \in V$, all computations are done in parallel.

Given the new partition after one iteration, we check if the termination condition is met. If the improvement in global quality exceeds a predefined threshold $\theta$, $\frac{WCC_{ij}(P') - WCC_{ij}(P)}{WCC_{ij}(P)} \geq \theta$, indicating a significant improvement in the community structure, the process continues with another iteration. Otherwise, if the improvement is less than $\theta$, the process terminates, and each node retains the community ID to which it belongs. This distributed termination approach ensures that the clustering algorithm converges efficiently while accommodating the one-way communication nature of the computation.

**Details of $WCC_{ij}(P)$ estimation:** To determine the actual global Weighted Clustering Coefficient ($WCC_{ij}(P)$), it is necessary to compute the values $t_{ij}(x, C)$ and $vt_{ij}(x, C)$ for each vertex $x$ and its community $C$. Note that for each vertex $w$, the $C_S(w)$ quantity is calculated during the preprocessing phase. Then, we use Equation (7) to calculate the quantity $|S \setminus \{x\}|_{ij}$, for vertices $x$ and their community $S = C$. This process is similar to the distributed approach used in the preprocessing, but with the distinction that messages are exclusively exchanged among vertices within the same community $C$. Consequently, this step is less computationally intensive than the global procedure. The resultant local $WCC_{ij}(C)$ values are then combined and averaged to derive the global $WCC_{ij}(P)$. In the event of achieving a new best $WCC_{ij}(P)$, vertices store their current communities. When the termination criterion is satisfied, vertices output their current community ID that contributed to the overall best $WCC_{ij}(P)$.

## 5 Conclusions

We address community detection in static networks with temporal semantics, where nodes and edges have valid time intervals. This is the first approach for Temporal Historical Graphs, introducing temporal node/edge contributions and adapting existing metrics. Future work includes analyzing time/message complexity and testing heuristics to improve performance or quality, plan to support nodes with multiple time intervals, like multi-interval edges, by treating each interval as a separate instance, and extensive experimentation.

## Acknowledgment

"This research was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "2nd Call for H.F.R.I. Research Projects to support Faculty Members & Researchers" (Project Number: 3480). "

## References

[1] Tariq Abughofa, Ahmed A. Harby, Haruna Isah, and Farhana Zulkernine. 2021. Incremental Community Detection in Distributed Dynamic Graph. In *2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService)*. 50–59. doi:10.1109/BigDataService52369.2021.00012

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

[3] Mehdi Azaouzi, Delel Rhouma, and Lotfi Ben Romdhane. 2019. Community detection in large-scale social networks: state-of-the-art and future directions. *Social Network Analysis and Mining* 9, 23 (2019). doi:10.1007/s13278-019-0566-x

[4] Luca Becchetti, Andrea E. Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. 2020. Find Your Place: Simple Distributed Algorithms for Community Detection. *SIAM J. Comput.* 49, 4 (2020), 821–864. doi:10.1137/19M1243026 arXiv:https://doi.org/10.1137/19M1243026

[5] Konstantinos Christopoulos, Evaggelos Daskalakis, Agorakis Bompotas, and Konstantinos Tsichlas. 2025. Triangle Counting in Large Historical Graphs. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing* (Catania International Airport, Catania, Italy) *(SAC '25)*. Association for Computing Machinery, New York, NY, USA, 432–439. doi:10.1145/3672608.3707982

[6] Jing Cui, Yi-Qing Zhang, and Xiang Li. 2013. On the clustering coefficients of temporal networks and epidemic dynamics. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2299–2302. doi:10.1109/ISCAS.2013.6572337

[7] R. Fathi, A. Rahaman Molla, and G. Pandurangan. 2019. Efficient Distributed Community Detection in the Stochastic Block Model. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, Los Alamitos, CA, USA, 409–419. doi:10.1109/ICDCS.2019.00048

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

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

[10] Mohsen Ghaffari. 2016. *An Improved Distributed Algorithm for Maximal Independent Set.* SIAM, 270–277. doi:10.1137/1.9781611974331.ch20 arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611974331.ch20

[11] Sayan Ghosh, Mahantesh Halappanavar, Antonino Tumeo, Ananth Kalyanaraman, Hao Lu, Daniel Chavarrià-Miranda, Arif Khan, and Assefaw Gebremedhin. 2018. Distributed Louvain Algorithm for Graph Community Detection. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 885–895. doi:10.1109/IPDPS.2018.00098

[12] Iman Hashemi. 2022. *Community Detection in Historical Data Using Knowledge Graphs.* Master's thesis. Utrecht University.

[13] Andreas Kosmatopoulos, Kostas Tsichlas, Anastasios Gounaris, Spyros Sioutas, and Evaggelia Pitoura. 2017. HiNode: an asymptotically space-optimal storage model for historical queries on graphs. *Distributed Parallel Databases* 35, 3-4 (2017), 249–285. doi:10.1007/S10619-017-7207-Z

[14] Nathan Linial. 1992. Locality in Distributed Graph Algorithms. *SIAM J. Comput.* 21, 1 (1992), 193–201. doi:10.1137/0221015 arXiv:https://doi.org/10.1137/0221015

[15] Zahra Masdarolomoor, Sadegh Aliakbary, Reza Azmi, and Noshin Riahi. 2011. Distributed Community Detection in Complex Networks. In *2011 Third International Conference on Computational Intelligence, Communication Systems and Networks*. 281–286. doi:10.1109/CICSyN.2011.66

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

[17] Apostolos N. Papadopoulos and Georgios Tzortzidis. 2021. Distributed Time-Based Local Community Detection. In *Proceedings of the 24th Pan-Hellenic Conference on Informatics* (Athens, Greece) *(PCI '20)*. Association for Computing Machinery, New York, NY, USA, 390–393. doi:10.1145/3437120.3437347

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

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

[20] Arnau Prat-Pérez, David Dominguez-Sal, Josep M. Brunat, and Josep-Lluis Larriba-Pey. 2012. Shaping Communities out of Triangles. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management* (Maui, Hawaii, USA) *(CIKM '12)*. Association for Computing Machinery, New York, NY, USA, 1677–1681. doi:10.1145/2396761.2398496

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

[22] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E* 76 (Sep 2007), 036106. Issue 3. doi:10.1103/PhysRevE.76.036106

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

[24] Matthew Saltz, Arnau Prat-Pérez, and David Dominguez-Sal. 2015. Distributed Community Detection with the WCC Metric. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW '15 Companion)*. Association for Computing Machinery, New York, NY, USA, 1095–1100. doi:10.1145/2740908.2744715

[25] Shihao Wu, Zhe Li, and Xuening Zhu. 2023. A distributed community detection algorithm for large scale networks under stochastic block models. *Computational Statistics & Data Analysis* 187 (2023), 107794. doi:10.1016/j.csda.2023.107794

[26] Yuyang Xia, Yixiang Fang, and Wensheng Luo. 2025. Efficiently Counting Triangles in Large Temporal Graphs. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–27.