Mosaic Benchmark Networks: Modular Link Streams for Testing Dynamic Community Detection Algorithms

Yasaman Asgari[1,3,4(✉)], Remy Cazabet[2], and Pierre Borgnat[1]

[1] Univ de Lyon, ENS de Lyon, CNRS, Laboratoire de Physique, 69342 Lyon, France
[2] Univ Lyon, UCBL, CNRS, INSA Lyon, LIRIS, UMR5205,
69622 Villeurbanne, France
[3] University of Zurich UZH Digital Society Initiative, Rämisttrasse 69,
8001 Zürich, Switzerland
yasaman.asgari@math.uzh.ch
[4] University of Zurich, Institut für Mathematik Winterthurerstrasse 190,
8057 Zürich, Switzerland

Abstract. Community structure is a critical feature of real networks, providing insights into nodes' internal organization. Nowadays, with the availability of highly detailed temporal networks such as link streams, studying community structures becomes more complex due to increased data precision and time sensitivity. Despite numerous algorithms developed in the past decade for dynamic community discovery, assessing their performance on link streams remains a challenge. Synthetic benchmark graphs are a well-accepted approach for evaluating static community detection algorithms. Additionally, there have been some proposals for slowly evolving communities in low-resolution temporal networks like snapshots. Nevertheless, this approach is not yet suitable for link streams. To bridge this gap, we introduce a novel framework that generates synthetic modular link streams with predefined communities. Subsequently, we evaluate established dynamic community detection methods to uncover limitations that may not be evident in snapshots with slowly evolving communities. While no method emerges as a clear winner, we observe notable differences among them.

Keywords: Temporal networks · Dynamic community detection · Network generator

1 Introduction

Community structure is a common feature in real networks. Essentially, a community represents a network pattern where nodes have strong connections within the community and weaker connections with nodes outside[1]. Network science initially emerged when real-world temporal network data, which captures the changing structure of networks over time, was scarce. Consequently, early research on community detection primarily focused on static networks [2]. With

the increasing availability of low-resolution temporal data, such as network snap-shots, attention naturally shifted to the dynamic community detection[3]. Nowa-days, we have access to highly detailed temporal networks like link streams, making the study of community structures more intricate due to the increased data granularity and time-sensitive nature.

One approach to quantitatively evaluate community detection algorithms is to employ synthetic graph generators to assess their performance and accuracy against a reference ground truth. Recently, several synthetic temporal network generators have been established to simulate snapshots with "slowly evolving" communities, i.e., having a meaningful community structure at each discrete time step. However, link streams fail to meet this criterion. Within a link stream, the frequency of interactions per node per unit of time is exceedingly low. Conse-quently, during a specific time instance, we can only discern a minimal number of connections, and these connections do not reveal clear community structures. Although we can employ varying or fixed window sizes to segment our net-work into multiple time slices to achieve low-resolution temporal networks like snapshots, a new line of research has emerged, aiming to develop algorithms tailored to the continuous-timed nature of instantaneous edges [4]. Motivated by this challenge, we intend to address the issue of the absence of a benchmark net-work for simulating modular link streams. To address this need, we introduce the "Mosaic benchmark network." Mosaic is designed to provide a reliable framework for evaluating and benchmarking dynamic community detection algorithms. It generates modular temporal networks with continuous-time edges built upon randomly planted partition networks, creating adjustable ground truths.

The paper's organization is outlined as follows: It begins by briefly reviewing the current literature concerning benchmarks used for creating modular net-works. Moving on to Sect. 3, a mathematical framework for link streams and the definition of a temporal community adapted for the context are presented. Then, we explain a comprehensive framework called "Mosaic" for creating link streams with communities. Finally, Sect. 4 involves the application of some dynamic com-munity detection algorithms to the benchmark, aiming to uncover their capabil-ities and limitations.

2   Related Works

Assessing and comparing community detection algorithms presents a significant challenge. Although real-world datasets can offer valuable insights, it has been shown that node metadata are not the same as ground truth and that treating them as such induces severe theoretical and practical problems [5].

To overcome this limitation, researchers have developed benchmarks to gen-erate synthetic networks for examining algorithm behavior on networks with diverse predefined properties[6]. Synthetic network Benchmarks enable checking an algorithm against:

– *Definition* of communities: since there is no universal definition of community, a benchmark with its ground truth defines what we want to find and check if the method recognizes it.

– *Stability*: the efficiency of a community detection method can be evaluated by testing it on numerous network instances that share similar characteristics.
– *Scalability*: gradually increasing the network size makes it possible to determine how well the algorithm handles larger and more complex networks.

Numerous network benchmarks have been introduced to establish modular static networks. Stochastic Block Models (SBM, also random planted partition graphs) [7] generate networks where edges between nodes in and within communities depend on a provided probability matrix.

As the need for synthetic temporal networks increased, several methods have been outlined in the literature to generate benchmark graphs for evolving communities. An evolving community scenario is defined as a structure characterized by fundamental events for communities such as birth, death, merging, splitting, expansion, contraction, iterative continuation, and the Ship of Theseus, as illustrated and described in [3].

Temporal network benchmarks have been developed with diverse perspectives and aims, yet they all share a common trait: they generate snapshots that reveal clear community structures. For instance, Granell et al. [8] propose two cyclic scenarios (migration and merge-split), and in each snapshot, communities are defined using SBMs. Bazzi et al. [9] introduce a method for generating multilayer networks with community structures by incorporating an SBM with additional interlayer dependency tensors. However, Cazabet et al. [10] argue that utilizing an SBM independently for generating edges in each snapshot is impractical. Therefore, they have developed a method that allows for evolving structures while maintaining the stability of most edges from one time step to the next.

Differing from all preceding approaches, our proposed benchmark introduces a framework that accomplishes two crucial objectives: 1) It enables the representation of novel scenario description generators that do not necessitate the inclusion of progressively evolving structures, and 2) It facilitates the generation of continuous-timed instantaneous edges while maintaining a low computational cost.

3   Mathematical Framework

3.1   Link Stream

Link streams, the category of temporal networks examined in this study, can be perceived as a collection of vertices denoted by V, which engage with one another at specific instances, and the duration of these engagements is considered negligible.

Based on this definition, we can formulate a link stream mathematically as:

Definition 1. *Link Stream: A link stream L is defined as a triple (V, E, T), where V represents the set of nodes involved in interactions within a defined time domain, $T = [T_s, T_e] \subseteq \mathbb{R}$, and $E \subseteq V \times V \times T$ is the set of temporal edges. Each temporal edge, $l = (u, v, t) \in E$ signifies an instantaneous interaction that took place between node $u \in V$ and node $v \in V$ at time $t \in T$ [4].*

The illustration in Fig. 1 presents a link stream featuring a set of vertices $V = \{v_1, v_2, v_3, v_4\}$ where multiple temporal edges are observed. For example, nodes v_1 and v_4 establish a connection twice within the given time domain, $T = [0, 10)$.
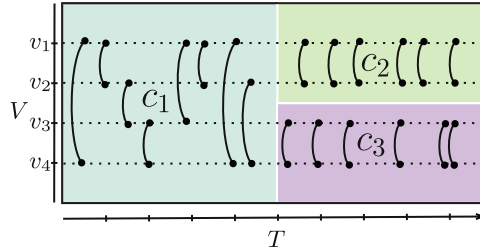
Fig. 1. Modular Link stream: A link stream (V, E, T) is shown, containing 4 nodes $V = \{v_1, v_2, v_3, v_4\}$ interacting several times with each other within the time domain $T = [0, 10)$. A mosaic partitioning, $\mathcal{C} = \{c_1, c_2, c_3\}$ is also observed. This partitioning covers $\{v_1, v_2, v_3, v_4\} \times [0, 10)$ without any overlap.

3.2 Mosaic: A Definition for a Community in Link Streams

Defining a community in a link stream is a challenging task due to the fine-resolution temporal dimension involved. However, any new definition must align with the intuitive understanding of real-world applications. To give an intuition of the meaning of communities in this setting to unfamiliar readers, we illustrate it using a well-known link-stream dataset.

Sociopatterns[1] is a renowned database for real-world link streams acquired in various contexts since 2008 [11]. In these experiments, RFID sensors track real-time proximity, creating co-presence networks between individuals. For example, a substantial dataset comes from a primary school study where 230 pupils and 10 teachers wore sensors for two consecutive days. This study recorded 125,000 face-to-face interactions over 32 h, with a temporal resolution of 20 s. Previous research often viewed its communities as evolving structures influenced by node and edge additions/deletions, identifying them by segmenting time into different slices with various window sizes and then applying dynamic community detection techniques.

In this study, we can observe communities emerging during specific time-frames from students interacting. For instance, students and teachers interact during lecture hours within their respective classes. During lunchtime, students with stronger friendships tend to dine together. An empty community exists at night, indicating no interactions occur during that period.

---

[1] http://www.sociopatterns.org/.

This perspective can be extended to other contexts like people discussing a particular topic on social networks, company meetings, or sports players participating in a match together. Building on this idea, we introduce a new definition called "Mosaic," which is a straightforward adaptation of non-overlapping communities from static networks to link streams.

A "Mosaic" community is defined as follows:

Definition 2. *A Mosaic, denoted as c, is defined as a pair of (nodes, period): $c = (V_c, T_c)$. V_c is set of n nodes denoted as $\{v_1, v_2, \cdots, v_n\}$. $T_c \subset \mathbb{R}$ is an time interval, $T_c = [T_{cs}, T_{ce})$ where, T_{cs} and T_{ce} represent the start and end times of a Mosaic c, respectively. It represents the interval in which nodes V are considered part of the community c.*

According to this definition, each node is assigned to only one community at any given time, and these communities collectively cover the entire link stream; refer to Fig. 1 for an example. In cases where nodes do not interact significantly for a certain duration, they are assigned to an "empty community", c_*. Therefore, a Mosaic partitioning can be defined as follows:

Definition 3. *Mosaic partitioning: Given a link stream $L = (V, E, T)$, $\mathcal{C}$ is a partitioning containing k mosaics, $\{c_1, c_2, \cdots, c_k, c_*\}$, that cover the link stream fully without any overlap. This requirement can be written as follows:*
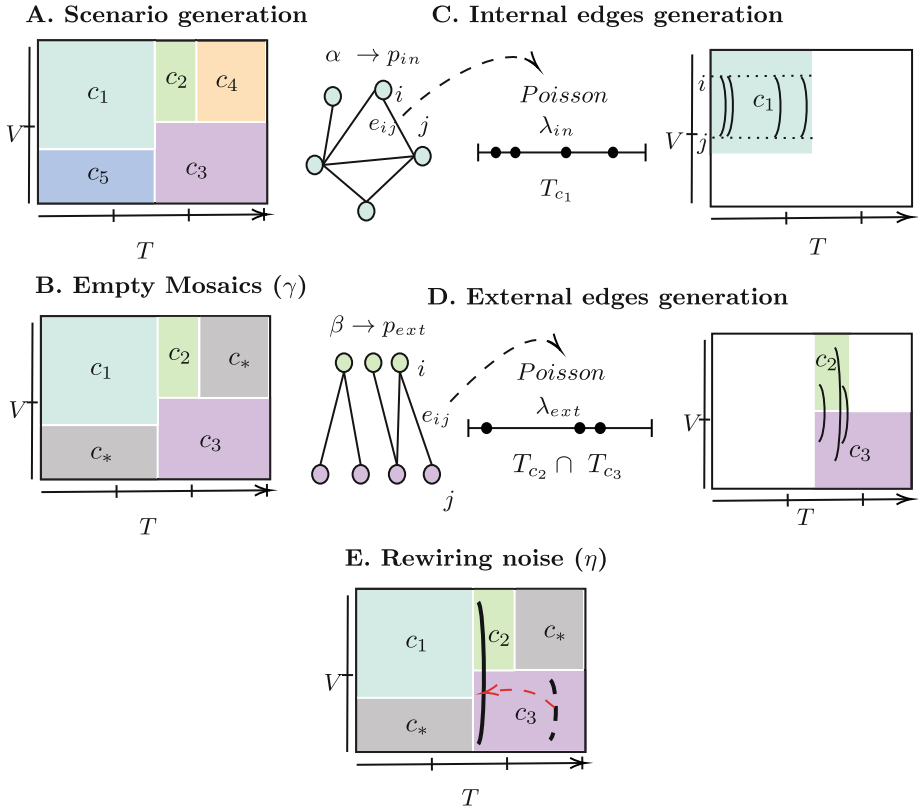
$$\bigcup_{c \in \mathcal{C}} V_c \times T_c = V_L \times T_L \qquad \bigcap_{c \in \mathcal{C}} V_c \times T_c = \varnothing$$

The mosaic c_* stands for an empty community.

3.3   Mosaic Link Stream Benchmark

Now, we delve into discussing the Mosaic Benchmark. The proposed framework follows a straightforward five-step process, as depicted in Fig. 2. The whole procedure is implemented as a user-friendly Python library[2].

---

[2] https://pypi.org/project/mosaic-benchmark/.

Fig. 2. **Random Mosaic Link Stream Benchmark:** This figure illustrates a five-step process. Step A involves generating a scenario. Step B focuses on removing mosaics with a probability of γ to create an empty community named c_*. Steps C and D add internal and external edges, respectively. Finally, in the last step, each edge in the link stream can be rewired with a probability of η.

We can condense these five steps into two primary stages:

1. Scenario description: The user describes communities using a scenario, either ad-hoc or generated from a provided scenario generator (Step A). Additionally, some communities may be emptied to match real-world properties (Step B).
2. Edge generation: Initially, edges are formed within the communities (Step C), followed by establishing connections between communities with overlapping timeframes (Step D). Some edges may be rewired to introduce imperfections into the community structures (Step E).

3.4   Scenario Description

In the proposed framework, we use Mosaic partitioning to generate modular link streams according to a scenario. Mosaic partitioning consists of multiple com-

munities in which nodes interact within and between them. It can be generated using three proposed scenario generators in our Python library: Experimental, Snapshots, and Random. A visualization can be found in Fig. 3.
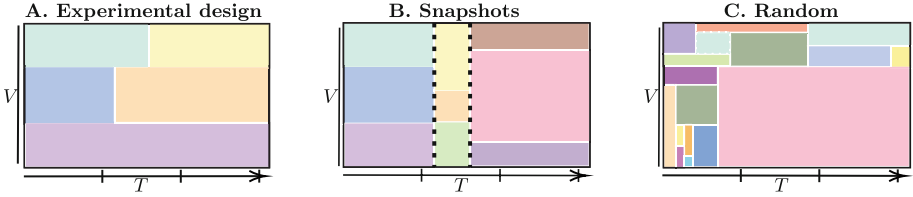
Fig. 3. Mosaic Partitioning Generation This figure illustrates three different approaches for generating Mosaic Partitioning in the Benchmark. **Type A**: User-defined nodes and periods create the desired scenario. **Type B**: The time domain is divided into multiple frames or snapshots, and node sets are randomly assigned to communities within each frame. **Type C**: Communities' time interval and node-set sizes are distributed inhomogeneously, covering the entire link stream.

Experimental: Using an experimental setup, we can intentionally generate various Mosaic communities with specific node counts and time intervals. This enables us to evaluate the performance and precision of a community detection algorithm under straightforward scenarios, gradually increasing complexity to gain insights into the algorithm's capabilities and constraints.

Snapshots: This scenario generator simulates low-resolution temporal networks resembling snapshots. It partitions the time domain into k segments defined by either fixed or variable window sizes. In the fixed case, each segment has a size of $\frac{|T|}{k}$, while in the varying case, the time domain is randomly divided. Then, we establish static communities for each segment, ensuring they consist of at least two or more nodes.

Algorithm 1. Snapshot Scenario Generator

1: **procedure** SNAPSHOT($T, V, k,$ Mode)
2:     Create an empty list $\mathcal{C}$
3:     Divide the time domain into k segments with fixed or varying window sizes and name it S
4:     **for** each s in S **do**
5:         Distribute the nodes V randomly into static communities and name the collection U
6:         **for** each u in U **do**
7:             Add (u, s) to $\mathcal{C}$
8:     **return** $\mathcal{C}$

Random: Generating random scenarios leading to mosaics with different node counts and time intervals is essential for comprehending the capabilities and constraints of a dynamic community detection algorithm. To accomplish this objective, we will suggest a simple procedure for constructing random communities using k iterations in a recursive algorithm. Once we have obtained the algorithm's output, we will proceed to remove communities consisting of just one node or communities that exist for less than a predefined time interval.

Algorithm 2. Random Scenario Generator

1: **procedure** RANDOM(T, V, k)
2:     Create a Mosaic $M = (T, V)$
3:     **for** i in 0 to k **do**
4:         **if** M.sub-mosaics is empty **then**
5:            Divide M into 4 sub-mosaics and store them in M.sub-mosaics
6:         **else**
7:            Select one sub-mosaics from M.sub-mosaics and divide it into 4 sub-mosaics
8:     Flatten the Mosaic M into $\mathcal{C}$
9:     **return** $\mathcal{C}$

Emptying Mosaics. We consider that Mosaics are assigned to an empty Mosaic c_* with a probability of γ. We mean that within this empty Mosaic, no edges can be active that originate from either inside or outside, affecting the nodes contained within it.

3.5 Generating Edges

This part focuses on generating edges between nodes within and across different communities. We will follow two steps: creating a Backbone connectivity network to establish static connections and using the Poisson Point Process to add a temporal dimension above it; refer to Algorithm 3.

Algorithm 3. Edges Generation

1: **procedure** EDGESGENERATION($\mathcal{C}, \alpha, \lambda, \beta$)
2:   Create an empty list E
3:   **for** c in $\mathcal{C} \backslash c^*$ **do**   ▷ Generate internal edges
4:     $p_{in}^c = (|V_c| - 1)^{\alpha-1}$
5:     List $S =$ **BackboneConnectivity**(c,c,p_{in}^c)
6:     **for** e in S **do**
7:       Add **PoissonProcessEdge**(e,P_c,λ^{cc}) to E
8:   **for** (c,c') in $\binom{\mathcal{C} \backslash c^*}{2}$ **do**   ▷ Generate external edges
9:     $p_{ext}^{cc'} = \beta((|V_c| + |V_c'|) - 1)^{\alpha-1}$
10:     List $S =$ **BackboneConnectivity**($c,c',p_{ext}^{cc'}$)
11:     **for** e in S **do**
12:       Add **PoissonProcessEdge**($e,P_c \cap P_{c'},\lambda^{cc'}$) to E
13:   **return** E

Backbone Connectivity Network. This Benchmark assumes that the connectivity between nodes, whether through internal or external edges, remains stable throughout the specified period. This is why we refer to it as the backbone connectivity network. A backbone connectivity network with a parameter p is a random graph in which each edge is present with probability p, independent of others.

We would like to emphasize that for establishing a well-defined internal structure of a community, it is necessary to utilize an appropriate range of values for p. This range's selection should depend on the number of vertices within the community. To achieve this, we will adopt the model described in [10], which provides the formula for p_{in}^c as follows:

$$p_{in}^c = (|V_c| - 1)^{\alpha-1}$$

Here, $\alpha \in (0,1]$ is a hyperparameter named community density coefficient shared between communities. When the value of α is increased, the probability of p_{in}^c also increases, leading to denser clusters. If α is set to 1, each community in Mosaic becomes a clique.

The external probability between two communities c and c' denoted as $p_{ext}^{cc'}$ is defined as:

$$p_{ext}^{cc'} = \beta(|V_c| + |V_c'| - 1)^{\alpha-1}$$

This hyperparameter $\beta \in [0,1]$ is related to "community identifiability," which is shared among all communities. Increasing the value of β results in more external edges between communities, making it more challenging for algorithms to identify each community as a separate cluster. In other words, β controls the external density of backbone connectivity by treating two communities as a single entity.

Poisson Point Process. To simplify the analysis, we assumed that the edges in a given backbone connectivity network follow a memory-less stochastic process

for their activation times. For each edge $e = (i, j)$ in the backbone connectivity network, we generate an independent and identically distributed random Poisson point process with a rate parameter $|T|\lambda$. This rate parameter determines the average number of this edge active times within the time frame T. Then, we use the uniform distribution to distribute this number of occurrences in the selected period. This means the edge time arrivals are uniformly spread over the interval T [12]. To establish connections within a community, we set T equal to T_c and $\lambda = \lambda_{in}^c$. Conversely, when creating the connections between two different communities c and c', T is defined by the overlap between T_c and $T_{c'}$. Furthermore, to generate external edges between communities c and c', we utilize a coefficient $\lambda_{ext}^{cc'}$.

Combining these, to create the temporal dimension, we need a symmetric matrix λ of size $k \times k$, where k represents the number of communities. The main diagonal of this matrix will be utilized for generating internal edges, and non-diagonal elements can be employed for external edges if there is a non-empty time overlap $(T_c \cap T_c' \neq \emptyset)$ between the communities c and c'.

In both the step of generating backbone connectivity networks and adding a Poisson point process layer, each community is handled independently, which can be efficiently parallelized. This enables handling large networks in a reasonable time. Additionally, finding an upper bound for memory and time complexity can not be provided due to dependence both on time and structure.

Rewiring Noise. A prior study [13] proposed that it is possible to distinguish a stable core within communities from random, short-lived fluctuations in real temporal networks. In our Benchmark, edges go through a rewiring process with a probability of $\eta = [0, 1]$ to highlight imperfections in community structures. During this step, for edge (u, v, t) selected for rewiring, we randomly choose two communities $c \neq c_*$ and $c' \neq c_*$ with non-empty time intersection. We then select two nodes $u \in V_c$ and $v \in V_{c'}$ where $u \neq v$, with a timestamp t randomly chosen within the $T_c \cap T_{c'}$ time frame.

4   Experiments

In this section, we use different community detection algorithms on an instance of our framework to assess their performance in identifying communities compared to the ground truth. Among the many dynamic community detection algorithms available, we have selected four that operate by aggregating the link stream into snapshots using window sizes. These algorithms are previously implemented in "tnetwork" Python package [14], as detailed in the existing literature review [3].

The algorithms compared in this paper are the following:

– **No-Smoothing:** the approach involves applying a static algorithm (in this case, the Louvain method) to each snapshot. Then, the most similar communities in consecutive steps are matched based on the Jaccard Coefficient with a coefficient named θ, set to 0.3 here.

– **Implicit-Global:** in this method, the Louvain algorithm is executed at each snapshot, but instead of initializing it with each node in its own community, the previous partition is utilized as the seed.
– **Label Smoothing:** this method first identifies communities in each slice. Then, attempts are made to match communities across different snapshots, forming a survival graph. A community detection algorithm is applied to this survival graph, resulting in dynamic snapshot communities.
– **Smoothed-Graph:** in this approach, the Louvain method is run at each slice t on a graph with a smoothed adjacency matrix defined as follows:

$$\mathrm{A}_{ij}^{t} = \alpha \cdot \mathrm{A}_{ij}^{t} + (1 - \alpha) \cdot C_{ij}^{t-1},$$

where $C_{ij}^{t-1} = 1$ if nodes i and j belong to the same community at step $t-1$, and 0 otherwise.

We considered a link stream L consisting of 100 nodes $|V| = 100$ interacting over the time interval $T = [0, 100)$. Then, we created a scenario using a Random scenario generator(as detailed in 3.4) with an iteration parameter $k = 30$. Additionally, we applied a probability $\mu = 0.2$ to empty Mosaic communities, resulting in $|\mathcal{C}| = 61$ communities with an average node count of $\bar{V}_c = 11.7$ and an average time interval of $\bar{T}_c = 10.2$, excluding the empty community c_*.

For the edge generation phase, we fine-tuned the parameters $\alpha = 0.9$ and $\beta = 0.1$. A high α value signifies strong connections among nodes within the internal backbone connectivity networks, while a low β value indicates sparse connections between communities in the external backbone connectivity networks. This configuration results in clear and distinguishable communities regarding the backbone structure.

To further emphasize this characteristic, we maintained the same values for $\lambda_{in} = 0.4$ and $\lambda_{ext} = 0.1$ for all communities, signifying that, in the Poisson Point Process, the likelihood of an edge forming within a community is higher than it is for an external one. It resulted in 28365 edges. Additionally, we exclude the rewiring process for temporal edges.

Since the chosen dynamic community detection methods are not tailored for link stream cases and are designed for low-resolution temporal networks, we convert our sample modular link stream to snapshots with a relatively small fixed window size 2. Subsequently, we applied community detection algorithms to the aggregated snapshots and extracted these communities for the purpose of comparison and visualization.

As depicted in Fig. 4, it is evident that different algorithms can yield varying interpretations of a benchmark network instance. Assessing the accuracy of community detection algorithms under such circumstances can pose a significant challenge. Similarly, when employing aggregation techniques on a link stream, diverse interpretations may arise. In certain social and political contexts, these variations can be contentious and potentially lead to misleading conclusions.

(a) Ground Truth

(b) No-Smooting

(c) Label smoothing

(d) Implicit global

(e) Smoothed graph
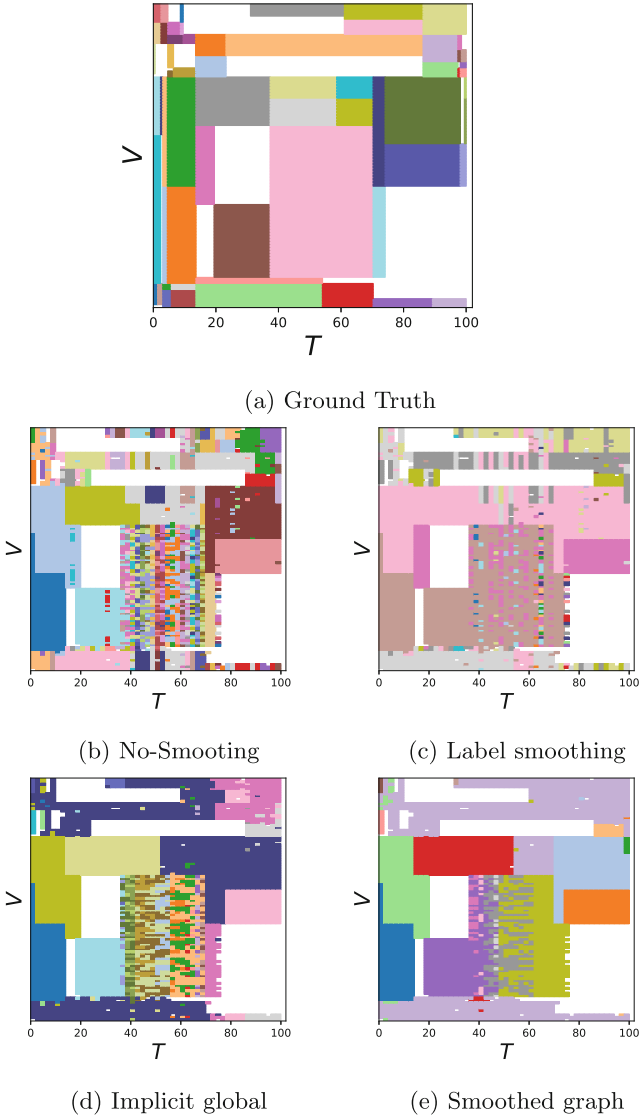
Fig. 4. Experiments: Comparison of partitions obtained using all methods on a sample Random scenario

To conduct a more rigorous experiment, we introduce a new parameter denoted as $\phi = 1 - \alpha = \beta$. A higher ϕ parameter value results in communities being less identifiable. We varied the ϕ value from 0 to 0.5 with the step of 0.1, did the same generation process, and aggregated the link streams to snapshots using a window size of 2, then identified communities ten times with the algorithms.
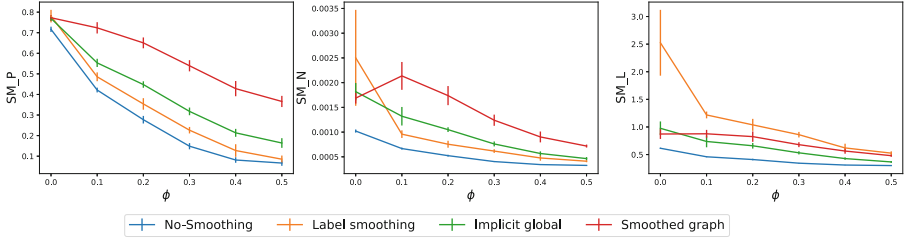
Fig. 5. Experiments: Comparison of smoothness measures obtained using all methods on a sample of a Random scenario.

Smoothness values were determined utilizing the formulas for smoothness presented in the work by Cazabet et al. [10], namely SM-P, SM-N, and SM-L. In all these smoothness metrics, a higher value indicates superior performance. The obtained values were averaged for each ϕ.

As depicted in Fig. 5, in terms of smoothness, two methods have high scores for the three aspects: Implicit-Global and Smoothed-Graph. Label-smoothing has the highest scores in most settings for the SM-L scores, which measure label smoothness. No Smoothing is the least stable in most cases. These results support the findings of other benchmarks presented in [10].

5   Discussion and Conclusions

In summary, we introduced the Mosaic benchmark networks as a new framework for generating modular link streams. These temporal networks enable the evaluation of dynamic community detection algorithms in terms of accuracy and performance. Additionally, we can provide a quantitative assessment based on community definition, stability, and scalability by simulating adaptable ground truth to determine an algorithm's capabilities and constraints. Furthermore, our framework acts as a foundational platform for assessing algorithms tailored for link streams, especially in the later stages of research in this domain.

The time complexity of this framework can benefit from parallelization because it avoids SBM calculations for each small step, and the future does not depend on past or present data. Moreover, when simulating link streams with millions of edges, it is crucial to efficiently organize edge and node storage. Multi-layered hash functions can enhance memory allocation for storing network edges instead of a large edge stream. To create directed modular link streams, we propose using the benchmark network with an asymmetric Poisson rate matrix (λ), along with two additional parameters, α_{out} and β_{out} for edge generation phase. However, it is essential to carefully check if created communities match the definition of communities in real-world directed link streams.

We applied our framework to explore differences among communities identified by various dynamic community detection algorithms. We emphasize that the aggregation method using window sizes has its limitations, and there is a

crucial need for a more comprehensive investigation into effectively handling continuously timed edges rather than aggregating them.

In conclusion, it is important to explore the characteristics of communities formed through the provided scenario generators. This framework incorporates various parameters such as γ for emptying communities, and (α, β, λ) for edge generation. Furthermore, the parameter η plays a crucial role in evaluating an algorithm's robustness. We advocate for further research to enhance the comprehension of dynamic community algorithms, achieved through the fine-tuning of these parameters.

References

1. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. **99**(12), 7821–7826 (2002)
2. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3–5), 75–174 (2010)
3. Rossetti, G., Cazabet, R.: Community discovery in dynamic networks: a survey. ACM Comput. Surv. (CSUR) **51**(2), 1–37 (2018)
4. Latapy, M., Viard, T., Magnien, C.: Stream graphs and link streams for the modeling of interactions over time. Soc. Netw. Anal. Min. **8**, 1–29 (2018)
5. Peel, L., Larremore, D.B., Clauset, A.: The ground truth about metadata and community detection in networks. Sci. Adv. **3**(5), e1602548 (2017)
6. Lancichinetti, A., Fortunato, S.: Community detection algorithms: a comparative analysis. Phys. Rev. E **80**(5), 056117 (2009)
7. Holland, P.W., Laskey, K.B., Leinhardt, S.: Stochastic blockmodels: first steps. Social networks **5**(2), 109–137 (1983)
8. Granell, C., Darst, R.K., Arenas, A., Fortunato, S., Gómez, S.: Benchmark model to assess community structure in evolving networks. Phys. Rev. E **92**(1), 012805 (2015)
9. Bazzi, M., Jeub, L.G., Arenas, A., Howison, S.D., Porter, M.A.: Generative benchmark models for mesoscale structure in multilayer networks. arXiv preprint arXiv:1608.06196, p. 20 (2016)
10. Cazabet, R., Boudebza, S., Rossetti, G.: Evaluating community detection algorithms for progressively evolving graphs. J. Complex Netw. **8**(6), cnaa027 (2020)
11. Cattuto, C., Van den Broeck, W., Barrat, A., Colizza, V., Pinton, J., Vespignani, A.: Dynamics of person-to-person interactions from distributed RFID sensor networks. PLOS ONE **5**, e11596 (2010)
12. Ross, S.M.: Introduction to probability models. Academic Press (2014)
13. Kobayashi, T., Takaguchi, T., Barrat, A.: The structured backbone of temporal social ties. Nat. Commun. **10**(1), 220 (2019)
14. Cazabet, R., Boudebza, S., Jorquera, T.: tnetwork (2023)