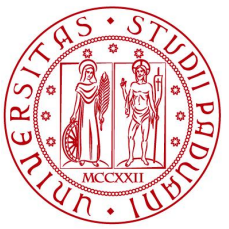


Università
degli Studi
di Padova

Head Office: Università degli Studi di Padova
Department: Information Engineering

Ph.D. course in: Information Engineering
Curriculum: Information and Communication Technologies (ICT)
Series: XXXV, 2023

Efficient and Rigorous Algorithms for the Analysis of Large Temporal Networks

Coordinator: Ch.mo Prof. Andrea Neviani
Supervisor: Ch.mo Prof. Fabio Vandin

Ph.D. Candidate: Ilie Sarpe

Efficient and Rigorous Algorithms for the Analysis of Large Temporal Networks

Abstract

Networks are ubiquitous tools that model many real world systems, ranging from academic collaborations to online markets. A massive amount of data is produced everyday from many systems that can be modeled through the network paradigm. The field of data mining aims at analyzing networked systems to extract meaningful *patterns*, to characterize the behaviour, and enhance our comprehension of the complicated systems being anayzed. Patterns play fundamental roles in defining important primitives that are used in exploratory analyses and specific applications to better understand networks. Modern systems, additionally to the network structure contain richer information about the timing of the interactions over the network, these networks are called *temporal networks*. Given the richer structure of temporal networks, novel patterns and primitives based on such patterns have been introduced in the field of data mining. Unfortunately, computing patterns on temporal networks is often a really hard and challenging task due to the more complicated structure of such networks and their patterns. In this thesis we develop efficient and rigorous algorithms for several primitives based on the computation of different patterns in large temporal networks.

The first patterns we consider are *temporal motifs*. A temporal motif is a small pattern defined by a small topology, capturing the function of the pattern, e.g., on a message network if such topology is a triangle then it captures the communication pattern between three users, and an ordering of its edges, that captures the dynamics of the topology over the temporal network. Analysing temporal motifs in temporal networks is a really challenging task, with the problem being NP-Hard in its general formulation. In this thesis we consider two different primitives based on such patterns.

We first develop exact and approximate algorithms for obtaining a count of an arbitrary temporal motif in a temporal network. Our algorithms are based on collecting small subnetworks of the input temporal network, and carefully combining the counts of the occurrences of the motif in such subnetworks. The exact algorithm combines the counts over subnetworks defined by a cover of the timeline of the temporal network, leading to a really scalable and efficient algorithm for computing exact counts. The approximation algorithm we develop uniformly samples small subnetworks of the input network, and weights occurrences of the temporal motif in the samples to obtain unbiased estimates of the motif count. We then show bounds on the number of samples for our sample algorithm to concentrate within desired accuracy with fixed probability. We then perform an extensive experimental evaluation to show that our algorithms improve over existing

state-of-the-art algorithms, providing therefore practical algorithms to address the computation of the count of a temporal motif over large temporal networks containing up to billions of temporal edges.

Then we propose a novel problem of computing *simultaneously* the counts of multiple temporal motifs, all sharing the same common static topology. This problem is motivated by the fact that the ordering of a temporal motif cannot often be known a priori with high accuracy, especially in exploratory analyses. We then propose a randomized approximation algorithm to obtain high-quality approximation of all the temporal motifs addressing the novel problem introduced. Our sampling algorithm is based on sampling temporal edges on the static graph associated to the temporal network, identifying and weighting opportunely the occurrences of the motifs containing the sampled edge, and obtaining unbiased estimates to the counts of all motifs. We show bounds on the number of samples of the proposed algorithm to obtain concentrated estimates. We then perform a large experimental evaluation to show how such algorithm can be used to address the problem of obtaining the counts of all the motifs mapped on the same static topology, improving significantly existing state-of-the-art algorithms.

The second type of patterns we consider in this thesis are *temporal paths and walks*, such patterns account for the connectivity on temporal networks, both structurally and temporally. These patterns are used to define an important primitive on temporal networks, the *temporal betweenness centrality*. Such centrality measure assigns to each node a value that is based on the fraction of optimal temporal paths (or walks) using a specific node, and nodes with higher temporal betweenness values are more central in spreading processes over the temporal network. We therefore address the problem of the efficient computation of the temporal betweenness centrality of the nodes in temporal networks. To solve this problem we develop a sampling-based approximation algorithm providing rigorous guarantees on its output. Our algorithm is based on sampling pairs of nodes and computing all the optimal paths (or walks) connecting the sampled pairs. We then use advanced tools from concentration theory, based on the empirical values of the estimates obtained, to provide high-quality probabilistic guarantees on the estimates computed by our algorithm. We empirically show how the proposed algorithm achieves tight theoretical guarantees and significantly improves the scalability and the resources used compared to the state-of-the-art exact algorithm, enabling novel analyses that were previously unpractical.

Acknowledgements

After few months that I started my journey (and what a journey!) the pandemics suddenly emerged and everything was shut down. I experienced some difficult times given such a situation (I particularly suffered social distancing), but if I am here willing to explore many other research questions is thanks to all the people constantly supporting and helping me. I am really grateful to all of you and this section is to let you know your importance in my professional and personal development. I will switch to Italian only for parts of the remainder section.

Prima di tutto voglio ringraziare Fabio, senza di te probabilmente non avrei mai intrapreso questo percorso, e soprattutto non mi sarei mai appassionato al mondo della ricerca. Oltre che essere un grande mentore, hai contribuito anche alla mia crescita come persona e non ti sarò mai grato a sufficienza. Continui sempre ad essere una fonte di ispirazione e mi stupisce quanto tu creda in me più di quanto lo faccia io. Inoltre hai supportato tutte le mie scelte, questo mi ha aiutato a rialzarmi e ad andare avanti, sono molto felice di averti avuto come mio supervisore.

Devo poi ringraziare tutti i miei colleghi di laboratorio (passati e presenti): Andrea, Leonardo, Thach, Federico, Dario S., Antonio, Dario P., Fabrizio, Matteo (anche se hai lasciato il lab..) e Paola. Grazie al confronto con voi spesso sono riuscito ad accrescermi dal punto di vista personale, inoltre è sempre divertente essere in laboratorio con voi. Fra i vari colleghi non posso non citare Davide e Diego con i quali abbiamo intrapreso e concluso questo percorso assieme, e condiviso anche i momenti più difficili; Diego inoltre grazie anche per essere un ottimo amico e coautore. Voglio inoltre ringraziare Francesco, per aver assistito alle mie terribili presentazioni di fine anno e darmi sempre consigli molto utili sulla ricerca e la mia carriera. Infine ringrazio Andrea e Geppino, dato che il confronto con voi mi fa sempre capire quanto io ancora abbia da imparare sul mondo della ricerca e vi ringrazio per avermi dato la possiblità di collaborare con voi.

I want then to thank Andrea and Evimaria for helping me with a preliminary version of this thesis with your precious comments.

I am also deeply grateful to Aris for hosting me as a visiting student and being such a great advisor. You were always kind with me, giving me self confidence to investigate the research problem we were trying to solve, I feel really lucky to work with you. At KTH I also met really great colleagues that I have to thank for being really friendly with me and for the good times we spent together, I learned a lot from you: Guangyi, Stefan, and Mohit (you

guys were always there when I needed something), Sijing, Honglian, Ruo-Chun, Tianyi, Thibault and Florian.

Un grazie particolare va alla mia famiglia che mi ha sempre supportato in questo percorso, nonostante io possa essere stato poco presente in molti momenti.

Infine devo ringraziare Andrea, dalla palestra allo suonare assieme nel gruppo, ho sempre la sensazione che la mia vita sia sempre più divertente quando ci sei te intorno e non sai quanto mi ha aiutato tutto ciò in questi ultimi tre anni.

Table of Contents

Chapter 1

Introduction

An enormous amount of data is produced everyday, and the growth of such volume of data is increasing at an exponential rate. Many systems widely spread in our society and most of the actions we perform on a daily basis contribute to such huge data generation. *Data Mining* is a large area of research with an important focus on analyzing and extracting meaningful *patterns* from such large amount of data, a task often called *pattern mining*. Through the analyses of patterns we better understand the real-world systems generating them, and more in general the processes regulating such complicated systems. Unfortunately, extracting meaningful patterns usually comes at a high-computational price. In fact, given the large amount of data available, processing and collecting patterns may be unfeasible in many scenarios. One of the major design goals in pattern mining is to develop algorithms that can *scale* their computation to very massive amounts of data, and are *efficient* in computing a solution that can be used for analysis. Such design goals lead researcher in the field of pattern mining to rely on *approximate* solutions that enable most of the algorithms being both scalable and efficient. An approximate solution, to be of practical interest, should also provide *rigorous guarantees* on the quality of its approximation, i.e., quantify how far from the optimum such solution is. In this work we will develop algorithms for data mining problems guided by the above design challenges.

In data mining there are many structured types of data, for which obtaining the respective patterns is really important and challenging, and *networked* data can be identified as one of the most important. Informally, networks are models where actors in the system are represented as nodes, e.g., users in a social network, and their actions are represented as edges between pairwise nodes, e.g., a friendship over such social network. Such general representation model captures most of the real-world systems we may want to study, such as web markets, social networks, road networks, biological networks, etc. Thanks to the analysis of the *interesting* patterns of a network we are able to identify objects that contribute and regulate functions over the specific network (e.g., user behaviours or influential users). There are many possible definitions of interesting patterns when analyzing networks, and small structures denoted as *motifs* and *paths* are among the most

important patterns used for analyses over networks. *Motifs* are small sub-networks specified by a set of nodes (usually smaller than ten) that capture a specific function of the network to be studied. For example, in biological networks motifs can capture molecular interactions, therefore, by *counting* motifs we can identify if a given set of reaction among pairwise molecules occurs frequently or not in the network, and this count can provide significant insights on the biological system that the network is modelling. Since each motif is associated to a specific function of the network, in many applications we can be given a set of motifs and the algorithm is required to count the occurrences of each motif to capture multiple functions of the network. Many algorithms have been devised to address such counting problems for large networks, both exactly and approximately and under many different constraints, highlighting both the importance of the problem for the data mining community and the technical challenges in addressing this problem in light of the design goals previously discussed. *Paths* are used to capture reachability in networks, and are among the most intuitive and fundamental patterns to be analyzed over networks. A path between a user to another user means that information propagated between the two users in a given network. The analysis of paths finds many applications. In particular paths can be used to identify important nodes in networks by opportunely defining scores, known as *centrality measures*, taking into account the number of paths in networks. As for motifs, there exists a huge literature in efficient and rigorous algorithms for computing paths and centrality measures for networks. Unfortunately, given the increase in complexity of the available data the network model is not sufficiently powerful to capture many aspects of the real-world systems we want to analyze, and therefore most of the algorithms we mentioned cannot be employed when such more complex information is available.

Temporal networks have been proposed in literature as a more powerful model to complement the (static) network model we presented, when temporal information is available. Informally, temporal networks account for the timing of occurrence of the events, modeled by the edges, i.e., each edge is assigned a timestamp, representing the instant at which such event occurred. As for static networks, temporal networks can be characterized by analyzing the *temporal patterns* they contain. Temporal patterns enable the analysis of temporal networks by taking into account the timing of occurrence of events, hence capturing and characterizing the spreading process regulating the network. Temporal dynamics in fact play a fundamental role when using temporal patterns to study temporal networks, since they can provide a wider view and novel insights about many systems. As already discussed, both motifs and paths are of fundamental importance for pattern mining on static networks, therefore their extension naturally emerged also for temporal networks.

Temporal Motifs are temporal sub-networks that can be used to better characterize the dynamics and high-order interactions spreading over a temporal network. As for static networks they find application in many scenar-

ios, from social network analysis to fraud detection, but in addition they also find novel applications thanks to the additional information they are able to capture, e.g., identifying short burst user behaviours. The analysis of temporal motifs in temporal networks is in fact able to highlight the important dynamics occurring over the network and distinguish how such dynamics differ across many networks. For example, in communication networks temporal motifs allow to distinguish between different ways to exchange messages in time, capturing user behaviour and the type of communication. Such precious information comes at a higher price, since identifying such patterns in temporal networks is really costly especially when compared with static networks, posing a major challenge for analyses based on such patterns. Additionally, large temporal networks are really common nowadays with more than tens or even thousands of billion edges networks, rendering such task impractical in most scenarios. Another issue not enabling the analysis of such patterns is that most of the existing algorithms developed for the static network scenario cannot be adapted to work on temporal motifs given their more complicated structure. Given the importance of the problem of studying temporal motifs and the challenges we discussed, in this work we developed efficient and rigorous scalable algorithms for analyzing temporal motifs in large temporal networks to bridge the gap in the existing literature. Our contributions for this problem are as follows:

- In Chapter 3 we address the problem of computing the count of an *arbitrary temporal motif* in a large temporal network. We devise algorithms based on partitioning the timeline of events of the temporal network in small intervals, and opportunely combining the counts obtained in each subnetwork defined by an interval to obtain the final output. In particular, we develop an exact framework, based on a smart partition of the timeline of events of the temporal network. Such framework is embarrassingly parallelizable and can significantly speed up any inherently sequential exact algorithm. We then develop PRESTO, a sampling-based approximation algorithm. PRESTO collects small subnetworks of the input temporal network according to two different sampling strategies on intervals over the timeline of events of the temporal network. On these subnetworks PRESTO carefully weights each instance of the temporal motif to an obtain unbiased estimate of the desired count. We then use tools from concentration theory and variance-aware tail bounds to show upper bounds on the number of samples needed by PRESTO to achieve the desired accuracy, therefore providing high-quality probabilistic guarantees. We then perform an extensive empirical evaluation to show the improvement of our algorithms over the existing state-of-the-art techniques for solving such problem. In particular, we evaluate both the exact and approximate algorithms we developed on very large networks and show they are able to substantially improve the scalability and the efficiency of the computation of temporal motif counts while providing rigorous guarantees on their output. A preliminary version of

this work was published in [Sarpe and Vandin, 2021a].

- Complementary to counting one single motif, in Chapter 4 we propose a novel problem not previously addressed in the literature for temporal networks. In such scenario, we propose to compute all the counts of the temporal motifs sharing a common underlying (static) topology. This is inspired by the problem of counting multiple motifs in static networks, but such problems takes advantage of the structure of temporal motifs. In fact, temporal motifs, informally, are defined by a topology and the dynamics that the temporal motif is capturing. In many scenarios, knowing the topology is trivial (e.g., a triangle or a square) but identifying the correct dynamics to be specified often is associated to some prior knowledge that may not be easily available. Therefore we propose the novel problem of computing *all the counts* of the temporal motifs mapping on the same topology, hence we only require the user to specify a static topology of the temporal motifs of interest. To address such problem we propose ODEN, an approximation algorithm based on sampling edges on the static projection of the temporal network according to different sampling distributions. Using tail bounds accounting for the variance of the variables used for the final estimation we are able to bound the number of samples needed by our algorithm ODEN to concentrate within given accuracy and desired confidence. We then show in practice with an extensive experimental evaluation that our algorithm is able to improve over existing state-of-the-art algorithms for estimating single temporal motif counts when executed on all the motifs of interest in the novel problem defined. This work appeared in [Sarpe and Vandin, 2021b].

The other patterns considered in this work are *temporal paths and walks*. Temporal paths are extended from static paths by taking into account the temporal information of the edges in a temporal network (and a temporal walk is a temporal path where nodes can be visited multiple times), that is, for the information to flow on a path, all edges on the path should occur at increasing time steps. A temporal path (or walk) can capture the information spreading across each sequence of edges, and this finds many applications such as defining *centrality measures* over nodes of the network. A centrality measure captures the centrality of a node with respect to some topological (and temporal) aspect of the network. Intuitively, highly central nodes have a role in the network that is more important than those with a small centrality value. *Temporal betweenness centrality* is based on the fraction of optimal temporal paths (or walks) flowing through a specific node. While many practical algorithms exist for static networks, this problem was not practically addressed on temporal networks due to the theoretical complexity of such task, therefore preventing the analyses of betweenness centralities of many temporal networks. To this end our contribution to improve the state-of-the-art is as follows:

- In Chapter 5 we propose ONBRA, an efficient approximation algo-

rithm based on sampling. ONBRA can be used on many definitions of temporal betweenness centrality according to how the *optimal* temporal paths (or temporal walks) are defined, since in temporal networks many optimality criteria can be adopted. The sampling strategy we propose is based on a previously proposed strategy for static graphs, but adapting such approach on temporal networks poses novel challenges to be addressed. In particular, we develop novel algorithms working on temporal betweenness centrality for shortest temporal paths, and shortest δ-restless temporal walks, i.e., two optimality criteria on which the temporal betweenness centrality can be defined. Additionally, ONBRA can be used on many other optimality criteria. ONBRA provides tight bound on the approximation guarantees it achieves, to obtain this results, we use bounds based on the *empirical* values of the estimates computed to bound the maximum error obtained by ONBRA when executed with a fixed sample size. We then perform an extensive experimental evaluation and we show how ONBRA enables the *practical* rigorous estimation of temporal betweenness centrality values in temporal networks. In particular, we show that our algorithm provides tight bounds of the estimates computed and that it can be used to obtain high-quality outputs requiring a fraction of time and memory compared to the existing state-of-the-art exact algorithm. This work appeared in [Santoro and Sarpe, 2022].

The organization of the rest of this thesis is as follows. In Chapter 2 we discuss basic definitions and theoretical tools that will be used throughout the development of this thesis. In Chapter 3 we discuss exact and approximate algorithms for obtaining the count of a temporal motif in a temporal network. In Chapter 4, we introduce a novel problem about counting multiple temporal motifs in a temporal network under specific constraints, and we present ODEN an approximation algorithm to address such problem. In Chapter 5 we address the problem of obtaining high-quality estimates of the temporal betweenness centrality values of all the nodes in a temporal network, and we present ONBRA, our approximation algorithm to address such problem. Finally in Chapter 6 we discuss the achievements obtained in this work and conclude with some final remarks and future directions.

Chapter 2

Preliminaries

In this chapter we will present the fundamental concepts and definitions used throughout this thesis. We start in Section 2.1 by introducing networks, we first discuss definitions for static networks in Section 2.1.1, and then present how these are adapted to temporal networks in Section 2.1.2. In Section 2.2 we discuss the general framework that will be used to derive estimates in our sampling algorithms. Finally, in Section 2.3 we discuss the main tools from concentration theory that will be employed for the analyses of our algorithms. We conclude summarizing the notation in Section 2.4.

2.1 Preliminaries on Networks

In this thesis we will focus on the development of data mining algorithms for problems on temporal networks, therefore *graphs* or *networks*[1] play a key role in the development of this work. We start by first presenting notation and definitions related to *static* networks, in Section 2.1.1, and we discuss definitions related to the task of counting *motifs* and finding the *betweenness centrality* values of the various nodes. Then in Section 2.1.2 we extend such definitions to temporal networks, namely introducing *temporal motifs* and *temporal betweenness centrality*. With this approach, the reader can appreciate the main differences and the additional complexity that temporal networks introduce over static networks.

2.1.1 Static Networks

Definition 2.1. *A* directed *(static) network is a pair* $G = (V_G, E_G)$ *where* $V_G = \{v_1, \ldots, v_n\}, |V_G| = n$ *and* $E_G = \{(v_i, v_j) : v_i \neq v_j, v_i, v_j \in V_G\}$. *Such network is said to be* undirected *if* $E_G = \{\{v_i, v_j\} : v_i \neq v_j, v_i, v_j \in V_G\}$.

In the above, V_G is often denoted as the set *nodes* of the network, and E_G as the set of (un)directed *edges*. Given that directed networks are a more

[1] While graphs are often used to denote undirected networks, in this thesis we will use the terms graphs and networks interchangeably.

general representation we will be considering such networks in the following discussion. Note also that it is always possible to represent an undirected network with a directed one by adding two edges, namely (u, v) and (v, u) for each edge $\{u, v\} \in E_G$.

2.1.1.1 Subgraph Isomorphism and Motifs

Given a graph $G = (V_G, E_G)$ we say that $G' = (V', E')$ is a *subgraph* of G, denoted with $G' \subseteq G$, if it holds that $V' \subseteq V_G, E' \subseteq E_G : \forall e = (u, v) \in E', u, v \in V'$.

Definition 2.2. *Given a set $V' \subseteq V_G$ we say that $G[V'] = (V', E')$ is the* induced subgraph by the set of nodes *V' if $E' \subseteq E_G, \forall e = (u, v) \in E' \Leftrightarrow u \in V', v \in V'$.*

Similarly, given a set $E' \subseteq E_G$ of edges we say that $G[E'] = (V', E')$ is the *induced subgraph by the set of edges E'* if $V' \subseteq V_G, \forall e = (u, v) \in E' \Leftrightarrow u \in V', v \in V'$. A fundamental problem in graph-mining is to identify graphs that are similar in their topological structure, as captured by network isomorphism.

Definition 2.3. *Given $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ we say that the two networks are* isomorphic *(denoted with $G_1 \sim G_2$) iff there exists a bijection f on the nodes (i.e., $f : V_1 \mapsto V_2$) such that:*

$$e_1 = (u, v) \in E_1 \Leftrightarrow e_2 = (f(u), f(v)) \in E_2 \; \forall e_1 \in E_1, e_2 \in E_2$$

Given two graphs G_1 and G_2, without assuming any particular topology of such graphs (e.g, planarity [Hopcroft and Wong, 1974]), determining if $G_1 \sim G_2$ is a really challenging problem that cannot be easily solved on large networks[2]. A different problem, as important and closely related to the graph isomorphism, is the *subgraph* isomorphism problem that asks, given a target graph G_1 and a query graph G_2 if there exists a subgraph $G_1' \subseteq G_1$ (often required to be induced by its set of nodes) such that $G_1' \sim G_2$. This problem is known to NP-hard in its general formulation [Cook, 1971], while is polynomially solvable for specific classes of target (e.g., see the work by Eppstein [2002]) or query (e.g., see the work by Williams et al. [2014]) graphs. This problem is of practical interest since in many applications query subgraphs capture the function an analyst may want to identify in the given target graph. For example, in networks representing molecular interactions a query graph (or motif) with k nodes may represent a target reaction among a fixed set of k molecules of the network. In addition to identifying if there exists a subgraph that interacts as captured by the query graph, it is usually of practical importance to *count* how many interactions shaped by the motif's structure occur over the network, as captured by the following definition.

[2]To state of the art, the problem is still not known to be NP-complete [Köbler et al., 1993]

Definition 2.4. *Given an input graph $G = (V_G, E_G)$ and a query graph, or motif, $Q = (V_Q, E_Q)$, we say that $C_Q = |\{G[V'] : \exists V' \subseteq V_G, G[V'] \sim Q\}|$ is the* count *of the motif Q in G.*

Differently from existing literature, in this chapter we consider as *motif* any subgraph isomorphic to the query graph. In existing works such as in [Milo, 2004] instead, a subgraph is flagged as a motif only if its count is found to be statistically significant with respect to a particular random model. The counts of motifs can be used to characterize networks in many different scenarios [Ribeiro et al., 2022] and define important scores used to compare networks across different domains (e.g., clustering coefficient and its high-order extensions [Yin et al., 2018]). To address the counting problem many algorithms have been proposed, exact algorithms usually require exponential time while approximate approaches can achieve better trade-offs between time and accuracy [Ribeiro et al., 2019].

2.1.1.2 Paths and Betweenness Centrality

Metrics that enable the comparison between different networks or better understanding of a network's structure are key tools for data mining on networks. A family of important metrics are centrality measures for nodes, i.e., functions that assign to each node a score capturing its centrality in the network. *Betweenness centrality* [Freeman, 1977] is one of the most important centrality measures, based on the notion of *paths*,

Definition 2.5. *Given a graph $G = (V_G, E_G)$, a* path $\mathsf{P} = \langle e_1 = (u_1, v_1), \ldots, e_\ell = (u_\ell, v_\ell)\rangle$ *is a sequence of edges $e_i \in E_G, i \in [1, \ell]$ s.t., $u_i = v_{i-1}, i \in [2, \ell]$.*

In the above path P is also said to go from u_1 to v_ℓ. If the length ℓ of the path P is minimum, i.e., $\nexists \mathsf{P}'$ from u_1 to v_ℓ of length $\ell' < \ell$, then P is said to be a *shortest path* from u_1 to v_ℓ. Given a path $\mathsf{P} = \langle e_1 = (s, v_1), \ldots, e_\ell = (u_\ell, z)\rangle$ we say that a node $v \in V$ is *internal* to P if it holds that $\exists e_j, j \in [1, \ell] : e_j \in \mathsf{P}, v \in e_j, v \neq s, v \neq z$. Given a pair of nodes $s, z \in V, s \neq z$ we will denote with $\sigma_{s,z}$ the number of *shortest* paths from s to z. Similarly, given a node $w \in V$ we will denote with $\sigma_{s,z}(w)$ the number of shortest paths from s to z to which w is internal to.

Definition 2.6. *Given a network $G = (V_G, E_G)$, the* normalized betweenness centrality *of a node $w \in V_G$ is defined as,*

$$b(w) = \frac{1}{n(n-1)} \sum_{s,z \in V^2 : s \neq z} \frac{\sigma_{s,z}(w)}{\sigma_{s,z}}.$$

Such statistic can be computed exactly for every node in the network in polynomial time in the size of the network using Brandes [2001] (in $O(n|E_G|)$ time on unweighted networks).

2.1.2 Temporal Networks

Temporal networks, have been proposed in literature as extension of static networks. Since there are many ways of defining the temporal properties of a graph, many models have been proposed in literature [Holme and Saramäki, 2012]. In this work we will focus on the model that arguably is the most used in practice, providing a fine-grain representation of temporal properties, which can be derived from the general model of link streams introduced by Latapy et al. [2018]. We therefore present definitions related to the model we adopted.

Definition 2.7. *A temporal network T is a pair $T = (V_T, E_T)$ where $V_T = \{v_1, \ldots, v_n\}, |V_T| = n$ and $E_T = \{(u, v, t) : u, v \in V, u \neq v, t \in \mathbb{R}^+\}, |E_T| = m$.*

In this work we will always consider temporal networks as directed networks. Extending some definitions already presented: given a temporal network $T = (V_T, E_T)$ we say that a temporal network $T_1 = (V_1, E_1)$ is a *temporal subnetwork* of T (denoted with $T_1 \subseteq T$) if $V_1 \subseteq V_T, E_1, \subseteq E_T : \forall e = (u, v, t) \in E_1 \iff u, v \in V_1$. We define the *induced subnetwork* $T[V']$ of T by a set of nodes $V' \subseteq V_T$ as $T[V'] = (V', E'), E' = \{(u, v, t) : u \in V', v \in V'\} \subseteq E_T$. Given a temporal network T, it is possible to obtain a *static representation* of such network ignoring the temporal information of its edges,

Definition 2.8. *Given a temporal network $T = (V_T, E_T)$ we denote with $G_T = (V_T, E_{G_T})$ its* projected static network *where $E_{G_T} = \{\{u, v\} : \exists (u, v, t) \in E \vee (v, u, t) \in E\}$.*

Note that in the above definition in addition to the timing of the events we also ignore the directions of the edges. It is worth mentioning that G_T is often a lossy representation of the network T, since it ignores the timing of the events in the entire network. Therefore, patterns defined on temporal networks need to account for the timing of the various events of the network, as described in the next sections.

2.1.2.1 Temporal Motifs

As for static networks where small motifs can be used to characterize important network properties, in temporal networks *temporal motifs* can be used to study and better understand the real world networks we analyze through such patterns. Given that temporal networks are richer than static networks in their model, there are many possible ways of defining temporal patters [Liu et al., 2021]. In this work we considered temporal motifs as defined by Paranjape et al. [2017] that is one of the most used definition of temporal motif in many applications.

Definition 2.9. *A k-node ℓ-edge* temporal motif *M is a pair $M = (\mathcal{K}, \sigma)$ where $\mathcal{K} = (V_\mathcal{K}, E_\mathcal{K})$ is a directed and weakly connected* multigraph *where $V_\mathcal{K} = \{v_1, \ldots, v_k\}, E_\mathcal{K} = \{(x, y) : x, y \in V_\mathcal{K}, x \neq y\}$ s.t. $|V_\mathcal{K}| = k$ and $|E_\mathcal{K}| = \ell$ and σ is an ordering of $E_\mathcal{K}$.*

We briefly comment the above. Temporal motifs are *small* patterns of interest to the analyst (i.e, $k, \ell \leq 10$ usually). $\mathcal{K}$ captures the structure of the motif through a multigraph, hence it can contain multiple edges between the same pair of nodes. In addition, since it is a directed graph, it needs to be weakly connected[3]. The ordering σ instead captures the temporal *dynamics* that the motif represents, e.g., if $\mathcal{K}$ is a directed path $\{e_1 = (u, v), e_2 = (v, w)\}$, σ can denote a time respecting path iff $\sigma = \langle e_1, e_2 \rangle$. Note that a k-node ℓ-edge temporal motif $M = (\mathcal{K}, \sigma)$ is also identified by the sequence $\langle (x_1, y_1), \dots (x_\ell, y_\ell) \rangle$ of ℓ edges ordered according to σ. Temporal motifs therefore play a fundamental role in characterizing both the *structure* and the *dynamics* of patterns to be observed in temporal networks, as captured by the following definition.

Definition 2.10. *Given a temporal network $T = (V, E)$ and $\delta \in \mathbb{R}^+$, a* time ordered *sequence $S = \langle (x'_1, y'_1, t'_1), \dots, (x'_\ell, y'_\ell, t'_\ell) \rangle$ of ℓ unique temporal edges from T is a* δ-instance *of the temporal motif $M = \langle (x_1, y_1), \dots, (x_\ell, y_\ell) \rangle$ if:*

1. *there exists a bijection $f : V_{T[S]} \mapsto V_{\mathcal{K}}$ on the vertices such that $f(x'_i) = x_i$ and $f(y'_i) = y_i$, $i = 1, \dots, \ell$; and*
2. *the edges of S occur within δ time, i.e., $t'_\ell - t'_1 \leq \delta$.*

In the above we slightly abused the notation by denoting with S both the sequence of edges and the set of edges obtained from such sequence, and $T[S]$ denotes the nodeset of the temporal subnetwork induced by S. Note that a δ-instance captures both the topology of the motif (imposing the constraint that the subgraph induced by S must be isomorphic to $\mathcal{K}$) but also the dynamics of the motif according to σ (edges are mapped to the multigraph *only* if they respect σ). The second constraint in Definition 2.10 requires all edges of the instance to occur within δ-time: this a common constraint to explore temporal correlation and is a parameter set by the analyst related to how fast information spreads in the network being analyzed.

Definition 2.11. *Let T be a temporal network, and M a temporal motif, $\delta \in \mathbb{R}^+$. The* count *$C_M(\delta)$ of a temporal motif M is $C_M(\delta) = |\{S : S$ is a δ-instance of M in $T\}|$.*

In this thesis we are interested in finding efficient algorithms for *counting* temporal motifs, i.e., finding the value (or an accurate estimate) of $C_M(\delta)$. As for static networks, obtaining such value is generally hard. One important and interesting result about the complexity of such task is that there are motifs that can be counted in polynomial time on static networks, while identifying their respective temporal motif in temporal networks becomes NP-Hard [Liu et al., 2019]. Related to the task of counting motifs, in Chapter 3 we will discuss algorithms for both exact enumeration and rigorous

[3]Recall that a directed (multi)graph $\mathcal{K}$ is said to be weakly connected if $\forall (u, v) \in V^2_{\mathcal{K}}, u \neq v$ if we consider the *undirected* multigraph of $\mathcal{K}$ obtained by ignoring edge directions, there should exists a path between u and v in such graph.

estimation of a count of a motif in temporal networks. In addition, in Chapter 4 we will discuss a randomized approximation algorithm for estimating multiple temporal motif counts in temporal networks, under constrained topologies of such motifs.

2.1.2.2 Temporal Betweenness Centrality

As for static networks, the importance of nodes in a network can be identified through the analysis of centrality measures. In this thesis we are particularly interested in the *temporal betweenness centrality*. As we will see, defining such score is much more challenging in temporal networks than in static networks. We start by introducing the definitions of temporal path and temporal walk.

Definition 2.12. *Given a temporal network T, a* temporal path P *is a sequence* $\mathsf{P} = \langle e_1 = (u_1, v_1, t_1), e_2 = (u_2, v_2, t_2), \ldots, e_k = (u_k, v_k, t_k)\rangle$ *of k edges of T ordered by increasing timestamps, i.e., $t_i < t_{i+1}, i \in \{1, \ldots, k-1\}$, such that the node v_i of edge e_i is equal to the node u_{i+1} of the consecutive edge e_{i+1}, i.e., $v_i = u_{i+1}, i \in \{1, \ldots, k-1\}$, and each node $v \in V$ is visited by* P at most *once. A* temporal walk *is a temporal path where we drop the constraint of a node being visited at most once.*

Note that in a temporal path we are accounting for the timing of the various edges of the sequence defining the path. While we are requiring the edges on the above paths (or walks) to be *strictly* increasing with respect to their timestamps (i.e., $t_{i+1} > t_i, i = 1, \ldots, k-1$), the techniques we will present can be adapted to work under non-strictly increasing case (i.e., $t_{i+1} \geq t_i, i = 1, \ldots, k-1$) depending on the paths (or walks) considered. As for static networks, let the first node of the path be $s \in V$ and let the last one be $z \in V$, we then say that the path goes from s to z. A node is said to be *internal* to path if it appears on a edge in P and is different from s or z. In a temporal network a path or a walk from s to z can be *optimal* according to different criteria, as described next,

Definition 2.13. *A* temporal path (or walk) $\mathsf{P} = \langle e_1 = (s, v_1, t_1), e_2 = (u_2, v_2, t_2), \ldots, e_k = (u_k, z, t_k)\rangle$ *is said to be:*

- Shortest*: if k is minimum among all paths (or walks) connecting s to z.*
- Foremost*: if t_k is minimum among all paths (or walks) connecting s to z.*
- Fastest*: if $t_k - t_1$ is minimum among all paths (or walks) connecting s to z.*
- Shortest δ-restless*: if given $\delta > 0$ it is shortest and $t_j - t_{j-1} \leq \delta, j \in [2, k]$.*

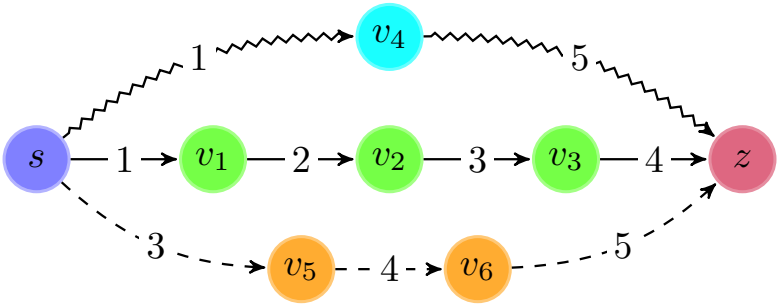


Figure 2.1: Optimality criteria for temporal paths from s to z. Starting from the top, the first path is shortest, the second path is foremost, and the last one is fastest. For $\delta \in [1, 4)$ the path on the bottom is also shortest δ-restless, while for $\delta \geq 4$ the optimal path is the one on the top.

In Figure 2.1 we show an example of such optimality criteria. Given an optimality criterion OPT and two nodes s, z we will denote with $\sigma_{s,z}^{OPT}$ the number of optimal temporal paths (or walks) connecting s to z, and given a node $w \in V_T$ we will denote with $\sigma_{s,z}^{OPT}(w)$ the number of optimal paths (or walks) connecting s to z to which w is internal to. Then we define the temporal betweenness centrality of a node w as follows.

Definition 2.14. *Given a temporal network $T = (V_T, E_T)$, an optimality criterion OPT over paths or walks, the* normalized temporal betweenness centrality *of a node $w \in V_T$ is*

$$b(w) = \frac{1}{n(n-1)} \sum_{s,z \in V_T^2 : s \neq z} \frac{\sigma_{s,z}^{OPT}(w)}{\sigma_{s,z}^{OPT}}.$$

Computing such scores for each node in the temporal network, differently from the static case, is challenging [Buß et al., 2020]. Importantly, some optimality criteria lead to the problem of computing statistics that are #P-Hard, therefore we will only focus on those criteria leading to polynomial-time formulations. Interestingly, for some formulations where counting paths is #P-Hard, counting the corresponding optimal walks leads to polynomial time-solvable formulations [Thejaswi et al., 2020, Rymar et al., 2021], therefore in such cases we focus on optimal walks. In Chapter 5 we discuss an approximation algorithm we developed for estimating the betweenness centrality of the nodes in a temporal network. Such algorithm can be used for shortest paths and shortest δ-restless walks, and it can be adapted to work under other optimality criteria (e.g., some of the ones here presented), when they are not #P-Hard computable.

2.2 Sampling Algorithms

In this thesis we will be developing efficient and rigorous algorithms for several *counting* problems on temporal networks. The technique we will often

leverage to obtain high-quality approximation algorithms is *sampling*. Here we state the general framework that is adapted to the different problems in the different chapters of this thesis.

In most of the data-mining problems on temporal networks we address, we are concerned with identifying a suitable sample space $\mathcal{X} = \{x_1, \ldots, x_{|\mathcal{X}|}\}$, $|\mathcal{X}| < \infty$, of objects to be *sampled* according to a specific discrete probability distribution ρ. We will be able to frame our estimates as follows. Let $f \in \mathcal{F}$, and $\forall f \in \mathcal{F}$ let $f : \mathcal{X} \mapsto [a, b] \subset \mathbb{R}$. Let $x \sim_\rho \mathcal{X}$[4], we derive estimates $\hat{X}_f(x)$ that depend on $f(x)$ and ρ, denoted by:

$$\hat{X}_f(x) \propto (f(x); \rho) \tag{2.1}$$

The way we will sample objects from $\mathcal{X}$ (e.g., leveraging on different sampling techniques such as importance sampling [Tokdar and Kass, 2009]), can change the set $\mathcal{F}$, and how the estimate $\hat{X}_f(x)$ depends on x and ρ. For example, on the problem of counting a single temporal motif we will define the set $\mathcal{X}$ with a suitable set of subnetworks of T, f will sum over the δ-instances of a sampled subnetwork $x \in \mathcal{X}$ weighting each instance depending on ρ, and $|\mathcal{F}| = 1$ since we will be concerned on estimating a single motif count. On other problems $\mathcal{F}$ will be more complicated but still each $f \in \mathcal{F}$ will be efficiently computable given an element $x \in \mathcal{X}$ sampled according to the known ρ (since samples will be small compared with the entire input space). According to the sampling strategy we will adopt to collect elements in $\mathcal{X}$, the distribution ρ can be different in the problems addressed, but we will show that the estimator in Equation (2.1) is *unbiased* with respect to the desired quantity to be estimated (e.g., counts of temporal motifs or node centralities).

Assuming that $\hat{X}_f$ is an unbiased estimate of some quantity μ_f, i.e., $\mathbb{E}_{x\sim_\rho \mathcal{X}}[\hat{X}_f(x)] = \mu_f$, (e.g., μ_f can be a count of a temporal motif M) then it is straightforward to note that given a sequence of s independent and identically distributed variables $(\hat{X}_f^j)_{j=1}^s$ the sample average $\bar{X}_f = 1/s \sum_j \hat{X}_f^j$ maintains the property of being an unbiased estimator of the desired quantity $\mu_f, f \in \mathcal{F}$. One important question that we will be interested in answering is how many samples we need to guarantee that our estimates are well-concentrated. In particular,

- When $|\mathcal{F}| = 1$ we will be interested in ensuring that the quantity $|\bar{X} - \mu|$ is bounded by some θ with high probability.
- When $|\mathcal{F}| > 1$ we will be interesting in bounding:

  $$\sup_{f\in\mathcal{F}} |\bar{X}_f - \mu_f|$$

  that is a fundamental quantity in concentration and statistical learning theory, known as *supremum deviation*.

[4]This means that x is sampled from $\mathcal{X}$ according to ρ.

Depending on the problems we will address we will discuss both algorithms that compute *relative ε-approximations*, i.e., $\forall f \in \mathcal{F}\ |\bar{X}_f - \mu_f| \leq \varepsilon\mu_f$ or *absolute ε-approximations*, i.e. $\forall f \in \mathcal{F}\ |\bar{X}_f - \mu_f| \leq \varepsilon$, both with probability greater than $1-\eta, \eta \in (0,1)$. In some chapters we will be more interested in characterizing the number of samples s to be collected such that the above quantities are bounded within the desired ε provided by the user. In other chapters we may be interested in finding the best ε given a fixed sample size. To deal with all the scenarios mentioned we borrow tools from concentration theory.

2.3 Concentration Tools

In this section we describe the main tools from concentration theory used in this thesis. In Section 2.3.1 we discuss the concentration of the estimates based on the variables $\hat{X}_f, f \in \mathcal{F}$ when $|\mathcal{F}| = 1$, in particular we present the main concentration inequalities used in the following chapters of this thesis. In Section 2.3.2 we present how to address the problem of evaluating the concentration of the estimates when $|\mathcal{F}| > 1$.

2.3.1 Concentration of a Single Function

We now discuss the case $|\mathcal{F}| = 1$, next we will discuss how to extend such techniques to the case $|\mathcal{F}| > 1$. Given a function $g : \mathcal{Y}^s \mapsto \mathbb{R}^+$ and $X_1, \ldots, X_s$ i.i.d random variables taking values in $\mathcal{Y}$ (in our case $\mathcal{Y} = [a,b] \subset \mathbb{R}$), in concentration theory we are interested in bounding the probability of an arbitrary function g computed over the X_i's, i.e., $g(X_1, \ldots, X_s)$, of being to far from its mean $\mu = \mathbb{E}_{X^s \sim \mathcal{Y}^s}[g(X_1, \ldots, X_s)]$. In this thesis we are interested in g when such function is simply the sample average., using the notation already introduced each $X_i = \hat{X}_f^i$ and $X = \bar{X} = g(X_1, \ldots, X_s) = 1/s \sum_i X_i$. Therefore we will be interested in characterizing $\mathbb{P}[|X - \mu| \geq \theta]$ where usually $\theta = \varepsilon$ or $\theta = \varepsilon\mu$. It is not hard to show that such probability will approach 0 as $s \to \infty$, given the central limit theorem and the fact that all quantities involved are finite. However, this observation is not useful when we deal with *finite* values of s, i.e., we are interested in the non-asymptotic behaviour of such estimator. Two important scenarios questions are of interest to this thesis:

1. Given a confidence parameter $\eta \in (0,1)$ and an accuracy parameter $\varepsilon > 0$ is there a bound on s such that $\mathbb{P}[|1/s \sum_i X_i - \mu| \geq \varepsilon\mu] \leq \eta$?

2. Given a sample size s and confidence parameter $\eta \in (0,1)$, is there a bound ε such that $\mathbb{P}[|1/s \sum_i X_i - \mu| < \varepsilon] > 1 - \eta$?

The tools we employ to deal with the above cases, are known as concentration inequalities. We discuss the main tools used in this work and how to adapt them to the two cases above. We start by stating one of the most used concentration inequalities, known as Hoeffding's inequality:

Lemma 2.1 ([Hoeffding, 1963]). *Let $X_1, \dots, X_s$ be independent random variables such that for all $1 \leq i \leq s$, $\mathbb{E}[X_i] = \mu$ and $\mathbb{P}(a \leq X_i \leq b) = 1$. Then*

$$\mathbb{P}\left(\left|\frac{1}{s}\sum_{i=1}^{s} X_i - \mu\right| \geq \theta\right) \leq 2\exp\left(-\frac{2s\theta^2}{(b-a)^2}\right)$$

With such tool we can already provide an answer to the two scenarios above, in particular:

1. If we fix $\theta = \varepsilon\mu$ and given $\varepsilon > 0, \eta \in (0,1)$ we get that if:

   $$s \geq \frac{(b-a)^2}{2\varepsilon^2\mu^2}\ln\left(\frac{2}{\eta}\right)$$

   then the mean estimator is within $\varepsilon\mu$ from its expectation with probability $> 1-\eta$,

2. If we fix $s, \eta \in (0,1)$ then we get that with probability $> 1-\eta$:

   $$\left|\frac{1}{s}\sum_{i=1}^{s} X_i - \mu\right| \leq \sqrt{\frac{(b-a)^2\ln(2/\eta)}{2s}}.$$

   So if we set ε to the right-hand side of the above equation we get the desired bound.

Note that such bounds have a practical impact on the algorithms we will develop since s in our framework denotes the number of samples we need to collect in our algorithms. Collecting more samples is usually expensive and comes at a high price of slowing down the algorithms. Therefore, we are also concerned in finding sufficiently good bounds in the above two scenarios, that often are far from those provided by Hoeffding's inequality. Such inequality in fact is based on the (rather pessimistic) assumption that the variance of the variables $X_1, \dots, X_s$ is maximal. It can be proven in fact that for a random variable taking values in $[a,b]$ its variance is bounded by $(b-a)^2/4$ (known as Popoviciou's inequality). Hence if we the variance of a random variable is significantly smaller than such bound, we can improve significantly the results based on Hoeffding's inequality, as we also discuss in Chapter 3. The main tool we will leverage in such scenario is the so called Bennett's inequality:

Lemma 2.2 ([Bennett, 1962]). *For a collection $X_1, \dots, X_s$ of independent random variables satisfying $X_i \leq M$, $\mathbb{E}[X_i] = \mu$ and $\mathbb{E}[(X_i-\mu)^2] = \sigma^2$ for $i = 1, \dots, s$ and for any $\theta \geq 0$, the following holds*

$$\mathbb{P}\left(\left|\frac{1}{s}\sum_{i=1}^{s} X_i - \mu\right| \geq \theta\right) \leq 2\exp\left(-s\frac{v}{B^2}h\left(\frac{\theta B}{v}\right)\right)$$

where $h(x) = (1+x)\ln(1+x) - x$, $B = M - \mu$ and $v = \sigma^2$.

We will also show that such result holds when a good bound on σ^2 is known (in Chapter 3). The application of the above result to the two scenarios we discussed is slightly more involved, therefore we will discuss it in the chapters where we need to derive the desired guarantees. While sometimes such result may be sufficient to derive sharp guarantees, leveraging on theoretical upper-bounds on the variance may still result in the bounds obtained through such computation not being tight enough for the estimation task considered. Therefore, in such cases we may resort on more involved bounds, taking into account the *empirical variance* of the variables considered, known as *empirical bounds*. Here we state the bound that we will use in Chapter 5.

Theorem 2.1 ([Maurer and Pontil, 2009]). *Let $X_1, \dots, X_s$ be i.i.d. random variables with values in $[a, b]$ and let $\eta > 0$. The with probability at least $1 - \eta$ in the vector $\mathbf{X} = (X_1, \dots, X_s)$ we have:*

$$\left| \mathbb{E}\left[\frac{1}{s} \sum_{j=1}^{s} X_j \right] - \frac{1}{s} \sum_{j=1}^{s} X_j \right| \leq \sqrt{\frac{2\mathbf{V}_s(\mathbf{X}) \ln 4/\eta}{s}} + \frac{7(b-a)\ln(4/\eta)}{3(s-1)}.$$

where

$$\mathbf{V}_s(\mathbf{X}) = \frac{1}{s(s-1)} \sum_{1 \leq k < \ell \leq s} (X_k - X_\ell)^2$$

2.3.2 Concentration of a Family of Functions

In this setting we are given a family of functions $\mathcal{F}, |\mathcal{F}| > 1$, and we will be interested in characterizing how $\bar{X}_f, f \in \mathcal{F}$ concentrates, in particular in this setting we will be concerned in bounding:

$$\mathbb{P}[\sup_{f \in \mathcal{F}} |\bar{X}_f - \mu_f| \leq \theta].$$

To achieve the above we can use concentration inequalities for a single function and combine them with a *union bound*. Taking for example the bound of Theorem 2.1, we can extend it to the setting above through a union bound as follows:

Corollary 2.1 ([Maurer and Pontil, 2009]). *Let $x_1, \dots, x_s$ be i.i.d. random variables with values in $\mathcal{X}$ and let $\eta > 0$, let $\mathcal{F}$ be a family of functions such that $\bar{X}_f \in [a, b] \forall f \in \mathcal{F}$. Then with probability at least $1 - \eta$ in the vector $\mathbf{X} = (x_1, \dots, x_s)$ for all functions $f \in \mathcal{F}$ we have:*

$$\left| \bar{X}_f - \mu_f \right| \leq \sqrt{\frac{2\mathbf{V}_{s,f}(\mathbf{X}) \ln 4|\mathcal{F}|/\eta}{s}} + \frac{7(b-a)\ln(4|\mathcal{F}|/\eta)}{3(s-1)}.$$

where

$$\mathbf{V}_{s,f}(\mathbf{X}) = \frac{1}{s(s-1)} \sum_{1 \leq k < \ell \leq s} (X_{k,f} - X_{\ell,f})^2$$

and $X_{j,f}$ is the estimator in Equation (2.1) applied to the j-th sample x_j, $f \in \mathcal{F}$.

Table 2.1: Notation table.

Symbol	Description
$T = (V, E)$	Temporal network
n, m	Number of nodes and temporal edges of T
G_T	Undirected projected static network of T
$M = (\mathcal{K}, \sigma)$	k-node ℓ-edge temporal motif.
$\mathcal{K}$	Multigraph associated to the motif M
σ	Ordering of the edges of the multigraph $\mathcal{K}$
δ	Duration limit of δ-instances or bound on δ-restless walks
$G_u[M]$	Undirected graph associated to $\mathcal{K}$
$\mathcal{U}(T, M, \delta)$	Set of δ-instances of M from T
C_M	Number of δ-instances of M in T
H	Static undirected simple graph (i.e., target template)
V_H, E_H	Set of nodes and edges of the target H
$C_M(e)$	Number of δ-instances of M containing $e \in G_T$
$p_e, p(e)$	Probability of sampling edge $e \in G_T$ in ODEN
C'_M	Estimate of C_M
ε, η	Quality and confidence parameters
$b(v), v \in V$	Temporal betweenness of node v in T
P_{v_1,v_2}	Temporal path connecting v_1 to v_2
$\sigma^{\dagger}_{v_1,v_2}$	Number of shortest temporal paths ($\dagger = sh$) or shortest δ-restless temporal walks ($\dagger = srtw$) connecting v_1 to v_2
$\sigma^{\dagger}_{v_1,v_2}(v)$	Number of shortest temporal paths ($\dagger = sh$) or shortest δ-restless temporal walks ($\dagger = srtw$) connecting v_1 to v_2 using v on at least one edge.

Note that this may provide a weak guarantee in some applications, since we are not fully exploiting the structure of the family of functions $\mathcal{F}$ [Boucheron et al., 2013]. With these tools we are able to provide probabilistic guarantees on the estimates provided by our algorithms for several estimation problems where $|\mathcal{F}| > 1$, such as we discuss in Chapter 4 and 5.

2.4 Notation

Finally, to conclude this chapter we summarize part of the notation we will be using in this work in Table 2.1.

Chapter 3

Algorithms for Exact and Approximate Counting of a Temporal Motif

In this chapter we extensively discuss the problem of obtaining a count of a temporal motif in a large temporal network. We present both exact and approximate algorithms, and evaluate them in practice on real world networks showing they significantly improve over state-of-the-art algorithms.

3.1 Introduction

The identification of patterns is a ubiquitous problem in data mining [Han et al., 2011] and is extremely important for networked data, where the identification of small, connected subgraphs, usually called *network motifs* [Milo, 2002] or *graphlets* [Pržulj et al., 2004], have been used to study and characterize networks from various domains, including biology [Mangan and Alon, 2003], neuroscience [Battiston et al., 2017], social networks [Ugander et al., 2013], and the study of complex systems in general [Milo, 2004]. Network motifs have been used as building blocks for various tasks in the analyses of networks across such domains, including anomaly detection [Sun et al., 2007] and clustering [Benson et al., 2016].

A fundamental problem in the analysis of network motifs is the counting problem [Bressan et al., 2017, Ahmed et al., 2014], which requires to output the number of instances of the given topology defining the motif. This challenging computational problem has been extensively studied, with several techniques designed to count the number of occurrences of simple motifs, such as triangles [Tsourakakis et al., 2009, Park et al., 2014, Stefani et al., 2017] or sparse motifs [De Stefani et al., 2017].

Most recent work has focused on providing techniques for the analysis of *large* networks, which have become the standard in most applications. However, in addition to a significant increase in size, modern networks also feature a richer structure, in terms of the type of information that is available for their vertices and edges [Ceccarello et al., 2017]. A type of information

that has drawn significant attention in recent years is provided by the temporal dimension [Holme and Saramäki, 2012, 2019]. In several applications edges are supplemented with *timestamps* describing the time at which an event, modeled by an edge, occurred: for example, in the analysis of spreading processes in epidemics, nodes are individuals, an edge represents a physical interaction between two individuals, and the timestamp represents the time at which the interaction was recorded [Peixoto and Gauvin, 2018].

When studying motifs in temporal networks, one is usually interested in occurrences of a given topology whose edge timestamps all appear in a small time span [Holme and Saramäki, 2012, Paranjape et al., 2017]. Discarding the temporal information of the network, i.e. ignoring the timestamps, may lead to incorrect characterization of the network of interest, while the analysis of temporal networks can provide insights that are not revealed when the temporal information is not accounted for [Holme and Saramäki, 2012]. For example, in a temporal network, a triangle $x \to y \to z \to x$ represents some feedback process on the information originated from x only if the edges occur at increasing timestamps (and the triangle occurs in a small amount of time). This information is revealed only by considering the timestamps, while by restricting to the static network (i.e., discarding edge timestamps) we may, often incorrectly, conclude that initial information starting from x always affects such sequence of events. Motifs that capture temporal interactions, such as the ones we consider, can provide more useful information than static motifs, as shown in several applications, including network classification [Tu et al., 2018] in the identification of mixing services from bitcoin networks [Wu et al., 2020], and in the analysis of travel patterns in road networks [Lei et al., 2020]. Furthermore, while on static networks motifs with high counts are associated with important properties of the dataset (e.g., its domain), temporal motifs provide additional insights of the networks they belong to, for example, they capture the presence of bursty or periodic activities [Belth et al., 2020]. Unfortunately, current algorithms do not enable the analysis of *arbitrary* temporal networks since they are tailored to specific classes of such motifs such as triangles or 2-node motifs with at most three edges [Paranjape et al., 2017, Pashanasangi and Seshadhri, 2021, Gao et al., 2022] or become impractical for even moderately-sized networks [Mackey et al., 2018], preventing therefore the analysis of many complex systems that can be modelled as a temporal network.

The problem of counting arbitrary motifs in temporal networks is, in fact, even more challenging than its counterpart for static networks, since there are motifs for which the problem is NP-hard for temporal networks while it is efficiently solvable for static networks [Liu et al., 2019]. Current approaches to count motifs in temporal networks are either *exact* [Paranjape et al., 2017, Mackey et al., 2018], and cannot be employed for very large networks, or *approximate* [Liu et al., 2019, Wang et al., 2020], but provide only rather weak guarantees on the quality of the estimates they return. In addition, even approximate approaches do not scale on large networks on arbitrary motifs, since some of them are limited to motifs of really small (up

to 4) temporal edges or they require impractical computational resources to compute a solution [Liu et al., 2019, Wang et al., 2020].

In this work we focus on the problem of counting arbitrary temporal motifs in temporal networks. Our goal is to obtain *practical* and *rigorous* algorithms to count arbitrary temporal motifs in large temporal networks. Our first focus will be the development of an exact algorithm that can be instantiated with any state-of-the-art algorithm for exact enumeration (e.g., [Mackey et al., 2018, Talati et al., 2022]). Our second focus will be to obtain an efficiently computable estimate of the count of a temporal motif having an arbitrary topology, while providing rigorous guarantees on the quality of the result. This work is an extension of our previous conference work [Sarpe and Vandin, 2021a], we next present our contributions and highlight the novel results that we added to the present work.

Our Contributions

This work provides the following contributions.

1. We develop a new exact algorithm (2δ-patch), embarrassingly parallelizable, that is used to count arbitrary temporal motifs on billion edges networks. This algorithm is based on a coverage of the timeline of events of the temporal network, such coverage is then used to carefully combine counts on the temporal subnetworks defined by each element in the coverage to obtain the exact count of the temporal motif in the input temporal network. In addition, it can be combined with any state-of-the-art algorithm for exact enumeration of temporal motif counts, becoming an extremely versatile algorithm.

2. We propose a novel algorithm to improve the exact backtracking algorithm by Mackey et al. [2018] (denoted with BT), for the enumeration of arbitrary temporal motifs in a temporal network. Our proposed algorithm builds over the existing algorithm by Mackey et al. [2018], devising a new procedure to match temporal edges when exploring the space of all possible matches of a temporal motif in a temporal network. Our algorithm is able to speed-up significantly the execution in [Mackey et al., 2018] thanks to the fact the it maintains a connected candidate match when matching a temporal motif, and pruning the search space combing ideas from existing literature [Wang et al., 2020].

3. We present PRESTO, an algorithm to approximate the count of motifs in temporal networks, which provides rigorous (probabilistic) guarantees on the quality of the output. We present two variants of PRESTO, both based on a common approach that counts motifs within randomly sampled small temporal windows. Both variants allow to analyze billion edges datasets providing sharp estimates. PRESTO features several useful properties, including: i) it has only one easy to interpret parameter, c, defining the length of the temporal windows for

the samples; ii) it can approximate the count of *any* motif topology; iii) it is easily parallelizable, with an almost linear speed-up with the available processors in most cases.

4. We provide tight and efficiently computable bounds to the number of samples required by our algorithms to achieve (multiplicative) approximation error $\leq \varepsilon$ with probability $\geq 1 - \eta$, for given $\varepsilon > 0$ and $\eta \in (0, 1)$. Our bounds are obtained through the application of advanced concentration results (i.e., Bennett's inequality) for the sum of independent random variables.

5. We show empirically, on large scale networks, the superiority of the 2δ-patch when combined with the backtracking algorithm (BT) by Mackey et al. [2018], and our new algorithm as a matching criteria with respect to using BT directly. In particular our proposed algorithm can save orders of magnitude of memory, and compute the count of arbitrary temporal motifs (even those for which BT fails). We then show how our 2δ-patch performs in parallel environments, rendering practical the *exact* computation of temporal motif counts.

6. We perform an extensive experimental evaluation on real datasets comparing approximate algorithms, including a dataset with more than 2.3 billion edges, never examined before. The results show that on large datasets our algorithm PRESTO significantly improves over the state-of-the-art sampling algorithms in terms of quality of the estimates while requiring a small amount of memory. In addition, we also discuss a simple parallel implementation of PRESTO that achieves almost linear speed-ups in most of the configurations.

In addition to our previous contribution [Sarpe and Vandin, 2021a], we made the following additional contributions: i) we developed the 2δ-patch algorithm (in Section 3.4.1) and ii) we developed an algorithm to improve the matching order of BT [Mackey et al., 2018] (in Section 3.4.2), iii) we add to the work the missing proofs from [Sarpe and Vandin, 2021a], iv) we perform a new experimental evaluation for the 2δ-patch algorithm (see Section 3.6.2) and v) we perform a new extensive experimental evaluation on real networks for all the sampling algorithms we compare, showing PRESTO's superiority, on large motifs never tested before (see Section 3.6.3).

3.2 Preliminaries

In this section we introduce the basic definitions used throughout this chapter. We start by recalling the definition of temporal networks.

Definition 3.1. *A* temporal network *is a pair* $T = (V, E)$ *where,* $V = \{v_1, \ldots, v_n\}$ *and* $E = \{(x, y, t) : x, y \in V, x \neq y, t \in \mathbb{R}^+\}$ *with* $|V| = n$ *and* $|E| = m$.

Given $(x, y, t) \in E$, we say that t is the *timestamp* of the edge (x, y). For simplicity in our presentation we assume the timestamps to be unique, which is without loss of generality since in practice our algorithms also handle non-unique timestamps. We also assume the edges to be sorted by increasing timestamps, that is $t_1 < \cdots < t_m$. Given an interval or *window* $[t_B, t_E] \subseteq \mathbb{R}$ we will denote $|t_E - t_B|$ as its *length*. Finally, given a temporal network $T = (V, E)$ and an interval $[t_a, t_b] \subseteq \mathbb{R}^+$ we denote with $T([t_a, t_b])$ the temporal subnetwork induced by the set of edges $E([t_a, t_b]) = \{e = (x, y, t) \in E : t \in [t_a, t_b]\}$, i.e., $T([t_a, t_b]) = (\{x : \exists (x, y, t) \vee (y, x, t) \in E([t_a, t_b])\}, E([t_a, t_b]))$.

We are interested in *temporal motifs*[1], which are small, connected subgraphs whose edge timestamps satisfy some specific constraints on their order of appearance. In particular, we consider the following definition introduced by Paranjape et al. [2017].

Definition 3.2. *A* k-node ℓ-edge temporal motif M *is a pair* $M = (\mathcal{K}, \sigma)$ *where* $\mathcal{K} = (V_\mathcal{K}, E_\mathcal{K})$ *is a directed and weakly connected* multigraph *where* $V_\mathcal{K} = \{v_1, \ldots, v_k\}$, $E_\mathcal{K} = \{(x, y) : x, y \in V_\mathcal{K}, x \neq y\}$ *s.t.* $|V_\mathcal{K}| = k$ *and* $|E_\mathcal{K}| = \ell$ *and* σ *is an ordering of* $E_\mathcal{K}$.

Note that a k-node ℓ-edge temporal motif $M = (\mathcal{K}, \sigma)$ is also identified by the sequence $\langle (x_1, y_1), \ldots (x_\ell, y_\ell) \rangle$ of ℓ edges ordered according to σ. Given a k-node ℓ-edge temporal motif M, k and ℓ are determined by $V_\mathcal{K}$ and $E_\mathcal{K}$. We will therefore use the term *temporal motif*, or simply *motif*, when k and ℓ are clear from context.

Given a temporal motif M, we are interested in counting how many times it appears within a time *duration* of δ, as captured by the following definition.

Definition 3.3. *Given a temporal network* $T = (V, E)$ *and* $\delta \in \mathbb{R}^+$, *a* time ordered *sequence* $S = \langle (x'_1, y'_1, t'_1), \ldots, (x'_\ell, y'_\ell, t'_\ell) \rangle$ *of* ℓ *unique temporal edges from* T *is a* δ-instance *of the temporal motif* $M = \langle (x_1, y_1), \ldots, (x_\ell, y_\ell) \rangle$ *if:*

1. *there exists a bijection* f *on the vertices such that* $f(x'_i) = x_i$ *and* $f(y'_i) = y_i$, $i = 1, \ldots, \ell$; *and*
2. *the edges of* S *occur within* δ *time, i.e.,* $t'_\ell - t'_1 \leq \delta$.

Note that in a δ-instance of the temporal motif $M = (\mathcal{K}, \sigma)$ the edge timestamps must be sorted according to the ordering σ. See Figure 3.1 for an example. Note also that Definition 3.3 requires a strict ordering of the timing of the events in the instance, such definition can be relaxed by requiring σ to provide a *non-strict* or even a *partial* ordering of the various events in the sequences, allowing therefore to account for instances where edges have the same timestamps. The techniques we will present can be adapted to work also under such definitions.

[1] In static networks, the term *graphlet* [Yaveroğlu et al., 2014] is sometimes used, with *motifs* denoting statistically significant graphlets. We use the term motif in accordance with previous works on temporal networks, e.g., [Paranjape et al., 2017, Liu et al., 2019, Wang et al., 2020].

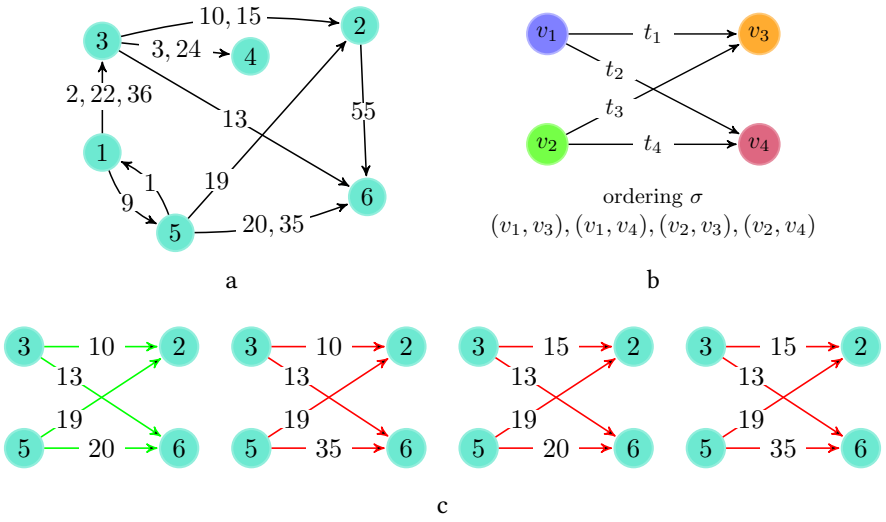


Figure 3.1: (3.1a): representation of a temporal network T with $n = 6$ nodes and $m = 13$ edges. (3.1b): a temporal motif, known as Bi-Fan [Liu et al., 2019]. (3.1c): sequences of edges of T that map topologically on the Bi-Fan motif, i.e., in terms of static (sub)graph isomorphism. For $\delta = 10$ only the green sequence is a δ-instance of the Bi-Fan motif, since the timestamps respect σ and $t'_\ell - t'_1 = 20 - 10 \leq \delta$. The red sequences are not δ-instances, since they do not respect such constraint or do not respect the order of σ.

Let $\mathcal{U}(T, M, \delta) = \{u : u \text{ is a } \delta\text{-instance of } M \text{ in } T\}$ be the *set of (all) δ-instances* of the motif M in T, denoted only with $\mathcal{U}$ when T, M and δ are clear from the context. Given a δ-instance $u \in \mathcal{U}(T, M, \delta)$, we denote the timestamps of its first and last edge with t^u_1 and t^u_ℓ, respectively. The *count* of M is $C_M(\delta) = |\mathcal{U}(T, M, \delta)|$, denoted with C_M when δ is clear from the context. We are interested in solving the following problem.

Problem 3.1 (Motif Counting Problem)**.** *Given a temporal network T, a temporal motif $M = (\mathcal{K}, \sigma)$, and $\delta \in \mathbb{R}^+$, compute the count $C_M(\delta)$ of the δ-instances of the motif M in the temporal network T.*

Solving the motif counting problem *exactly* may be infeasible for large networks, since even determining whether a temporal network contains a simple *star motif* is NP-hard [Liu et al., 2019]. State-of-the-art exact techniques [Mackey et al., 2018, Paranjape et al., 2017] require exponential time and memory in the number of edges of the temporal network, which renders them impractical for large temporal networks. Therefore, we are also interested in obtaining efficiently computable approximations of motif counts, as follows.

Problem 3.2 (Motif Approximation Problem)**.** *Given a temporal network T, a temporal motif $M = (\mathcal{K}, \sigma)$, $\delta \in \mathbb{R}^+$, $\varepsilon \in \mathbb{R}^+_0, \eta \in (0, 1)$ compute C'_M such that $\mathbb{P}[|C'_M - C_M(\delta)| \geq \varepsilon C_M(\delta)] \leq \eta$, i.e., C'_M is a* relative ε-approximation *to $C_M(\delta)$ with probability at least $1 - \eta$.*

We call an algorithm that provides such guarantees an (ε, η)-*approximation algorithm.*

3.3 Related Work

Various definitions of temporal networks and motifs have been proposed in the literature; we refer the interested reader to [Holme and Saramäki, 2012, 2019, Liu et al., 2021, Hanauer et al., 2021, Longa et al., 2021]. Here we focus on those works that adopted the same definitions used in this work. We organized this section distinguishing between works addressing Problem 3.1 and Problem 3.2.

Algorithms for Exact Counting

The definition of temporal motif we adopt was first proposed by Paranjape et al. [2017], which provided efficient exact algorithms to solve the motif counting problem for specific motifs. Boekhout et al. [2019] proposed an extension of such algorithms to some other motifs, in the context of multilayer temporal motifs, where the edges of the network are in the form (x, y, t, h) with h denoting the layer of the edge. Such algorithms are efficient only for specific motif topologies and do not scale to very large datasets. Gao et al. [2022] developed exact counting algorithms tailored to 2, 3-node and 3-edge temporal motifs by devising more efficient algorithms improving [Paranjape et al., 2017]. An algorithm for the motif counting problem on general motifs has been introduced by Mackey et al. [2018] which we denote as BT. Their algorithm is the first exact technique allowing the user to enumerate all δ-instances $u \in \mathcal{U}$ without any constraint on the motif's topology. The major back-draws of such algorithm are: i) that it may be impractical even for moderately-sized networks, due to its exponential time complexity and memory requirements ii) as discussed in [Wang et al., 2020] its matching order is not suited for arbitrary motifs since it is fixed and not dynamic according to the motif being matched. Recently Talati et al. [2022] devised specific hardware and programming framework to improve the BT algorithm [Mackey et al., 2018] showing significant speed-ups. Pashanasangi and Seshadhri [2021] proposed specialized algorithms for computing exactly the counts of temporal triangles with 3 edges, leveraging the concept of graph degeneracy. While such algorithms are proved to be very efficient in practice their approach is, to the best of our knowledge, only limited to triangles with 3 temporal edges therefore not applicable for general motifs, that is the problem addressed in this work.

Algorithms for Approximate Counting

Liu et al. [2019] proposed the first sampling algorithm for the motif approximation problem. The main strategy of Liu et al. [2019] is to partition the time interval containing all the edges of the network into *non overlapping*

and contiguous windows of length $c\delta$ (i.e., a grid-like partition), for some $c > 1$. The partition is then randomly shifted (i.e., the starting point of the first window may not coincide with the smallest timestamp of the network). The edges in each partition constitute the candidate samples to be analyzed using an exact algorithm. An *importance sampling* scheme is used to sample (approximately) r windows among the candidates, with a window being selected with probability proportional to the fraction of edges it contains. The estimate for each sampled window is obtained by weighting each δ-instance in the window, and the estimates are averaged across windows. This procedure is repeated b times to reduce the variance of the estimate. While interesting, this partition-based approach prevents such algorithm to provide (ε, η)-guarantees (see Section 3.5.1). Additionally, in practice this approach tends to sample subnetworks of T with a higher number of edges, and this works well only there is a strong positive correlation between "dense" temporal subnetworks and motif instances, this is not always the case for arbitrary motifs M on most of the temporal networks.

Recently Wang et al. [2020] proposed an (ε, η)-approximation algorithm for the motif counting problem. Their approach selects each edge in T with a user-provided probability p. Then for each selected edge $e = (x, y, t)$, the algorithm collects the edges with timestamps in the edge-centered window $[t-\delta, t+\delta]$, of length 2δ, computes on these edges all the δ-instances $u \in \mathcal{U}$ containing e, weights the instances, and combines the weights to obtain the final estimate. From the theoretical point of view, the main drawback of this approach is that in order to achieve the desired guarantees one has to set $p \geq 1/(1 + \eta\varepsilon^2)$, resulting in high values of p (i.e., almost all edges are selected) for reasonable values of η and ε (e.g., $p > 0.97$ for $\eta = 0.1$ and $\varepsilon = 0.5$). In addition, such approach is impractical on very large datasets, mainly due its huge memory requirements, and does provide accurate estimates only for specific motifs with a small number of edges and limited diameter of the multigraph $\mathcal{K}$ associated to M (see Section 3.6.3 and Section 3.6.4). Recently Sarpe and Vandin [2021b] developed ODEN, an efficient approximation algorithm to compute the estimates of *all* the temporal motifs that share a fixed common topology, e.g., a triangle, with a fixed number of temporal edges. We highlight that such problem is very different from the one we address in this work, since we aim at computing the estimates of *only one* fixed temporal motif. Finally, Porter et al. [2022] devised an algorithm based on a stochastic block model named TASBM (Temporal Activity Stochastic Block Model) to model occurrences of temporal motifs in temporal networks, the authors are able to analytically compute the expectation of temporal motif instances in their proposed model. While this approach is very promising for synthetic network generation, we enforce that it has no guarantees in terms of approximation if applied on real-world networks, therefore being far from applicable in our scenario where the algorithms are required to provide tight (probabilistic) guarantees on the approximation error.

3.4 Exact Computation of Motif Counts

In this section we develop algorithms for computing the exact count of a temporal motif. We first introduce a new algorithm based on a coverage of the timeline of events of the temporal network that can be used in combination with existing state-of-the-art approaches resulting in an efficient, scalable and parallelizable algorithm (in Section 3.4.1). Then in Section 3.4.2 we devise a novel algorithm to identify the matching order to be used within the state-of-the-art algorithm (BT) for enumerating temporal motif instances, such matching order will be in fact embedded with BT in our new exact algorithm (we will show its advantages over BT in practice in Section 3.6.2).

3.4.1 A New Algorithm for Exact Counting

In this section we present "2δ-patch", an exact algorithm that can be used in combination with many *exact sequential* algorithms, to parallelize their execution when computing motif counts. This algorithm, coupled with the BT algorithm by Mackey et al. [2018] often leads to an improvement in the scalability and running time of BT, given that it enables the BT algorithm to work with data-structures built on small subnetworks, and in addition it enables the possibility to easily parallelize the work, as we also show empirically in Section 3.6.2.

Before describing the framework, we start by presenting the idea on which it is based. First, such framework partitions the time interval with an approach similar to the one adopted by Liu et al. [2019], while enriching such partition with additional windows, to avoid missing the count of (possibly many) δ-instances. The idea is to first partition the time interval $[t_1, t_m]$ by contiguous and non overlapping windows of length $c\delta, c > 1$ starting from t_1 to t_m. Note, by counting the δ-instances occurring within the temporal subnetworks defined by such partition one may miss an arbitrary number of δ-instances (as we will discuss more in details for the approach in [Liu et al., 2019] in Section 3.5.1). Interestingly, the missing instances are the ones and only that have a starting timestamp in a window and an ending timestamp in the neighbouring window of the partition already defined. Therefore, to account for such instances we also consider a patch of length 2δ centred at each time-point adjacent to two different windows. A schema of such coverage strategy is reported in Figure 3.2. Hence, we can obtained an alternative formulation for C_M as follows, let $\mathcal{P} = \{[t_1 + jc\delta, t_1 + (j+1)c\delta] : j = 0, \ldots, \lceil \frac{t_m - t_1}{c\delta} \rceil - 1\}$ and $\bar{\mathcal{P}} = \{[t_e - \delta, t_e + \delta] : \exists [t_b, t_e], [t_e, t_{b'}] \in \mathcal{P}\}$, be the sets of windows obtained as shown from Figure 3.2, and let $C_M(T([t_a, t_b]))$ be the count of the δ-instances in the temporal subnetwork of T existing the interval $[t_a, t_b]$ (i.e., $T([t_a, t_b])$) then C_M can be computed as captured by the following lemma.

Lemma 3.1. *Given a temporal network T, a k-node ℓ-edge temporal motif M, a duration parameter $\delta \in \mathbb{R}^+$ and $c > 1$. Let $\mathcal{P}, \bar{\mathcal{P}}$ be the sets of real-valued*

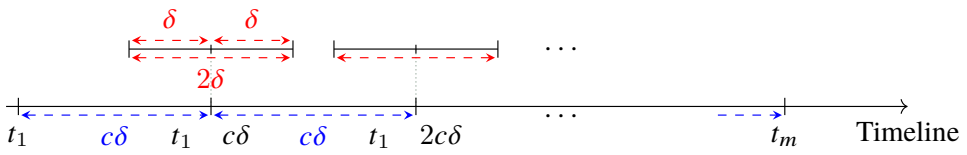


Figure 3.2: Coverage schema of the timeline of events of the temporal network T adopted in our Algorithm 1. The whole interval ($[t_1, t_m]$) is covered with contiguous non-overlapping windows of length $c\delta$ (the windows marked in blue). Additionally, over each time-point adjacent to two such windows, we place a patch of length 2δ (marked with red) covering the rightmost and leftmost portions of each of the two adjacent windows above. The final coverage is the union of both the red and blue windows.

intervals as defined above then:

$$C_M = \sum_{[t_b,t_e]\in\mathcal{P}} C_M(T([t_b,t_e])) + \sum_{[t_b,t_e]\in\bar{\mathcal{P}}} \sum_{u\in\mathcal{U}(M,\delta,T[t_b,t_e])} \mathbb{1}\left[t_1^u < \frac{(t_e-t_b)}{2} \wedge t_\ell^u > \frac{(t_e-t_b)}{2}\right]. \tag{3.1}$$

Proof. Let us first assume $c \geq 2$. Let us consider $\mathcal{U}(M,T,\delta)$, i.e., the set of δ-instances of M in T. Let $\mathcal{P}, \bar{\mathcal{P}}$ be as in statement, note that each δ-instance $u \in \mathcal{U}(M,T,\delta)$ spans over the interval $I_u = [t_1^u, t_\ell^u]$, then only the following three cases can occur:

1. $\exists I_1 \in \mathcal{P}, \nexists I_2 \in \bar{\mathcal{P}}$ such that $I_u \subset I_1, I_u \subset I_2$;
2. $\exists I_1 \in \mathcal{P}, I_2 \in \bar{\mathcal{P}}$ such that $I_u \subseteq (I_1 \cap I_2)$;
3. $\nexists I_1 \in \mathcal{P}, \exists I_2 \in \bar{\mathcal{P}}$ such that $I_u \subset I_1, I_u \subset I_2$.

Therefore, instances of type (1) and (2) can be counted through $\sum_{[t_b,t_e]\in\mathcal{P}} C_M(T([t_b,t_e]))$, that is the first term of Equation (3.1). Then, to account for the missing instances (of type (3)) we should count over $\bar{\mathcal{P}}$ but we need to avoid overcounting instances of type (2). Assuming that for a δ-instance $u \in \mathcal{U}(M,T,\delta)$ case (2) holds, then only the following two subcases can occur:

- $\exists I_2 = [t_b,t_e] \in \bar{\mathcal{P}} \wedge I_1 = [t'_b, (t_b-t_e)/2] \in \mathcal{P}$ s.t. $t_\ell^u \leq (t_b-t_e)/2, I_u \subseteq (I_1 \cap I_2)$;
- $\exists I_2 = [t_b,t_e] \in \bar{\mathcal{P}} \wedge I_1 = [(t_b-t_e)/2, t'_e] \in \mathcal{P}$ s.t. $t_1^u \geq (t_b-t_e)/2, I_u \subseteq (I_1 \cap I_2)$.

Therefore when counting over $\bar{\mathcal{P}}$ we only need to ensure that such δ-instances are filtered, that can be done by computing the missing term by $\sum_{[t_b,t_e]\in\bar{\mathcal{P}}} \sum_{u\in\mathcal{U}(M,\delta,T[t_b,t_e])} \mathbb{1}\left[t_1^u < \frac{(t_e-t_b)}{2} \wedge t_\ell^u > \frac{(t_e-t_b)}{2}\right]$, that is the second term in Equation (3.1). Now let us consider the case where $c \in (1,2)$, all the

cases above still hold but additionally in case (2), fixed a δ-instance u there may exist up to two windows $I_1, I_1' \in \bar{\mathcal{P}} \wedge I_2 \in \mathcal{P}$ s.t. $I_u \subseteq (I_1 \cap I_1' \cap I_2)$ but this is still filtered in the second term in Equation (3.1) as it can be easily verified, concluding therefore the proof. □

That is, in Lemma 3.1 the first term is simply the sum over the subnetworks with timestamps in the windows of the initial cover of the time interval spanning $[t_1, t_m]$, while the second term accounts for the missing instances in the first sum (i.e., it sums over the patches of length 2δ avoiding any over-counting).

Algorithm Description

Such simple way of writing C_M, from Lemma 3.1, inspires our Algorithm 1, which performs the computation of C_M through the formula above. In particular we first obtain the cover of the timeline of events $[t_1, t_m]$ of the network T (i.e., $\Gamma = \mathcal{P} \cup \bar{\mathcal{P}}$) by calling the function GetTimelineCover (line 12), which computes the cover as described above and in Figure 3.2, observe that each interval has an additional value denoting if it is a patch or not, represented by the boolean value "patch". The algorithm then iterates through each window in Γ, i.e., the coverage of the timeline already computed (Line 2). Collects the temporal subnetwork corresponding to the current window of Γ and checks if such window corresponds to a patch or not (lines 3-5). If the interval is not from a patch then it applies the exact algorithm to count all δ-instances in the current subnetwork (line 6). Instead if the interval corresponds to a patch the algorithm only counts those instances with starting time point (t_1^u) in the previous window and ending point (t_l^u) in the subsequent window (line 10) of the partition Γ, as from Equation 3.1.

2-δ Patch: Time Complexity

The worst-case time complexity of such algorithm depends strictly on the algorithm used for enumerating δ-instances in the patches of length 2δ. In our implementation we used the BT algorithm by Mackey et al. [2018] that has a worst case complexity of $O(m\hat{\kappa}^{(\ell-1)})$ when executed on a temporal network with m edges, $\hat{\kappa}$ maximum number edges in a window of length δ and a motif with $\ell \geq 2$ temporal edges. Therefore the worst case complexity of Algorithm 1 is bounded by $O(\sum_{\gamma\in\Gamma} m_\gamma \hat{\kappa}_\gamma^{(\ell-1)})$ (here we use γ to denote the temporal network fined by the interval $\gamma \in \Gamma$) that is still $O(m\hat{\kappa}^{(\ell-1)})$ since $\sum_{\gamma\in\Gamma} m_\gamma = O(m)$. Therefore, the worst case complexity is the same as the worst case required by the execution of its subroutine on the whole temporal network, but as we will show 2δ-patch empirically almost always performs better, given the smaller data-structures it needs to create on small windows and given the parallelization that it enables.

We conclude highlighting that there are several useful advantages in the 2-δ patch algorithm. 1) It parallelizes any inherently sequential algorithm for exact enumeration. 2) Works on very small sized windows of length 2δ

Algorithm 1: 2-δ Patch

Input: Temporal Network $T = (V, E)$, Motif M, Motif Duration $\delta > 0$, $c > 1$.

Output: Count C_M.

$\Gamma \leftarrow$ GetTimelineCover(T, c, δ); $C_M \leftarrow 0$;
for $j \leftarrow 1$ to $|\Gamma|$ (in parallel) **do**
  $([t_b, t_e], \text{patch}) \leftarrow \Gamma_j$;
  $\bar{T} \leftarrow T([t_b, t_e])$;
  if !patch **then**
    $C_M \leftarrow C_M + |\mathcal{U}(\bar{T}, M, \delta)|$
  else
    $j' \leftarrow j - 1$; $j'' \leftarrow \max(j + 1, |\Gamma|)$;
    $([t_b^{j'}, t_e^{j'}], \cdot) \leftarrow \Gamma_{j'}$; $([t_b^{j''}, t_e^{j''}], \cdot) \leftarrow \Gamma_{j''}$;
    $C_M \leftarrow C_M + |\{u : u$ is δ-instance of M in $\bar{T} \wedge (t_1^u < t_e^{j'} \wedge t_\ell^u > t_b^{j''})\}|$;
return C_M;
Function GetTimelineCover(T, c, δ) **:**
  $t \leftarrow t_1$; $i \leftarrow 1$;
  while $t < t_m$ **do**
    $\Gamma_i \leftarrow ([t, t + c\delta], \text{False})$;
    if $t + c\delta < t_m$ **then**
      $\Gamma_{i+1} \leftarrow ([t + (c - 1)\delta, t + (c + 1)\delta], \text{True})$;
    $t \leftarrow t + c\delta$; $i \leftarrow i + 2$;
  return $\mathcal{TC}$

and $c\delta, c > 1$ (note that bigger windows require much more resources to be processed, given that they often correspond to bigger subnetworks). 3) Enables scalability, since often exact algorithms work on global data structure that are very costly to be built on large scale (e.g., adjacency matrices), while such structures become practical on small sized subnetworks. 4) It enables the usage of different algorithms to process the various subnetworks, in fact, only the patches require an algorithm for the *enumeration* of the δ-instances, the other windows can be processed with an algorithm for exact *counting*, which in general can be less time consuming [Paranjape et al., 2017].

3.4.2 A New Matching Ordering for BT

In this section we discuss an improvement that can be adopted within the BT algorithm by Mackey et al. [2018] to improve its running time on *arbitrary* temporal motifs.

In particular, as observed by Wang et al. [2020] there may be several inefficient steps in the original algorithm [Mackey et al., 2018]. The inefficien-

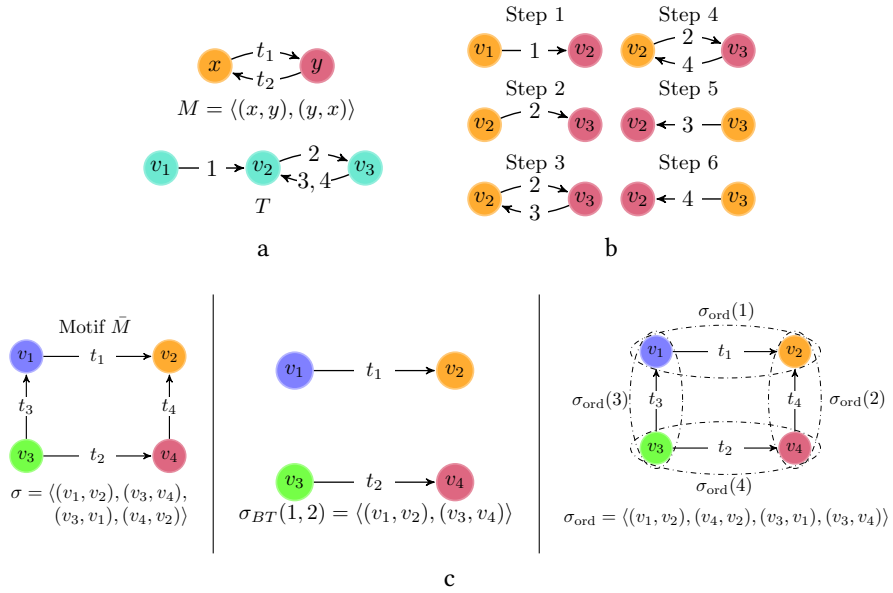


Figure 3.3: (3.3a): Temporal network T and temporal motif M. (3.3b): Steps of the matching algorithm BT applied on the motif M and network T from Figure (3.3a), and $\delta > 1$: the algorithm at each step maps one of the edges of T on the motif M following the ordering of the motif, colours are used to show how such mappings are made. When a match cannot be extended further, the algorithm proceeds to consider the next edge in the network T as candidate for the last matched edge of M, following therefore a backtracking procedure. (3.3c): Left: a motif M and its ordering σ. Centre: following the (time-first) matching order of BT, when matching the second edge of $\bar{M}$ ($\sigma_{\text{BT}}(1, 2)$ denotes the first two edges in the matching order adopted by the BT algorithm) the algorithm searches for disconnected pairs of edges. Right: Matching order (σ_{ord}) of the motif $\bar{M}$ computed by our Algorithm 2, the motif is now matched avoiding the generation of partial matches that are not connected.

cies arise form the matching order adopted in the BT algorithm, in fact such algorithm adopts a time-first approach to match the temporal edges of the temporal motif (see Figure (3.3a-3.3b)). Such matching order maps the edges of the motif to the ones of the temporal network following the ordering σ of the motif. While being fine for some motifs this approach is not suitable in general for *all* temporal motifs. In fact, the algorithm is very inefficient when such matches are, for example, not connected at a certain step of the algorithm (see Figure (3.3c) left and centre). The algorithm becomes inefficient under such conditions since it generates a lot of partial matches. Wang et al. [2020] proposed two different matching orders, respectively a) enforcing connectivity and b) enforcing the matching of the edges more advanced in time first (to prune the search space), as heuristics for the matching order that the BT algorithm should adopt during its matching phase. Despite such interesting contribution, the authors then adopted a schema limited to $\ell = 4$

Algorithm 2: Fixing the matching order for the BT algorithm

Input: Motif $M = \langle e_1 = (x_1, y_1), \ldots, e_\ell = (x_\ell, y_\ell)\rangle$.
Output: Matching order σ_{ord}.

1 $\sigma_{\text{ord}} \leftarrow \langle e_1 \rangle$;
2 $V_{\text{matched}} \leftarrow \{x_1, y_1\}$; $\sigma_{\text{matched}} \leftarrow \{e_1\}$;
3 **for** $j \leftarrow 1$ to $\ell - 1$ **do**
4 $\quad r \leftarrow \max\{i \in [2, \ell] : e_i \notin \sigma_{\text{matched}}, e_i \in \mathcal{N}(V_{\text{matched}})\}$;
5 $\quad \sigma_{\text{ord}} \leftarrow \sigma_{\text{ord}}.\text{append}(e_r)$;
6 $\quad V_{\text{matched}} \leftarrow V_{\text{matched}} \cup \{x_r, y_r\}$; $\sigma_{\text{matched}} \leftarrow \sigma_{\text{matched}} \cup \{e_1\}$;
7 **return** σ_{ord};

edges, for matching the motifs in their algorithms, given the complexity of adapting the BT algorithm to work for *general* matching orders.

Building on such contribution we devised an algorithm that *dynamically* finds the matching order for a general motif combining the two ideas from [Wang et al., 2020], and adapted the BT algorithm to work for a general matching order, as provided by our algorithm. In Algorithm 2, we show how, given a motif M, we find the matching order to be used inside the BT algorithm. We start by first ordering the first edge of the motif M as first in the order (σ_{ord}) to be reported in output (line 1), this is arbitrary and is done to reduce the complexity of implementation inside the BT algorithm. Then in line 2 we update the data structures V_{matched} keeping the set of nodes of the motif M adjacent to an edge already accounted in σ_{ord}, and σ_{matched} a set keeping the edges from the motif M already fixed (in σ_{ord}). We then start a loop fixing an edge of the motif at each iteration, in particular we select such edge in line 4. To fix an edge we select the edge that is the one with highest order in M that respects two constraints, i) it is not already matched and ii) it is connected to the already matched subgraph, i.e., such edge is in the set $\mathcal{N}(V_{\text{matched}})$ where $\mathcal{N}(V_{\text{matched}}) = \{e = (x, y) \in M : x \in V_{\text{matched}} \vee y \in V_{\text{matched}}\}$. Once found the edge to be matched, we updated the matching order (line 5) and update the data structures keeping track of the fixed edges (line 6). At the end of the loop we return the matching order σ_{ord} (line 7). In Figure (3.3c Right) we report the output of our Algorithm 2 on the motif $\bar{M}$, the reader may appreciate how such motif is now matched without the generation of disconnected matches. Finally note that the running time of such algorithm is bounded by $O(\ell)$, therefore negligible in all practical applications.

3.5 PRESTO: Approximating Temporal Motif Counts with Uniform Sampling

We now describe and analyze our algorithm PRESTO (apPRoximating tEmporal motifS counTs with unifOrm sampling) for the motif counting

problem. We start by describing, in Section 3.5.1, the common strategy underlying our algorithms. We then briefly highlight the differences between PRESTO and the existing sampling algorithms for the counting problem. In Section 3.5.2 and Section 3.5.3 we present and analyze two variants of the common strategy introduced in Section 3.5.1. We conclude with the analysis of PRESTO's complexity in Section 3.5.4.

3.5.1 PRESTO: General Approach

The general strategy of PRESTO is presented in Algorithm 3. Given a network T, PRESTO collects s samples, where each sample is obtained by gathering all edges $e \in E$ with timestamp in a *small random* window $[t_r, t_r + c\delta]$ of length $c\delta$, using `SampleWindowStart(`E`)` (Lines 2-3) to select t_r (i.e., the starting time point of the window) through a uniform random sampling approach. For each sample, an exact algorithm is then used to extract all the δ-instances of motif M in such sample (Line 4). The weight of each δ-instance extracted is computed with the call `ComputeWeight(`u`)` (Line 6), and the sum of all such weights constitutes the estimate provided by the sample (Line 7). The weights account for the probability of sampling the δ-instances in the sample, making the final estimate *unbiased*. The final estimate produced in output is the average of the samples' estimates (Lines 8-9). Note that the `for` cycle of Line 1 is trivially parallelizable. The two variants of PRESTO we will present differ in the way i) `SampleWindowStart(`E`)` and ii) `ComputeWeight(`u`)` are defined. Finally note that we can frame the estimates obtained by PRESTO in the framework introduced in Section 2.2. In particular $\mathcal{X}$ is identified with the set of temporal subnetworks of T spanning intervals of length $c\delta$, $|\mathcal{F}| = 1$ since we are interested in obtaining only the count of a motif M given a sampled temporal network. The distribution ρ is defined by the probability of sampling a random timestamp t_r in line 2, and finally $f(x)$ is the aggregation of the weights of each δ-instance identified in a sampled temporal network that will account for ρ. Hence $\hat{X}_f(x) = X_i, i = 1, \ldots, s$.

Differently from Liu et al. [2019], our algorithm PRESTO does not partition the edges of T in non overlapping windows, and relies instead on *uniform sampling*. We recall (see Section 3.3, and the partition $\mathcal{P}$ in Section 3.4.1) that in Liu et al. [2019], after computing all the non overlapping intervals defining the candidate samples, there may be several δ-instances $u \in \mathcal{U}$ that cannot be sampled (i.e., all δ-instances having t_1^u in window j and t_ℓ^u in window $j + 1$). This significantly differs from PRESTO, which instead samples at each iteration a small random window from $[t_1, t_m]$ without restricting the candidate windows, allowing to sample *any* δ-instance $u \in \mathcal{U}(T, M, \delta)$ at each iteration. This enables us to provide stronger guarantees on the quality of the output, since each δ-instance has a non-zero probability of being sampled at each step, leading to (ε, η)-approximation guarantees. Additionally, as discussed previously the work of Liu et al. [2019] is based on *importance sampling* and it achieves satisfactory performances

Algorithm 3: PRESTO

Input: Temporal Network $T = (V, E)$, Motif M, Motif Duration $\delta > 0$, $s > 0$, $c > 1$

Output: Estimate C'_M of C_M

1 **for** $i \leftarrow 1$ to s **do**
2 $\quad t_r \leftarrow$ SampleWindowStart(E);
3 $\quad T_i \leftarrow T([t_r, t_r + c\delta])$; $X_i \leftarrow 0$;
4 $\quad \mathcal{S}_i \leftarrow \{u : u$ is δ-instance of M in $T_i\}$;
5 $\quad$ **foreach** $u \in \mathcal{S}_i$ **do**
6 $\quad\quad w(u) \leftarrow$ ComputeWeight(u);
7 $\quad\quad X_i \leftarrow X_i + w(u)$;
8 $C'_M \leftarrow \frac{1}{s} \sum_{i=1}^{s} X_i$;
9 **return** C'_M;

only when there is a strong positive correlation between motif instances and number of edges in a sample, that is not often the case in general.

Differently from the work of Wang et al. [2020] PRESTO samples temporal windows of length $c\delta$ and does not follow the edge-centric approach (i.e., sampling temporal edges with a user-provided probability p) of Wang et al. [2020]. In addition, the approach of [Wang et al., 2020] collects edges in temporal-windows of length 2δ (see Section 3.3), while in PRESTO the window size is controlled by the parameter c, which, when fixed to be < 2, leads to much more scalability than [Wang et al., 2020] while requiring less memory (see Section 3.6). We also enforce that the approach in Wang et al. [2020] is suitable for very *small* temporal motifs (in terms of $\ell \leq 4$ edges, and small diameter of the multigraph $\mathcal{K}$), given that it needs to identify all δ-instances containing the sampled edges for *all the possible orderings* of such edges inside the each instance, i.e., $1, \ldots, \ell$, and each such step requires up to exponential complexity in general for a fixed edge, therefore practical only for very small values of ℓ.

3.5.2 PRESTO-A: A First Sampling Approach

In this section we present and analyze PRESTO-A, our first (ε, η)-approximation algorithm obtained by specifying i) how the starting point t_r of the temporal window defining a sample T_i is chosen (function SampleWindowStart(E) in Line 2) and ii) how the weight $w(u)$ of a δ-instance u in a sample is computed (ComputeWeight(u), Line 6).

The starting point t_r of sample T_i is sampled uniformly at random in the interval $[t_\ell - c\delta, t_{m-\ell}] \subseteq \mathbb{R}$, where we recall $\ell = |E_{\mathcal{K}}|$ and $m = |E|$. Regarding the choice of the weight $w(u)$ for each instances $u \in \mathcal{S}_i$, PRESTO-A considers $w(u) = 1/p_u$, with p_u being the probability of u to be in $\mathcal{S}_i$, that is $p_u = r_u/\Delta_{T,1}$, where $\Delta_{T,1} = t_{m-\ell} - t_\ell + c\delta$ is the total length of the interval from which t_r is sampled (recall that we choose t_r from $[t_\ell - c\delta, t_{m-\ell}]$), and

$r_u = c\delta - (t^u_\ell - t^u_1)$ is the length of the interval in which t_r must be chosen for u to be in $\mathcal{S}_i$, $i = 1, \ldots, s$.

We now present the theoretical guarantees of PRESTO-A and give efficiently computable bounds for the sample size s needed for the (ε, η)-approximation to hold. Recall the definition of $\mathcal{U}(T, M, \delta)$, which is the set of δ-instances of M in T. Let u be an arbitrary δ-instance of motif M, and let T_i be an arbitrary sample obtained by PRESTO-A at its i-th iteration. We define the following set of indicator random variables, for $u \in \mathcal{U}$ and for $i = 1, \ldots, s$: $X^i_u = 1$ if $u \in \mathcal{S}_i$, 0 otherwise. Each variable $X^i_u, i = 1, \ldots, s, u \in \mathcal{U}$ is a Bernoulli random variable with $\mathbb{P}(X^i_u = 1) = \mathbb{P}(u \in \mathcal{S}_i) = \frac{r_u}{\Delta_{T,1}} = p_u$. Therefore, for each variable X^i_u, it holds $\mathbb{E}[X^i_u] = p_u$. Thus, for each $i = 1, \ldots, s$, iteration i provides an estimate X_i of C_M, which is the random variable: $X_i = \sum_{u \in \mathcal{U}} \frac{1}{p_u} X^i_u$. We therefore have the following result.

Lemma 3.2. *For each $i = 1, \ldots, s$, X_i and $C'_M = \frac{1}{s}\sum_{i=1}^{s} X_i$ are unbiased estimators for C_M, that is $\mathbb{E}[X_i] = \mathbb{E}[C'_M] = C_M$.*

Proof. Considering the linearity of expectation and the definition of the variables X_i and X^i_u, $i = 1, \ldots, s, u \in \mathcal{U}$, we have:

$$\mathbb{E}[X_i] = \sum_{u \in \mathcal{U}} \frac{1}{p_u} \mathbb{E}\left[X^i_u\right] = \sum_{u \in \mathcal{U}} \frac{1}{p_u} p_u = C_M.$$

Combining with the linearity of expectation to C'_M the statement follows. □

The following result provides a bound on the variance of the estimate of C_M provided by C'_M.

Lemma 3.3. *For* PRESTO-A *it holds*

$$\mathrm{Var}\left(C'_M\right) = \mathrm{Var}\left(\frac{1}{s}\sum_{i=1}^{s} X_i\right) \leq \frac{C^2_M}{s}\left(\frac{\Delta_{T,1}}{(c-1)\delta} - 1\right).$$

Proof. We start by observing that, $\mathrm{Var}\left(\frac{1}{s}\sum_{i=1}^{s} X_i\right) = \frac{1}{s^2}\sum_{i=1}^{s}\mathrm{Var}(X_i)$ by the mutual independence of the variables $X_1, \ldots, X_s$. We observe that $\mathrm{Var}(X_i) = \mathbb{E}[X^2_i] - \mathbb{E}[X_i]^2, i = 1, \ldots, s$ and we recall that $\mathbb{E}[X_i] = C_M$ by Lemma 3.2, thus we need to bound $\mathbb{E}[X^2_i]$,

$$\begin{aligned}\mathbb{E}[X^2_i] &= \sum_{u_1 \in \mathcal{U}}\sum_{u_2 \in \mathcal{U}} \frac{1}{p_{u_1} p_{u_2}} \mathbb{E}[X^i_{u_1} X^i_{u_2}] \\ &\overset{(1.)}{\leq} \sum_{u_1 \in \mathcal{U}}\sum_{u_2 \in \mathcal{U}} \frac{1}{p_{u_2}} \overset{(2.)}{\leq} \sum_{u_1 \in \mathcal{U}}\sum_{u_2 \in \mathcal{U}} \frac{\Delta_{T,1}}{(c-1)\delta} = \frac{C^2_M \Delta_{T,1}}{(c-1)\delta}\end{aligned} \tag{3.2}$$

where (1.) follows from $\mathbb{E}[XY] \leq \mathbb{E}[X]$ and (2.) from the definition of p_{u_2}. Based on the above, we can bound the variance of the variables

X_i, $i = 1, \ldots, s$, as follows, $\mathrm{Var}(X_i) = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 \leq C_M^2 \left(\frac{\Delta_{T,1}}{(c-1)\delta} - 1\right)$. Note that such bound does not depend on the index $i = 1, \ldots, s$, thus substituting in the original summation we obtain, $\mathrm{Var}\left(\frac{1}{s}\sum_{i=1}^{s} X_i\right) \leq \frac{C_M^2}{s}\left(\frac{\Delta_{T,1}}{(c-1)\delta} - 1\right)$. □

We now present a first efficiently computable bound on the number of samples s required to have that C'_M is a relative ε-approximation of C_M with probability $\geq 1 - \eta$. Such bound is based on the application of Hoeffding's inequality (see [Mitzenmacher and Upfal, 2017] and Theorem 3.5 for the statement) an advanced but commonly used technique in the analysis of probabilistic algorithms, the proof is found in Section 3.7.1.

Theorem 3.1. *Given $\varepsilon \in \mathbb{R}^+, \eta \in (0,1)$ let $X_1, \ldots, X_s$ be the random variables associated with the counts computed at iterations $1, \ldots, s$, respectively, of* PRESTO-A. *If $s \geq \frac{\Delta_{T,1}^2}{2(c-1)^2\delta^2\varepsilon^2} \ln\left(\frac{2}{\eta}\right)$, then it holds*

$$\mathbb{P}\left(\left|\frac{1}{s}\sum_{i=1}^{s} X_i - C_M\right| \geq \varepsilon C_M\right) \leq \eta$$

that is, PRESTO-A *is an (ε, η)-approximation algorithm.*

We now show that by using Bennett's inequality (see [Boucheron et al., 2013] and Theorem 3.2), a more advanced result on the concentration of the sum of independent random variables, we can derive a bound that is much tighter than the above, while still being efficiently computable.

Theorem 3.2 ([Bennett, 1962])**.** *For a collection $X_1, \ldots, X_s$ of independent random variables satisfying $X_i \leq M_i$, $\mathbb{E}[X_i] = \mu_i$ and $\mathbb{E}[(X_i - \mu_i)^2] = \sigma_i^2$ for $i = 1, \ldots, s$ and for any $t \geq 0$, the following holds*

$$\mathbb{P}\left(\left|\frac{1}{s}\sum_{i=1}^{s} X_i - \frac{1}{s}\sum_{i=1}^{s} \mu_i\right| \geq t\right) \leq 2\exp\left(-s\frac{v}{B^2}h\left(\frac{tB}{v}\right)\right)$$

where $h(x) = (1+x)\ln(1+x) - x$, $B = \max_i M_i - \mu_i$ and $v = \frac{1}{s}\sum_{i=1}^{s}\sigma_i^2$.

One of the difficulties in applying such result to our scenario is that we know only an upper bound to the variance of the random variables X_i, $i = 1, \ldots, s$. A discussion and proof of why Bennett's inequality can be applied to our context is in 3.7.1.1. We now state our main result.

Theorem 3.3. *Given $\varepsilon \in \mathbb{R}^+, \eta \in (0,1)$ let $X_1, \ldots, X_s$ be the random variables associated with the counts computed at iterations $1, \ldots, s$, respectively, of* PRESTO-A. *If $s \geq \left(\frac{\Delta_{T,1}}{(c-1)\delta} - 1\right)\frac{1}{(1+\varepsilon)\ln(1+\varepsilon)-\varepsilon}\ln\left(\frac{2}{\eta}\right)$, then it holds*

$$\mathbb{P}\left(\left|\frac{1}{s}\sum_{i=1}^{s} X_i - C_M\right| \geq \varepsilon C_M\right) \leq \eta$$

that is, PRESTO-A *is an (ε, η)-approximation algorithm.*

Proof. Let $Pr = \mathbb{P}\left(|\frac{1}{s}\sum_{i=1}^{s} X_i - C_M| \geq \varepsilon C_M\right)$, we want to show that for s as in the statement it holds $Pr \leq \eta$.

In order to apply Bennett's bound, note that $\frac{1}{s}\sum_{i=1}^{s}\mathbb{E}[X_i] = \frac{1}{s}\sum_{i=1}^{s}\mu_i = C_M$ by Lemma 3.2. Moreover $B = \max_i M_i - \mu_i = \frac{\Delta_{T,1}}{(c-1)\delta}C_M - C_M = C_M\left(\frac{\Delta_{T,1}}{(c-1)\delta} - 1\right)$ and in addition $\mathbb{E}[(X_i - \mu_i)^2] = \sigma_i^2 \leq C_M^2\left(\frac{\Delta_{T,1}}{(c-1)\delta} - 1\right) = \hat{\sigma}_i^2, i = 1, \ldots, s$ (see Equation (3.2) and Equation (3.3)), thus $v = \frac{1}{s}\sum_{i=1}^{s}\sigma_i^2 \leq \frac{1}{s}\sum_{i=1}^{s}\hat{\sigma}_i^2 = \frac{1}{s}\sum_{i=1}^{s} C_M^2\left(\frac{\Delta_{T,1}}{(c-1)\delta} - 1\right) = C_M^2\left(\frac{\Delta_{T,1}}{(c-1)\delta} - 1\right) = \hat{v}$. Let $t = \varepsilon C_M$, then it holds

$$\frac{tB}{\hat{v}} = \varepsilon \qquad \text{and} \qquad \frac{\hat{v}}{B^2} = \frac{1}{\left(\frac{\Delta_{T,1}}{(c-1)\delta} - 1\right)}$$

applying Theorem 3.2, with the upper bound $\hat{v}$ to v we obtain,

$$Pr \leq 2\exp\left(-s\frac{\hat{v}}{B^2}h\left(\frac{tB}{\hat{v}}\right)\right) \leq \eta$$

by substituting the quantities above and by the choice of s as in statement, which concludes the proof. □

Note that the bound in Theorem 3.3 is significantly better than the one in Theorem 3.1, since the former has a quadratic dependence from $\Delta_{T,1}$ while the latter enjoys a linear dependence from $\Delta_{T,1}$. Furthermore, differently from the bounds by Wang et al. [2020], our bounds depend on characteristic quantities of the datasets ($\Delta_{T,1}$) and of the algorithm's input (c, δ). Therefore we expect our bounds to be more informative, and also possibly tighter than the bounds of [Wang et al., 2020] (with the improvement due, at least in part, to the use of Bennett's inequality, while [Wang et al., 2020] leverages on Chebyshev's inequality, which usually provides looser bounds [Mitzenmacher and Upfal, 2017]).

A natural question is to understand how tight are the guarantees that we proved. Unfortunately, the main backdraw behind the application of Bennett's inequality is our, sometimes, loose bound on the range of the variables X_i and on their variance. We discuss in Section 3.7.1.2 how novel results in concentration theory (e.g., based on the *empirical* values of the variables, or considering higher moments of such variables), cannot easily be adapted to our scenario while maintaining the efficient computability of the bounds considered.

3.5.3 PRESTO-E: An Alternative Sampling Approach

In this section we present and analyse our second (ε, η) approximation algorithm, PRESTO-E, obtained with a different variant of the general strategy of Algorithm 3.

PRESTO-E selects the starting point t_r (Line 2 in Algorithm 3) of sample T_i only from the timestamps of edges of T. In particular, t_r is chosen uniformly at random in $\{t_1, \ldots, t_{last}\} \subseteq \{t_1, \ldots, t_m\}$, where $t_{last} = \min\{t : (x, y, t) \in E \wedge t \geq t_m - c\delta\}$. When the edges of T are far from each other or have a skewed distribution in time, or occur mostly in some well-spaced subsets $\{[t_a, t_b] \subseteq [t_1, t_m] : t_a \leq t_b\}$, we expect PRESTO-E to collect samples with more δ-instances than PRESTO-A (which samples t_r uniformly at random almost from the entire time interval $[t_1, t_m]$ but without restricting to existing edges). Similarly to PRESTO-A, the weight $w(u)$ (Line 6 of Algorithm 3) is computed by $w(u) = 1/\tilde{p}_u$, where $\tilde{p}_u$ is the probability that $u \in \mathcal{U}$ is in the set $\mathcal{S}_i$. Due to the (random) choice of t_r, $\tilde{p}_u = \tilde{r}_u/\Delta_{T,2}$, where $\Delta_{T,2} = |\{t : (x, y, t) \in E \wedge t \in [t_1, t_{last}]\}|$ is the number of possible choices for t_r (if $t_{last} = t_m$ then $\Delta_{T,2} = m$) and $\tilde{r}_u = |\{t : (x, y, t) \in E \wedge t \in [\max\{t_1, t^u_\ell - c\delta\}, \min\{t_{last}, t^u_1\}]\}|$ is the number of choices for t_r such that $u \in \mathcal{U}$ is in $\mathcal{S}_i$, $i = 1, \ldots, s$.

We then define the indicator random variables, for each $u \in \mathcal{U}(T, M, \delta)$ and for $i = 1, \ldots, s$: $\tilde{X}^i_u = 1$ if $u \in \mathcal{S}_i$, 0 otherwise. Then each variable $\tilde{X}^i_u$, $i = 1, \ldots, s$, $u \in \mathcal{U}$ is a Bernoulli random variable for which it holds $\mathbb{P}(\tilde{X}^i_u = 1) = \mathbb{P}(u \in \mathcal{S}_i) = \frac{\tilde{r}_u}{\Delta_{T,2}} = \tilde{p}_u$. Therefore the variable denoting the count of PRESTO-E at its iteration $i \in [1, s]$ can be written as $X_i = \sum_{u \in \mathcal{U}} 1/\tilde{p}_u \tilde{X}^i_u$.

Similarly to what done for PRESTO-A we show that PRESTO-E provides an unbiased estimate of C_M, with bounded variance (proofs are omitted since they can be trivially recovered from the ones for PRESTO-A).

Lemma 3.4. *$C'_M = \frac{1}{s}\sum_{i=1}^{s} X_i$ is an unbiased estimator for C_M, that is $\mathbb{E}[C'_M] = C_M$.*

Lemma 3.5.

$$\mathrm{Var}\left(C'_M\right) = \mathrm{Var}\left(\frac{1}{s}\sum_{i=1}^{s} X_i\right) \leq \frac{C_M^2}{s}\left(\Delta_{T,2} - 1\right)$$

So we can now derive the result on PRESTO-E's guarantees by the application of Bennett's inequality.

Theorem 3.4. *Given $\varepsilon \in \mathbb{R}^+, \eta \in (0, 1)$ let $X_1, \ldots, X_s$ be the random variables associated with the counts computed at iterations $1, \ldots, s$, respectively, of* PRESTO-E. *If $s \geq \frac{(\Delta_{T,2}-1)}{(1+\varepsilon)\ln(1+\varepsilon)-\varepsilon} \ln\left(\frac{2}{\eta}\right)$, then it holds*

$$\mathbb{P}\left(\left|\frac{1}{s}\sum_{i=1}^{s} X_i - C_M\right| \geq \varepsilon C_M\right) \leq \eta$$

that is, PRESTO-E *is an (ε, η)-approximation algorithm.*

As for PRESTO-A, by using Bennett's inequality we obtain a linear dependence between the sample size s and $\Delta_{T,2}$, while commonly used techniques (e.g., Hoeffding's inequality) lead to a quadratic dependence on such term.

3.5.4 PRESTO: Complexity Analysis

In this section we analyse the time complexity of PRESTO's versions introduced in the previous sections.

Note that our algorithms employ an exact enumerator as subroutine. For the sake of the analysis we consider the complexity when the subroutine is the algorithm by Mackey et al. [2018], which we used in our implementation. Let us first start with a definition, given a temporal network $T = (V, E)$, we denote with $\dim(T)$ the set $\{t_i | (x, y, t_i) \in E, i = 1, \ldots, m\}$. The algorithm in [Mackey et al., 2018] has a worst-case complexity $O(m\hat{\kappa}^{(\ell-1)})$ when executed on a motif with ℓ edges and a temporal network with m temporal edges, where $\hat{\kappa}$ is the maximum number of edges within a window of length δ, i.e., $\hat{\kappa} = \max\{|S(t)| : S(t) = \{(x, y, \bar{t}) \in E : \bar{t} \in [t, t + \delta]\}, t \in \dim(T)\}$. Note that such complexity is *exponential* in the number of edges ℓ of the temporal motif. Obviously our algorithms benefit from any improvement to the state-of-the-art exact algorithms.

Complexity of PRESTO-A. The worst-case complexity is $O(s\hat{m}\hat{\kappa}^{(\ell-1)} + sC^*_M)$ when executed sequentially, where $\hat{m}$ is the maximum number of edges in a window of length $c\delta$, i.e., $\hat{m} = \max\{|S(t)| : S(t) = \{(x, y, \bar{t}) \in E : \bar{t} \in [t, t + c\delta]\}, t \in \dim(T)\}$, and C^*_M is the maximum number of δ-instances contained in any window of length $c\delta$. This corresponds to the case where PRESTO-A collects samples with many edges for which many δ-instances occur. The complexity becomes $O(\frac{s}{\tau}\hat{m}\hat{\kappa}^{(\ell-1)} + \frac{s}{\tau}C^*_M)$ when PRESTO-A is executed in a parallel environment with τ threads (parallelizing the `for` cycle in Algorithm 3).

Complexity of PRESTO-E. Using the notation defined above, the worst-case complexity of a naive implementation of PRESTO-E is $O(s\hat{m}\hat{\kappa}^{(\ell-1)} + s\hat{m}C^*_M)$. As for PRESTO-A, the first term comes from the complexity of the exact algorithm for computing $\mathcal{S}_i, i = 1, \ldots, s$, whereas the second term is the worst-case complexity of computing the weights for each sample. The additional $O(\hat{m})$ complexity of the second term arises from the computation of $\tilde{r}_u$ for each $u \in \mathcal{S}_i, i = 1, \ldots, s$. The complexity of this computation can be reduced to $O(\log(m))$ by applying binary search to the edges of T. With such an approach, we obtained the final complexity $O(s\hat{m}\hat{\kappa}^{(\ell-1)} + s\log(m)C^*_M)$. The complexity reduces to $O(\frac{s}{\tau}\hat{m}\hat{\kappa}^{(\ell-1)} + \frac{s}{\tau}\log(m)C^*_M)$ when τ threads are available for a parallel execution.

3.6 Experimental Evaluation

In this section we present the results of our extensive experimental evaluation on large scale datasets. To the best of our knowledge we consider one of the largest dataset, in terms of number of temporal edges, ever used for the motif counting problem with more than 2.3 billion temporal edges. After describing the experimental setup and implementation (Section 3.6.1), we first compare the exact algorithm by Mackey et al. [2018] and our proposed 2δ-patch algorithm with respect to their running time and memory require-

Table 3.1: Datasets used in our experimental evaluation. We report: number of nodes n; number of edges m; *precision* of the timestamps; *timespan* of the network.

Name	n	m	Precision	Timespan
Stackoverflow (SO)	2.58M	47.9M	sec	2774 (days)
Bitcoin (BI)	48.1M	113M	sec	2585 (days)
Reddit (RE)	8.40M	636M	sec	3687 (days)
EquinixChicago (EC)	11.16M	2.32B	μ-sec	62.0 (mins)

ments (Section 3.6.2), and we also evaluate a parallel implementation of our 2δ-patch algorithm. We then proceed by comparing the quality of the estimated counts provided by PRESTO with the estimates from state-of-the-art sampling algorithms from Liu et al. [2019], and Wang et al. [2020] (Section 3.6.3). Then we compare the memory requirements of the approximate algorithms (Section 3.6.4), which may be a limiting factor for the analysis of very large datasets. Additional experiments on PRESTO's running time comparison with the exact algorithm, with the current state of-the art sampling algorithms, and on PRESTO's parallel implementation are in supplementary material (Section 3.7).

3.6.1 Experimental Setup and Implementation

The characteristics of the datasets we considered are in Table 3.1. EquinixChicago [noa] is a *bipartite* temporal network that we built. A detailed description is in Section 3.7.2. Descriptions of the other datasets are in [Paranjape et al., 2017, Liu et al., 2019].

We implemented our algorithms in C++20, using the parallel version (with 20 threads) of Algorithm 1 using [Mackey et al., 2018] as exact subroutine for extracting exact counts when comparing sampling algorithms. The experiments were performed on a 72 core Intel Xeon Gold 5520 @ 2.2GHz machine with 1008GB of RAM available, running Ubuntu 20.04. The code we developed, and the dataset we build are entirely available at https://github.com/VandinLab/PRESTO/, links to additional resources (datasets and other implementations) are in Section 3.7.3.1. We tested all the sampling algorithms on the motifs from Figure 3.4b. We considered motifs with at most $\ell = 4$ when evaluating Wang et al. [2020] since their algorithm does not allow for motifs with a higher ℓ in input (as we also discussed in Section 3.5.1). Since the EC dataset is a bipartite network, it can contain all but the motifs colored in blue with in Figure 3.4b. Therefore, for the EC dataset only considered the motifs colored with orange nodes in Figure 3.4b. For approximate algorithms, we compared our algorithms PRESTO-A and PRESTO-E, collectively denoted as PRESTO, with 2 baselines: the sampling algorithm by Liu et al. [2019], denoted by LS and the sampling algorithm by Wang et al. [2020], denoted by ES.

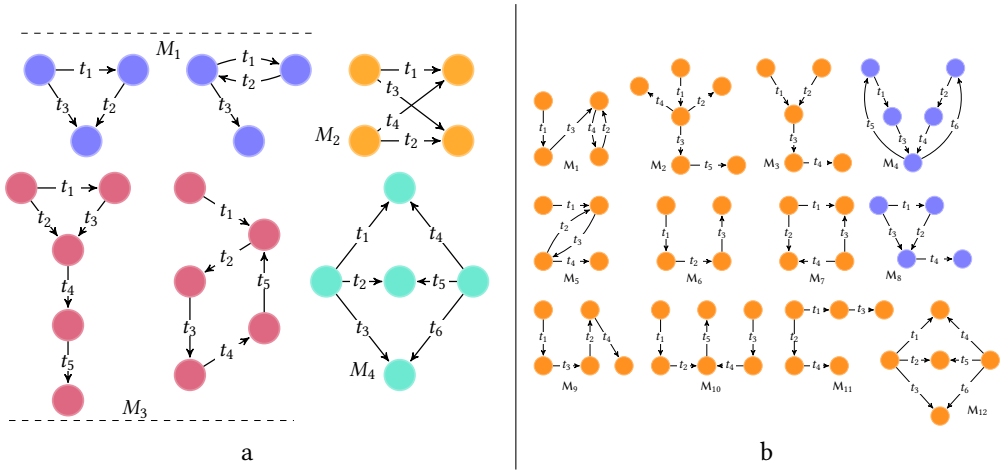


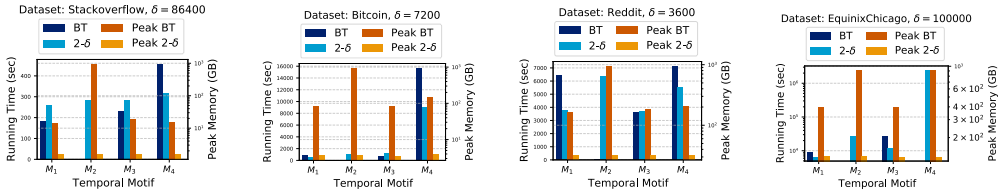
Figure 3.4: Motifs used in the experimental evaluation. The edge labels denote the edge order in σ according to Definition 3.2. (3.4a): Motifs used for the comparison of exact algorithms. When we denote M_1 and M_3 we refer to the motifs on the left on all datasets but the EC dataset, given that such dataset is bipartite and it cannot contain the motifs on the left. Therefore on the EC dataset we will use the motifs on the right denoted with the same labels M_1 and M_3. (3.4b): Motifs used for the comparison of approximate algorithms: from M_1 to M_{12}, motifs coloured in blue (i.e., M_4, M_8) will not be used on the EC dataset, since such network is bipartite and it cannot contain such motifs.

3.6.2 2δ-patch: Comparison with BT

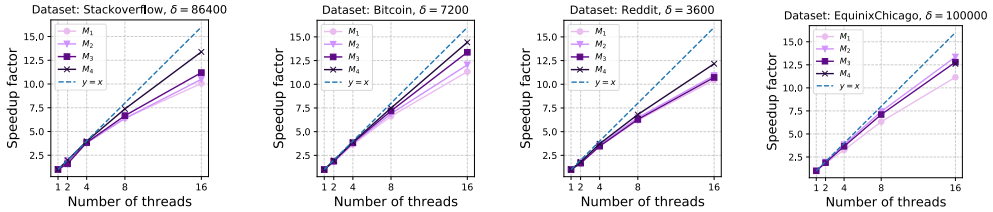
In this section we compare the state-of-the art exact algorithm by Mackey et al. [2018] (denoted with BT) and our Algorithm 1 (denoted with 2-δ) to compute the exact count of temporal motifs. Our 2-δ uses the BT algorithm by modifying the ordering used to match the temporal motif instances as discussed in section 3.4.2.

3.6.2.1 Setup

We first will show that by running both the algorithms (BT and 2-δ) sequentially, our algorithm 2-δ enables an efficient and scalable counting of temporal motifs over large temporal networks, differently from BT that requires significantly more resources (time and memory), hence becoming impractical on many inputs. To show such aspect we executed both the algorithms on the datasets from Table 3.1 and with the four motifs (with different structure and number of temporal edges, i.e., $\ell \in [3, 6]$) from Figure 3.4a. Note that on the EC dataset M_1 and M_3 are different with respect to the other datasets since such dataset is bipartite, and therefore it cannot contain triangle-shaped interactions. Additionally, M_2 is chosen such that it creates disconnected components when matching it inside the BT algorithm (see Section 3.4.2), so we expect our 2-δ patch algorithm to improve significantly the running time and memory when counting δ-instances of such



a



b

Figure 3.5: (3.5a): Average running time (blue bars) and peak RAM usage (orange bars) comparison of the BT and 2-δ patch algorithms on the motifs of Figure (3.4a) on the different datasets of Table 3.1. (3.5b): Parallel speed-up achieved over the sequential implementation of the 2-δ patch algorithm with different threads on each dataset and motif, we also report on each plot the function $y = x$ that represents the best speedup achievable by our algorithm.

motif over BT. We executed each algorithm on the given dataset and configuration of parameters (motif, δ, and $c = 2$ for 2-δ) five times and measured the resulting average running time for each of the two algorithms, additionally over an independent run we measured the peak RAM usage of each of the two algorithms. For each dataset we used different values of δ given that smaller datasets (in terms of temporal edges m) allow for a higher value of such parameter (given that they contain fewer edges in each temporal window of at most δ), we only highlight that such choices of parameters were made to facilitate the BT algorithm to terminate, since with higher values of δ on larger datasets, for some motifs, such algorithm may end up using too much memory or running time, differently from our 2-δ patch. The values of δ will be reported along the various experiments.

3.6.2.2 Resources Comparison

The results are reported in Figure (3.5a). By considering the running time we see that on the datasets SO and BI on motifs M_1 and M_3 both BT and 2-δ achieve similar performances (BT is slightly faster on SO and slightly slower on M_3 on the BI dataset). Interestingly on the larger dataset (RE) such motif counts are computed in a similar time (on M_3) or in a much faster way by the 2-δ algorithm (M_1's count is computed saving more than 33% of the running time w.r.t. to BT). Then we note that M_4 is computed much more faster by 2-δ w.r.t. BT on all datasets, for example on the BI dataset 2-δ saves

50% of the running time of BT (that corresponds to more than 2 hours), this is probably achieved by our ordering adopted in 2-δ (resulting from the algorithm in Section 3.4.2), this is really surprising since M_4 is not temporally disconnected so our proposed matching algorithm may achieve good performances on arbitrary motifs, not only on disconnected ones. Interestingly, on EC and M_4, BT was not able to conclude due to large memory usage, while 2-δ-patch used a limited amount of memory, but required more than 2-weeks to compute such motif count. Still on the EC dataset we note that both M_1 and M_3 (recall, such motifs differ from the ones on previous datasets) are computed in a similar or smaller time than BT, supporting the general trend of 2-δ patch being more efficient that BT. Finally we conclude by noting that BT cannot compute the count of M_2 on any of the datasets, even the smallest one, given that it generates disconnected matches, this is supported by the fact that the RAM peak of BT corresponds to the maximum available on the machine (therefore the execution was terminated for exceeding the available memory). Considering the memory usage we observe consistently 2-δ patch uses at least one order of magnitude less RAM memory than BT and less than two orders of magnitude on motif M_2 on each dataset.

3.6.2.3 2-δ Patch Parallel

On each of the configurations between motif, dataset, and δ used in the previous section we executed the 2-δ algorithm's parallel version. We executed each parameter combination five times and computed the resulting average running time, let RT_1 be the (average) running time obtained from executing 2-δ sequentially and $RT_\tau, \tau \in \{2, 4, 8, 16\}$ the (average) running time obtained by executing 2-δ with τ threads. We computed $RT_1/RT_\tau, \tau \in \{2, 4, 8, 16\}$ that is the speedup achieved by the parallel implementation using τ threads. We report the results of all the configurations in Figure (3.5b). We observe that on all the configurations the speedup is almost linear up to 8 threads, and becomes slightly less than linear for 16 threads, except for the BI dataset that maintains an almost linear speedup on most of the motifs.

Combining the results of the running time, memory and parallel implementation we have that 2-δ achieves significant results towards making practical the computation of arbitrary temporal motif counts, for example while on BI dataset counting M_4 with BT takes more that 4 hours, by using more than 100GB of RAM memory of peak, with 2-δ in its parallel version we can obtain such count in less that 10 minutes by using less than 20 GB of peak RAM memory. We enforce that while BT is not parallelizable, our 2-δ is embarrassingly parallel, and can be used to parallelize any sequential algorithm for exact enumeration. Additionally our 2-δ enables the counting of arbitrary motifs differently from BT that has a large memory usage, as we discussed for M_2 (or M_4 on EC) that cannot with BT since such algorithm exceeds the memory of our machine.

3.6.3 Quality of Approximation

We start by comparing the approximation provided by PRESTO with the current state-of-the-art sampling algorithms. To allow for a fair comparison, since the algorithms we consider depend on parameters that are not directly comparable and that influence both the algorithm's running time and approximation, we run all the algorithms sequentially and measure the approximation they achieve by fixing their running time (i.e., we fix the parameters to obtain the same running time).

While PRESTO can be trivially modified in order to work within a user-specified time limit, the same does not apply to LS or ES. Thus, to fix the same running time for all algorithms we devised the following procedure: i) for a fixed dataset and δ, we found the parameters of LS such its runtime is significantly faster[2] than that of the 2-δ patch algorithm, that we executed in parallel to extract the exact counts on each motif[3] ii) we run LS on all the motifs on the fixed dataset with the parameters set as from i) and measure the time required for each motif; iii) we run ES such that it requires the same time as LS; iv) we ran PRESTO-A and PRESTO-E with $c = 1.25$ and a time limit that is the *minimum* between the running time of LS and the one of ES. The parameters for all algorithms are in Section 3.7.3.2. Further discussion on how to set c when running PRESTO is in Section 3.7.4.

For each configuration, we run each algorithm ten times, with a limit of 600 GB of (RAM) memory. We computed the so called Mean Average Percentage Error (MAPE) on the ten runs. MAPE is defined as follows: let C'_M be the estimate of the count C_M produced by one run of an algorithm, then the relative error of such estimate is $|C'_M - C_M|/C_M$; the MAPE of the estimates is the average of the relative errors in percentage.

Table 3.2 shows the MAPE of the sampling algorithms on SO and BI, with their variances across the ten runs. On the SO dataset we observe that both versions of PRESTO provide more accurate estimates over current state-of-the-art sampling algorithms. PRESTO-A provides the best approximation on 5 out of 12 motifs, with PRESTO-E providing the best results for all the other motifs. Furthermore, both variants of PRESTO improve on all motifs w.r.t. LS and ES, with the error from ES being more than twice the error of PRESTO on all motifs. We note PRESTO-A and PRESTO-E achieve similar results, supporting the idea that similar performances are obtained when the network's timestamps are distributed evenly, that is the case for such dataset (see Section 3.7.7). Interestingly on this dataset PRESTO is able to estimate even small counts with sufficient accuracy, note in fact that motif M_4 has only ~320 δ-instances, and PRESTO is able to estimate its count within 16% of relative error. We also note that the variance of our algorithm is almost always smaller than the one of other algorithms, highlighting that we can

[2]We use such criterion since sampling algorithms are often required to run within a small fraction of time w.r.t. exact algorithms.

[3]We bound the running time of the exact algorithm using half the number of threads used times the running time by such parallel version, i.e., assuming a pessimistic speedup.

Table 3.2: Comparison of the relative approximation error (see Section 3.6.1). For each combination of dataset and motif we highlight the algorithm with minimum MAPE value. "✗" that the motif cannot be counted by the method due to ℓ being too large.

	SO					BI				
		Approximation Error					Approximation Error			
		PRESTO					PRESTO			
Motif	C_M	A	E	LS	ES	C_M	A	E	LS	ES
M_1	$2.0{\cdot}10^7$	**1.1±0.9%**	1.2±0.5%	22.2±18.9%	4.4±2.9%	$4.2{\cdot}10^8$	**3.0±2.3%**	9.3±7.7%	31.4±27.1%	10.2±6.3%
M_2	$5.7{\cdot}10^9$	13.6±14.3%	**8.9±8.8%**	16.0±11.4%	✗	$6.3{\cdot}10^{12}$	**11.5±9.7%**	12.7±11.5%	31.8±12.0%	✗
M_3	$7.8{\cdot}10^8$	6.1±4.8%	**3.6±3.5%**	6.3±4.2%	14.9±11.5%	$1.9{\cdot}10^{11}$	**10.4±9.5%**	13.5±8.7%	23.1±21.3%	27.1±40.2%
M_4	$3.2{\cdot}10^3$	16.2±9.3%	**10.8±7.5%**	75.3±29.1%	✗	$1.5{\cdot}10^7$	53.1±38.5%	**40.1±32.6%**	128.4±109.3%	✗
M_5	$1.1{\cdot}10^8$	4.6±3.9%	**2.7±1.7%**	6.1±4.9%	13.3±8.8%	$1.0{\cdot}10^9$	**6.9±4.4%**	7.8±4.6%	153.9±26.3%	137.1±70.0%
M_6	$2.3{\cdot}10^8$	4.9±4.8%	**3.5±1.9%**	6.5±3.5%	6.0±4.3%	$3.8{\cdot}10^9$	7.1±3.0%	**3.8±3.2%**	9.1±4.3%	9.4±6.5%
M_7	$8.0{\cdot}10^5$	**3.8±2.3%**	4.3±2.2%	8.6±5.5%	17.4±23.7%	$7.2{\cdot}10^7$	13.8±8.7%	**13.1±9.2%**	16.9±11.9%	15.3±8.6%
M_8	$6.8{\cdot}10^6$	5.0±2.3%	**4.5±2.9%**	5.6±4.4%	25.9±18.4%	$1.4{\cdot}10^8$	11.4±8.6%	**11.2±9.7%**	42.3±50.6%	21.0±19.9%
M_9	$1.5{\cdot}10^8$	**0.9±0.7%**	1.6±1.3%	38.5±33.2%	3.4±2.7%	$2.5{\cdot}10^9$	**4.6±3.7%**	4.9±3.1%	118.4±97.8%	79.4±62.8%
M_{10}	$1.2{\cdot}10^9$	**1.4±1.1%**	2.0±1.1%	51.1±32.3%	✗	$1.4{\cdot}10^{11}$	**2.9±1.6%**	5.4±10.1%	34.2±21.2%	✗
M_{11}	$2.3{\cdot}10^8$	5.2±5.9%	**3.6±2.4%**	6.4±3.8%	17.8±15.0%	$2.6{\cdot}10^9$	6.1±3.0%	**4.8±2.9%**	81.7±15.5%	71.2±53.8%
M_{12}	$2.3{\cdot}10^4$	**9.5±6.9%**	12.4±8.3%	21.0±14.3%	✗	$2.4{\cdot}10^8$	**12.9±12.8%**	15.9±7.8%	51.0±29.8%	✗

achieve tighter and more concentrated estimates in the same amount of time of the baselines.

For the BI dataset the results are similar to the ones for the SO dataset. PRESTO-A achieves the lowest approximation error on 7 out of 12 motifs with PRESTO-E achieving the lowest approximation error on all the other motifs. PRESTO achieves better estimates than LS and ES on all the motifs as for the SO dataset. Interestingly, on this dataset where some motifs are easily counted approximately, while others such as M_4 have a worse approximation factor, this may be related to their distribution in the temporal network. Despite this, PRESTO significantly improves the results of the state of the art algorithms on most of the motifs.

On the RE dataset shown in Table 3.3, PRESTO outperforms over all the motifs on ES and over 6 motifs on LS, with PRESTO-A achieving the lowest approximation error on 4 out of 12 motifs and PRESTO-E on 3 out of 12 motifs. Even when not improving LS (on 6 out of 12 motifs) we note that the average error is quite close to such algorithm, while the average error of LS is often larger on the motifs where PRESTO performs better. This may be related to the edge distribution of the dataset RE, since Wang et al. [2020] showed that RE has a more skewed edge distribution, a scenario for which LS may be competitive when there is a positive correlation between edges and motif instances inside sampled windows [Liu et al., 2019].

Finally we discuss the results on the EC dataset, again in Table 3.3, which is a 2.3 billion edges *bipartite* temporal network. Note that the results for ES are missing, since ES did not complete any run with 600GB of memory on the motifs tested. (We discuss the high memory usage of ES in Section 3.6.4). On such dataset both variants of PRESTO perform better than LS on almost all the motifs considered, PRESTO-A achieves the lowest approximation error on 6 out 10 motifs, while PRESTO-E improves over LS on all motifs but M_4, achieving the best approximation error on 4 out of 10 motifs. Interestingly, the variance of PRESTO is often similar or significantly lower than the one

Table 3.3: Comparison of the relative approximation error (see Section 3.6.1). For each combination of dataset and motif we highlight the algorithm with minimum MAPE value. "✗" that the motif cannot be counted by the method due to ℓ being too large. For the EC dataset we do not report values for ES since such algorithm was not able to complete any run due to out of memory error.

	RE					EC			
		Approximation Error					Approximation Error		
		PRESTO					PRESTO		
Motif	C_M	A	E	LS	ES	C_M	A	E	LS
M_1	$1.6{\cdot}10^8$	**4.2±2.1%**	4.4±3.0%	27.5±20.2%	12.7±7.0%	$1.2{\cdot}10^{11}$	**2.7±2.2%**	43.2±17.1%	70.5±41.1%
M_2	$2.0{\cdot}10^{10}$	18.6±3.1%	18.5±6.1%	**17.1±7.9%**	✗	$2.6{\cdot}10^{12}$	**6.6±4.6%**	15.5±7.7%	28.6±18.8%
M_3	$7.8{\cdot}10^9$	**10.1±9.0%**	15.1±12.0%	10.3±9.0%	24.0±17.5%	$6.3{\cdot}10^{10}$	4.5±3.6%	**4.5±2.8%**	7.9±5.1%
M_4	$1.6{\cdot}10^7$	**11.7±15.5%**	22.5±14.4%	88.5±8.5%	✗	-	-	-	-
M_5	$6.7{\cdot}10^8$	**23.1±16.1%**	23.8±26.5%	29.0±12.4%	31.1±10.2%	$2.7{\cdot}10^{11}$	9.4±4.9%	**6.9±3.2%**	13.8±11.6%
M_6	$5.0{\cdot}10^8$	4.7±4.2%	3.2±1.4%	**2.0±1.0%**	10.7±7.0%	$1.6{\cdot}10^{10}$	5.3±2.4%	**3.3±2.4%**	12.4±7.9%
M_7	$1.1{\cdot}10^8$	10.3±6.4%	**8.1±4.0%**	9.6±4.5%	21.7±17.2%	$3.4{\cdot}10^{11}$	**11.8±7.4%**	13.7±9.1%	19.1±8.8%
M_8	$8.4{\cdot}10^7$	12.6±6.9%	17.5±8.9%	**9.4±7.6%**	26.6±20.6%	-	-	-	-
M_9	$5.9{\cdot}10^8$	2.2±1.5%	**2.1±1.4%**	34.3±33.7%	5.1±4.0%	$8.7{\cdot}10^{10}$	**1.1±1.2%**	50.6±17.2%	67.7±52.2%
M_{10}	$3.8{\cdot}10^9$	9.3±4.6%	**8.4±12.6%**	92.0±60.9%	✗	$2.3{\cdot}10^{11}$	**5.5±2.6%**	52.3±25.5%	61.7±25.4%
M_{11}	$7.0{\cdot}10^8$	6.8±5.0%	4.8±2.7%	**3.4±3.5%**	31.9±27.5%	$9.5{\cdot}10^{10}$	7.1±7.8%	**3.8±2.1%**	7.6±6.1%
M_{12}	$1.0{\cdot}10^8$	62.0±23.4%	57.0±18.7%	**53.3±58.8%**	✗	$4.0{\cdot}10^{13}$	**48.8±30.1%**	93.1±7.8%	62.0±29.4%

Table 3.4: Minimum and maximum peak memory usage in GB over all motifs in Figure 3.4b, "✗" denotes out of memory.

Dataset	PRESTO-A	PRESTO-E	LS	ES
SO	1.5 - 1.5	1.5 - 1.5	1.5 - 13.6	9.7 - 39.3
BI	3.5 - 3.5	3.5 - 3.5	3.5 - 16.4	31.3 - 42.0
RE	19.5 - 19.5	19.5 - 19.5	19.5 - 79.8	124.9 - 498.0
EC	71.0 - 71.0	71.0 - 71.0	71.0 - 85.0	✗

of LS, which is another of the advantages of our algorithms, as confirmed by the experiments we already discussed.

These results, coupled with the theoretical guarantees of Section 3.5, show that PRESTO outperforms the state-of-the-art sampling algorithms LS and ES for the estimation of most of the motif counts on temporal networks. Based on the results we discussed, PRESTO-E seems also to usually provide similar estimates to PRESTO-A, while sometimes improving over PRESTO-A depending on the distribution of the network edges or motif instances.

3.6.4 Memory Usage

In this section we discuss the RAM usage of the various sampling algorithms. For each motif in Figure 3.4b with the parameters set as in Section 3.6.3, we computed for each motif the peak RAM required for the approximation of its count by the various approximation algorithms. The results are in Table 3.4, which shows for each algorithm the minimum and maximum amount of peak (RAM) memory used over all motifs in Figure 3.4. Both PRESTO's versions have lower memory requirements than ES, and equal or lower mem-

ory requirements than LS, on all datasets. ES ranges from requiring 6.5× up to 25.5× the memory used by PRESTO, making ES not practical for very large datasets such as EC (where ES did not completed any of the runs since it exceeded the 600GB of memory allowed for its execution). While LS and PRESTO have similar memory requirements when compared to the minimum peak RAM required, but LS requires more maximum peak RAM, such as up to 9× on dataset SO, which may be due to the fact that choosing windows with more edges, as done by LS, may require more memory to run the exact algorithm as subroutine. PRESTO is, thus, a memory efficient algorithm and hence a very practical tool to tackle the motif counting problem on temporal networks up to billions edges.

3.7 Additional Material

3.7.1 Missing Theoretical Results

Theorem 3.5 ([Hoeffding, 1963]). *Let $X_1, \ldots, X_s$ be independent random variables such that for all $1 \leq i \leq s$, $\mathbb{E}[X_i] = \mu$ and $\mathbb{P}(a \leq X_i \leq b) = 1$. Then*

$$\mathbb{P}\left(\left|\frac{1}{s}\sum_{i=1}^{s} X_i - \mu\right| \geq t\right) \leq 2\exp\left(-\frac{2st^2}{(b-a)^2}\right)$$

Proof of Theorem 3.1. Let $Pr = \mathbb{P}(|\frac{1}{s}\sum_{i=1}^{s} X_i - C_M| \geq \varepsilon C_M)$, we want to show that for s as in statement it holds $Pr \leq \eta$. Observe that the variables $X_1, \ldots, X_s$, are mutually independent. And, by Lemma 3.2, $\mathbb{E}[X_i] = C_M = \mu, i = 1, \ldots, s$. Then for $X_i, i = 1, \ldots, s$, the following holds:

$$X_i = \sum_{u \in \mathcal{U}} \frac{1}{p_u} X_u^i \overset{(1.)}{\leq} \sum_{u \in \mathcal{U}} \frac{\Delta_{T,1}}{(c-1)\delta} \leq \frac{C_M \Delta_{T,1}}{(c-1)\delta} \tag{3.3}$$

where (1.) follows from the definition of p_u, by considering each of the variables $X_u^i = 1$, and noting that for each $u \in \mathcal{U}$ it holds $r_u = c\delta - (t_\ell^u - t_1^u) \geq (c-1)\delta$. Thus w.p. 1, $a = 0 \leq X_i \leq C_M \Delta_{T,1}/((c-1)\delta) = b, i = 1, \ldots, s$. Now let $t = \varepsilon C_M$. Then, $(b-a)^2 = \left(\frac{\Delta_{T,1}}{(c-1)\delta} C_M - 0\right)^2 = \frac{\Delta_{T,1}^2 C_M^2}{(c-1)^2\delta^2}$ and applying Theorem 3.5 to bound Pr we obtain,

$$Pr \leq 2\exp\left(-\frac{2s\varepsilon^2 C_M^2}{\frac{\Delta_{T,1}^2 C_M^2}{(c-1)^2\delta^2}}\right) \leq \eta$$

by the choice of s as in statement. □

3.7.1.1 Note on Bennett's Inequality

In this section we will discuss how to apply Bennett's inequality when the exact variance term, i.e. v, from Theorem 3.2 is unknown and we only have an upper bound $\hat{v}$ to such term (which is our case in practice).

Let us consider the function $f(v) = \alpha v\, h(\gamma/v)$ with $\alpha = s/B^2$, $\gamma = tB$ where $h(\cdot)$ is the same function $h(\cdot)$ appearing in Bennett's inequality (Theorem 3.2). With a somehow tedious analysis (see Lemma 3.6) one can prove that the function $f(v)$ is continuous and monotonic non increasing for every $v, \gamma > 0$ thus by definition, given $v_1 \geq v_2 > 0$ it holds $f(v_1) \leq f(v_2)$. Observe that we can write the right-hand side of the Bennett's bound as $2\exp(-f(v))$, now is clear that given $v_1 \geq v_2 > 0$ it holds $2\exp(-f(v_2)) \leq 2\exp(-f(v_1))$ by the property of negative exponential functions. This observation allows us to apply Bennett's inequality using an upper bound $\hat{v}$ to v.

By the above observation we also get the intuition that a smaller gap between $\hat{v}$, the upper bound to the variance, and v the actual variance will result in a tighter bound to the sample size s (i.e., Bennett's inequality will provide more strict bounds). In our work we focused on a trade-off between the sharpness of the bound $\hat{v}$ and its efficient computability in practice, since sharper bounds on $\hat{v}$ (than the one we provide) could be proved but they are much harder to compute and thus may not be of practical interest. We still leave to further research the space of improving the bounds in this work.

Lemma 3.6. *The function $f(v) = \alpha v h(\gamma/v)$ with $h(x) = (1+x)\ln(1+x) - x$ is monotonic non-increasing for every $\alpha, v, \gamma > 0$*

Proof. Recall the definition of the function $f(v) = f(v) = \alpha v \left((1+\frac{\gamma}{v})\ln(1+\frac{\gamma}{v}) - \frac{\gamma}{v}\right)$, let us take its first derivative w.r.t. to v.

$$\begin{aligned}\frac{\partial f(v)}{\partial v} &= \frac{\partial}{\partial v}\alpha\left[v\ln(1+\frac{\gamma}{v}) + \gamma\ln(1+\frac{\gamma}{v}) - \gamma\right] \\ &= \alpha\left[\ln(1+\frac{\gamma}{v}) - \frac{\gamma}{(v+\gamma)} - \frac{\gamma^2}{v(v+\gamma)}\right]\end{aligned}$$

Now we need to study the sign of the first derivative,

$$\begin{aligned}&\alpha\left[\ln(1+\frac{\gamma}{v}) - \frac{\gamma}{(v+\gamma)} - \frac{\gamma^2}{v(v+\gamma)}\right] > 0 \leftrightarrow \\ &\ln(1+\frac{\gamma}{v}) - \frac{\gamma}{(v+\gamma)} - \frac{\gamma^2}{v(v+\gamma)} > 0 \leftrightarrow \\ &v^2\ln(1+\frac{\gamma}{v}) + \gamma v\ln(1+\frac{\gamma}{v}) - \gamma v - \gamma^2 > 0 \leftrightarrow \\ &\frac{v^2}{\gamma^2}\ln(1+\frac{\gamma}{v}) + \frac{v}{\gamma}\left(\ln(1+\frac{\gamma}{v}) - 1\right) - 1 > 0 \leftrightarrow \\ &y^2\ln(1+\frac{1}{y}) + y(\ln(1+\frac{1}{y}) - 1) - 1 > 0 \leftrightarrow \\ &(y+1)(y\ln(1+\frac{1}{y}) - 1) > 0\end{aligned}$$

By setting $y = v/\gamma$, then we first note that $y + 1 > 0$ for every value of $v, \gamma > 0$. We only need to study the sign of the function $(y\ln(1+\frac{1}{y}) - 1)$ i.e.,

find the values of $y > 0$ s.t. $g(y) = y \ln(1 + \frac{1}{y}) > 1$. Since for $y > 0$ it holds $g(y) \leq 1$ it cannot be that for any $y > 0$ $g(y) - 1 > 0$ therefore the function is non-increasing over the domain of interest as claimed. □

3.7.1.2 On the Derivation of Sharper Guarantees

In this section we discuss some limitations to provide sharper guarantees on the concentration of the estimator adopted in PRESTO, we hope such section to be of inspiration for future works.

In recent years, several concentration inequalities for the sum of independent random variables have been proposed to improve Bennett's inequality. Leveraging for example, on higher moments (e.g., [Zheng, 2017, Light, 2020]) of the random variables considered (i.e., bounding $\mathbb{E}[X_i^p], p > 1, i \in [1, s]$, see for example Lemma 3.7). While such inequalities do provide sharper results than Bennett's inequality, we argue that the main backdraw of applying such results in our scenario is the gap between the bound on the range of the variables X_i, and the values that are assumed in practice by such variables, given that our bound assumes a rather pessimistic worst case scenario. Observe in fact that the bound on the range of X_i is $[0, \frac{\Delta_{\mathcal{T},1} C_M}{(c-1)\delta}]$ may be very loose, especially w.r.t. scenarios where the distribution of the δ-instances is not as assumed by such worst case scenario (this is almost always the case in practical applications). Therefore such novel bounds lead only to small improvements over the guarantees computed by using Bennett's inequality. Observe in fact that the bound on the range of the variables plays a key role in determining the number of samples needed for the concentration of our estimator.

Lemma 3.7. *Given $p > 1$ for the r.v. X_i computed at the i-th iteration ($i \in [1, s]$) by* PRESTO-A*, it holds that,*

$$\mathbb{E}[X_i^p] \leq \left(\frac{\Delta_{\mathcal{T},1}}{(c-1)\delta}\right)^{p-1} C_M^p$$

Another line of research in concentration theory aims at using the *empirical* values of the variables, for example leveraging on their variance to provide sharper concentration results [Maurer and Pontil, 2009]. Again to apply such results to our scenario we need to obtain a bound on the range of the variables X_i, and as argued before our bound may be too loose to obtain significant improvements. Additionally in this setting, due to some technical challenges the fact that the range depends on C_M prevents us to employ these results to obtain desired bounds. We leave for future research obtaining reasonable bounds on the range of X_i being efficiently computable, since trivial techniques to improve such bound would make our algorithm very inefficient, therefore not practical.

3.7.2 Description of the Equinix-Chicago Dataset

In this section, we briefly describe the Equinix-Chicago used in our experimental evaluation.

We built the Equinix-Chicago dataset[4] starting from the internet packets collected by two monitors situated at Equinix data center, which captured the packets from Chicago to Seattle and vice versa on the 17 February 2011 for an observation period of 62 minutes. Each edge (x, y, t) represents a packet sent at time t, with microseconds precision, from the IPv4 address x to the IPv4 destination y[5]. Such network is thus a *bipartite* temporal network since edges, i.e., packet exchanges, occur only between nodes belonging to different cities, with no interactions captured among nodes of the same city. The final temporal network has more than 2.325 billion temporal edges and 11.157 million nodes.

We made available this dataset to the community at the following link: https://drive.google.com/drive/folders/1HXMEO4wwMOT1H9icwiP6siQTp2sm5pgA.

3.7.3 Reproducibility

3.7.3.1 Links to External Resources

For reproducibility purposes, in this section we report the links to the datasets and implementations of LS and ES we used.

- Dataset SO is available at: http://snap.stanford.edu/temporal-motifs/data.html;
- Datasets RE and BI are available at http://www.cs.cornell.edu/~arb/data/;
- The current implementation of the LS algorithm is available at: https://gitlab.com/paul.liu.ubc/sampling-temporal-motifs;
- The implementation of the ES algorithm is available at: https://github.com/jingjing-hnu/Temporal-Motif-Counting.

3.7.3.2 Reproducibility - Parameters of the Experimental Evaluation

We now discuss discuss the choice of the parameters resulting from our procedure for comparing the different sampling algorithms in Sec. 3.6.3.

[4]We used data available at https://www.caida.org/data/passive/passive_2011_dataset.xml publicly available under request.

[5]We filtered packets which used different protocols, e.g., IPv6. Further, we did not assigned nodes at port level but at IP address level (e.g, let addr be an IPv4 address, then the edges corresponding to two packets sent at different ports: addr:80, addr:22 are mapped on the same endpoint node corresponding to addr).

Table 3.5: Parameters used in the comparison of LS, ES, PRESTO-A and PRESTO-E. For the parameter s r, p we report the range since their values depends on the specific motifs (see Sec.3.6.3). We further discuss in Supplement 3.7.4 the motivation for our choice of c.

Dataset	c	δ	r	b	p
SO	1.25	86400	10-40	5	$10^{-[5,4]}$
BI	1.25	7200	5-80	5	$10^{-[5,3]}$
RE	1.25	3600	80-1000	5	$10^{-[5,4]}$
EC	1.25	100000	2-240	5	10^{-5}

Note that our algorithms PRESTO-A and PRESTO-E have only 1 tuning parameter in the experimental setting of Sec. 3.6.3, since δ is provided in input (i.e., c, for which we also discuss how to set it to obtain the maximum efficiency in Supplement 3.7.4). The same holds for ES which has the parameter p, that controls the fraction of sampled edges (see Sec. 3.3). While LS has instead non trivial dependencies among the parameters b and r which are not entirely easy to set when it comes to different values of c and δ also due to the lack of a qualitative theoretical analysis relating the parameters to the approximation quality of the results.

Table 3.5 reports all the parameters used on the different datasets. Given the procedure for running PRESTO's versions (described in Section 3.6.3), their sample size s is not deterministic, since only the execution time is fixed. Additionally for some of the hardest motifs for LS (i.e., motifs M_1, M_8, M_9) we used small values of r, while for all the other motifs we used higher values of r, the results values are reported in Table 3.5. Under the column r we report on the left the value used for hard motifs, and on the right the value for all the other motifs. Recall that the values of p for ES where adapted to each motif on each configuration following the approach described in Section 3.6.3, we selected the best p in order to have a final running time of ES to be close to the one of LS choosing p as follows, we started from $p = 10^{-6}$ and after executing LS, we iteratively increased p till the running time of ES was close or slightly greater (of at most 10%) to the running time of LS, the resulting values of all parameters are reported in Table 3.5. Thus, we report in Table 3.5 the range from which p was chosen in practice as result of our procedure for fixing the parameters.

3.7.4 Selecting the Value of c

In this section we briefly discuss how to choose the value of c when running PRESTO.

The major advantages of PRESTO, come from the efficiency in time, small memory usage and from the scalability it achieves, which coupled with the theoretical guarantees provided and the precise estimates achieved in practice, make our algorithm a fundamental tool when analysing large temporal networks, it is thus important to set PRESTO's parameter (i.e., c) correctly to

Table 3.6: Running times (geometric mean of five different runs) of PRESTO-A using 8 threads on motif M_7 on EC dataset.

c	s	running time (sec)
10	125	242.09
5	250	128.94
2.5	500	76.43
1.25	1000	61.41

exploit such features at their best. Our versions of PRESTO have only the parameter c to be selected, since δ is part of the problem's input. We now discuss the importance of keeping c small (e.g., $c < 2$). To do this, we consider a concrete example to show that a small c leads to more efficiency and scalability. Suppose we want to approximate the count of motif M_7 from Figure 3.4 on EC dataset[6] and we want to cover with samples a fixed length of the interval $[t_1, t_m]$ over the s iterations, i.e., $\sum_{i=1}^{s} |t_r^i + c\delta - t_r^i| = sc\delta = V$ where $[t_r^i, t_r^i + c\delta]$ is the temporal window sampled by PRESTO-A[7] at iteration $i = 1, \ldots, s$ and V is the fixed value, V can be interpreted as the summation of the durations of each sample ($c\delta$) over all the samples (i.e., how much time of $[t_1, t_m]$ over the s iterations PRESTO examines). Fixed V and δ, one can explore different values of c which thus let only one possibility to set s, by solving the equation $s = V/c\delta$. Table 3.6 presents the running times for various values of c with fixed $\delta = 10^5$, $V = 1.25 \cdot 10^8$. When c decreases there is a substantial decrease in running time, in fact with $c = 10$ the running time is more than 3.9× the running time with $c = 1.25$. This should not surprise the reader, since such results is in accordance with our complexity analysis (see Section 3.5.4) which shows that larger samples in terms of edges (as obtained by sampling windows with a higher c) result in a higher running time.

Thus the best choice in order to set c is to keep $c < 2$ based on all the experiments we performed, to exploit efficiency and scalability at the maximum of their level.

3.7.5 Running Time Comparison

In this section we discuss the running times under which we obtained the results in Section 3.6.3.

We measured the running time of PRESTO-A, PRESTO-E, LS, and ES over all experiments (i.e., datasets and motifs) described in Section 3.6.3, we also measured the running time of 2δ over one run on the same configurations (i.e., dataset ad δ) executed with 20 cores. For all algorithms we then computed the average of their running times for each pair (dataset, motif), for 2δ we will assume a speedup of 10, that is supported by our experiments

[6] This is an arbitrary example, similar results can be observed on almost all the motifs and datasets.

[7] Similar observations hold for PRESTO-E.

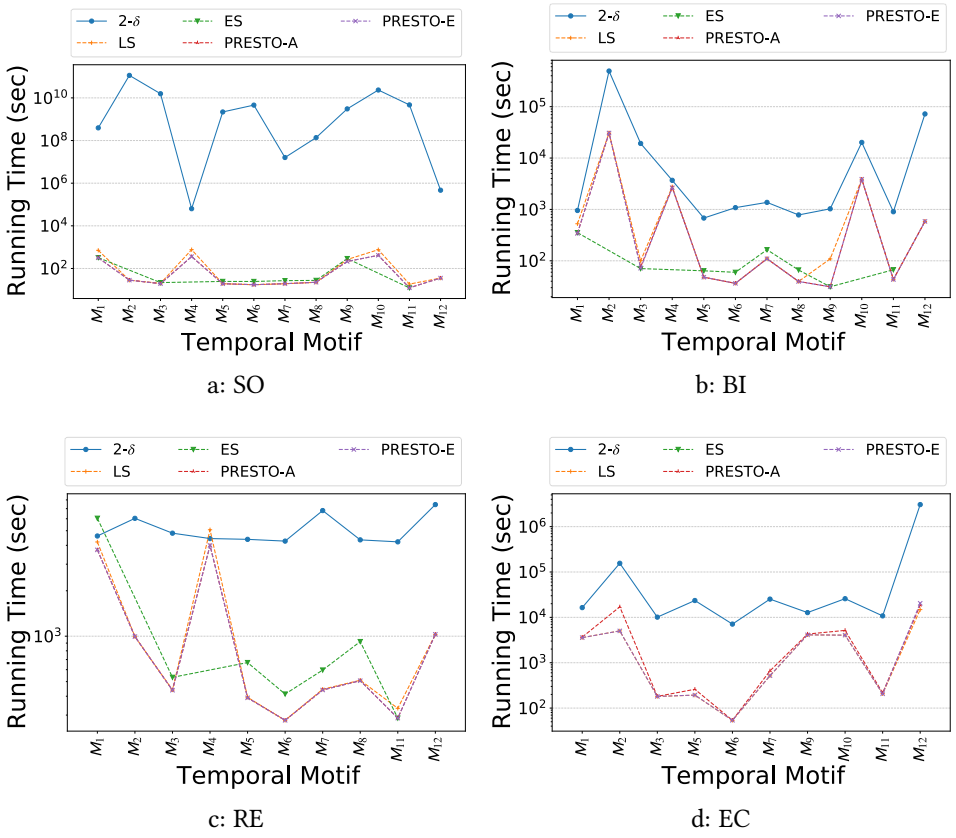


Figure 3.6: Running times on the different datasets, according to the procedure in Sec. 3.6.1.

(see Section 3.6.2), therefore let $t_{2\delta}$ be the running time of 2δ obtained on a specific configuration, when we will report its running time we will report $10 \cdot t_{2\delta}$. The results are shown in Figure 3.6. As expected, PRESTO runs almost always significantly faster than 2δ. The reader may appreciate how the results in Sec. 3.6.3 were obtained with PRESTO's running time bounded by the minimum running time among ES and LS (fixed the motif and the dataset). Furthermore we note that on several motifs on almost all datasets, even if we set p small, i.e., 10^{-5}, ES runs within a higher time than LS and thus than PRESTO, we believe that this may be due to ES's procedure to explore the search space which may not suite such motifs. To this end we find that PRESTO is not affected by the motif topology. Note that PRESTO runs orders of magnitude faster than 2δ while still providing accurate estimates. For example on M_6 PRESTO achieves a 3% relative error (see Table 3.3) while being 49× faster than 2δ.

Coupled with the results of Section 3.6.3 and 3.6.4, these aspects show that PRESTO is an efficient algorithm to obtain accurate estimates of motif counts from large temporal networks, especially when the running time to obtain an estimate is very limited, this may be the case on very large datasets.

3.7.6 Scalability of Parallel PRESTO

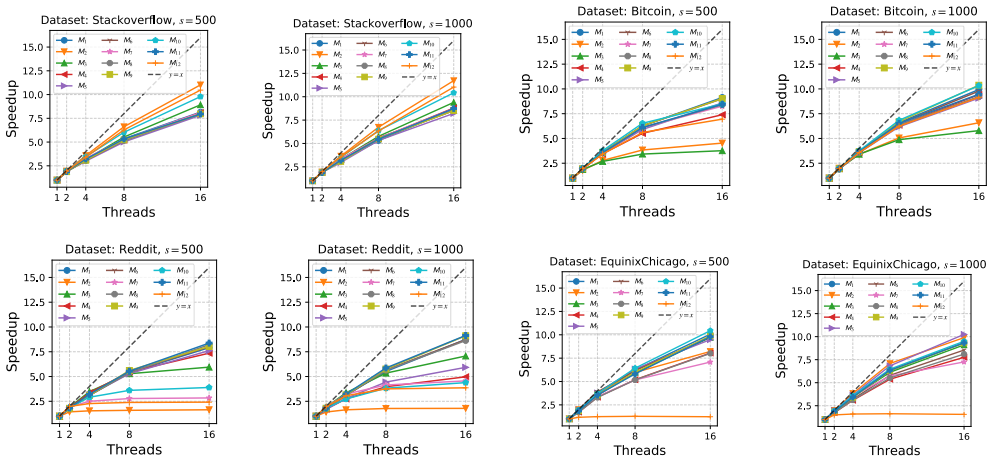


Figure 3.7: Speed-up factor achieved through the parallel implementation of PRESTO by varying the number of threads, In these experiments δ was fixed to the following values $\delta_{SO} = 86400, \delta_{BI} = 7200, \delta_{RE} = 14400, \delta_{EC} = 100000$. We show how the speedup varies across two values of $s \in \{500, 1000\}$.

As discussed in Section 3.5.1, PRESTO can be easily parallelized by executing the s iterations of Algorithm 3 in parallel, in this section we discuss such implementation. We implemented such strategy through a *thread pooling* design pattern. We present the results of the multithread version of PRESTO-A. Results for PRESTO-E are similar to the ones we will discuss.

To study how the speedup of PRESTO-A varies across different configurations we proceed as follows. Given a configuration of parameters (s, δ, τ), where τ is the number of threads, we repeat PRESTO-A on five independent runs on the given configuration. We then computed the speed-up T_1/T_τ where $T_\tau, \tau \in \{1, 2, 4, 8, 16\}$ is the average of the running time over the five runs (with fixed parameters) executing PRESTO-A with τ threads. In Figure 3.7 we fixed $c = 1.25$ and δ accordingly for each dataset as reported and we show how the speed-up changes by varying the sample size $s \in \{500, 1000\}$. We observe a nearly linear speed-up up to 8 threads, and a speed-up for 16 threads that is slightly less than linear for datasets SO, and BI, and for most motifs in EC but M_{12}, for dataset RE the speedup is worse than the other datasets and strongly dependent on the motifs. We observe that increasing the sample size often increases slightly the speedup, i.e., the speedups with $s = 500$ are slightly worse than those with $s = 1000$, we believe this to be related to the specific paradigm of parallelism we adopted. Overall the speedup achieved by our parallel algorithm is satisfactory except for very specific configurations.

All together, the results of our experimental evaluation show that PRESTO obtains up to hundreds of order of magnitude of speed-up over exact procedure on the tests we performed while providing accurate estimates through its parallel implementation (the sequential implementation is al-

ready faster more than ten times on some configurations, while its parallel implementation with 16 threads executes with a speedup of more than 10× over the sequential one), enabling the efficient analysis of billion edges networks.

3.7.7 Edge Timestamps Distributions – Skewed vs Uniform

In this section we compare the edge distributions, with respect to the timestamps of the datasets SO and RE used in our experimental evaluation in Section 3.6. To obtain such distribution we computed for each timestamp $t_i, i = 1, \ldots, m$ s.t. $\exists(u, v, t_i) \in E$, the number of edges in the forward window of length $c\delta$ starting from t_i, i.e., we computed $|\{(x, v, t) \in E | t \in [t_i, t_i + c\delta]\}|$ for each $t_i, i = 1, \ldots, m$, where we kept the parameters c and δ as in Table 3.5. This corresponds, for example, to the number of edges in each sample candidate of PRESTO-E. The results are shown in Figure 3.8.

We first discuss the distribution of the timestamps of the edges in the RE dataset (Figure 3.8 (left)), we observe immediately that such distribution is very skewed. In particular, the number of edges in the windows of length $c\delta$ starting from edges with timestamps in the first quarter of $[t_1, t_m]$, i.e., $t < 1.25 \cdot 10^9$, do not exceed 10^3 edges. While most of the temporal windows starting at timestamps in the middle interval of $[t_1, t_m]$, i.e., $1.25 \cdot 10^9 < t < 1.4 \cdot 10^9$, have much more edges ranging from 10^3 to 10^5 (several orders of magnitude edges more than windows from the first quarter of the network's timespan). In the last quarter of $[t_1, t_m]$ instead ($t > 1.4 \cdot 10^9$) the dataset presents very sparse edges with most of the windows (which are sample candidates in PRESTO) not exceeding 10^2 edges, note that the windows starting in the middle section of $[t_1, t_m]$ have at least one order of magnitude additional edges (i.e., $[10^3, 10^5]$ edges). This shows how the timestamps of the edges in RE have a very skewed distribution.

Figure 3.8 (right) instead shows the distribution of the timestamps of the edges on the dataset SO. As we can see, except for a very brief transient state, the timestamps are almost uniformly distributed on $[t_1, t_m]$, with the windows of length $c\delta$ centred at each timestamp $t_i, i = 1, \ldots, m$ having almost the same number of edges, or ranging in at most one order of magnitude of difference. Therefore the edges of E present a very uniform distribution of the timestamps in the dataset SO over the interval $[t_1, t_m]$.

We believe that such distributions may affect the results of the approximation of the different version of PRESTO as discussed in the main text.

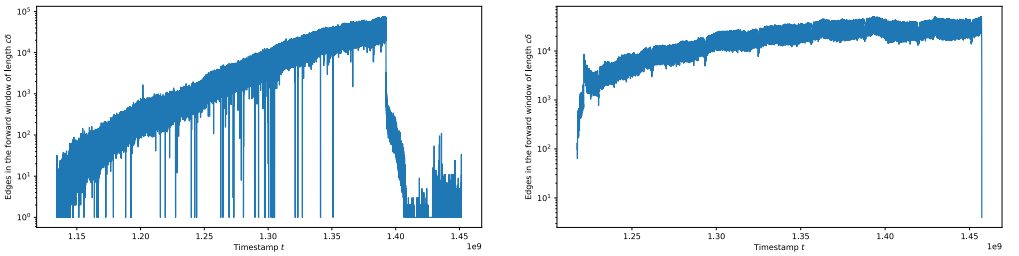


Figure 3.8: Left: distribution of the edges on dataset RE. Right: distribution of the edges on dataset SO. Both plot's y-axis are in *log* scale.

Chapter 4

Counting Multiple Temporal Motifs Simultaneously

In this chapter we discuss existing limitations of the temporal motif counting problem introduced in Chapter 2, to solve these issues we propose a novel counting problem based on obtaining the counts of multiple temporal motifs, sharing a common topology. We then propose ODEN, our algorithm to solve with high accuracy the estimation task defined on such novel counting problem. We then evaluate such algorithm to show how it outperforms existing state-of-the-art algorithms adapted to work on the novel problem.

4.1 Introduction

Networks are ubiquitous representations that model a wide range of real-world systems, such as social networks [Cho et al., 2011], citation networks [Ding, 2011], biological systems [Girvan and Newman, 2002], and many others [Newman, 2010]. One of the most fundamental primitives in network analysis is the mining of *motifs* [Shen-Orr et al., 2002, Milo, 2002, 2004] (or *graphlets* [Przulj, 2007, Bressan et al., 2019]), which requires to count the occurrences of small connected subgraphs of k nodes. Motifs represent key building blocks of networks, and they provide useful insights in wide range of applications such as network classification [Shervashidze et al., 2009, Milenković and Pržulj, 2008], network clustering [Baumes et al., 2005], and community detection [Batagelj and Zaversnik, 2003].

Modern networks contain rich information about their edges or nodes [Zong et al., 2015, Rossi et al., 2021, Kosyfaki et al., 2018, Ceccarello et al., 2017] in addition to graph structure. One of the most important information is the time at which the interactions, represented by edges, occur. Networks for which such information is available are called *temporal* [Holme and Saramäki, 2012, 2019]; novel insights about the underlying *dynamics* of the systems can be uncovered by the analysis of such networks [Kumar et al., 2006, Kovanen et al., 2013, Kumar and Calders, 2018]. In recent years, many primitives [Kovanen et al., 2011, Hulovatyy et al., 2015, Paranjape et al., 2017, Schwarze and Porter, 2020] have been proposed as counter-

part, in temporal networks, to the study of subgraph patterns for nontemporal networks, with each primitive capturing different temporal aspects of a network. One of the most important such primitives is the study of *temporal motifs* [Paranjape et al., 2017]. Temporal motifs are small connected subgraphs with k nodes and ℓ edges occurring with a prescribed order within a time interval of duration δ (as presented in Section 2.1.2.1). Temporal motifs describe the patterns shaping interactions over the network, e.g., networks from similar domains tend to have similar temporal motif counts [Paranjape et al., 2017], and their analysis is useful in many applications, e.g., anomalies detection [Belth et al., 2020], network classification [Tu et al., 2019], and social networks [Boekhout et al., 2019].

Temporal information poses several challenges in the analyses of motifs. A major challenge is represented by the large number of temporal motifs that can be build even with a limited number of vertices and edges. For example, even considering directed (and connected) temporal motifs with only 3 vertices and 3 edges, there are 32 such motifs. In several domains when motifs are studied in the exploratory analysis of a temporal network it is almost impossible for the data analyst to known *a priori* which motif is the most interesting and useful. In social networks, for example, a set of 3 vertices represents the smallest non trivial community, and different temporal motifs with 3 vertices describe different patterns of interactions in such community. Hence, studying all such motifs can provide novel insights on the interactions within such communities. In network classification, considering the counts of all the 32 motifs with 3 vertices and 3 edges lead to models with improved accuracy [Tu et al., 2019].

However, since state-of-the-art approaches for general temporal motifs only allow the analysis of one motif at the time (see Chapter 3), the user needs to *iteratively* select and analyze the various motifs, resulting in an inefficient and time consuming process, in particular for large networks.

In this paper, we define and study the problem of simultaneously counting the occurrences of various temporal motifs. In particular, we consider all motifs corresponding to the same *static* target template (e.g., all triangles - see Figure 4.1a). This problem is extremely challenging, since computing the count of even a single temporal motif is NP-Hard in general [Liu et al., 2019], with existing state-of-the-art approaches having complexity exponential in the number of edges of the motif to obtain even a single motif's count [Liu et al., 2019, Wang et al., 2020, Sarpe and Vandin, 2021a].

The task of counting temporal motifs is hindered by the sheer size of modern datasets and, therefore, scalable techniques are needed to deal with such amount of data as discussed in Chapter 1. Since exact approaches [Paranjape et al., 2017, Mackey et al., 2018, Gurukar et al., 2015] are impractical, rigorous and efficient approximation algorithms providing tight guarantees are needed. In this work we develop ODEN, a sampling algorithm that provides a high quality approximation for the problem of counting multiple temporal motifs with the same static topology. Our main contributions are as follows:

- We propose the *motif template counting problem*, where, given a temporal network, a k-node target template graph H, the number ℓ of edges of each temporal motif, and a bound δ on the duration of the temporal motifs, the problem requires to output all the counts of the temporal motifs whose static topology corresponds to H and having exactly ℓ temporal edges, occurring within δ-time.
- We propose ODEN, a randomized sampling algorithm providing a high quality approximation for the motif template counting problem. ODEN's approach is to sample a set of motif occurrences, ensuring that they all share the same static topology H. Thus, ODEN takes advantage of the constraint that all motifs must share a common target template H, aggregating the computation of all motif counts in a sample. ODEN's approximation, as in other data mining applications, is controlled by two parameters ε, η, which control respectively the quality and the confidence of the approximations.
- We show a tight and efficiently computable bound on the number of samples required by ODEN for the approximation to be within ε error with confidence $> 1 - \eta$ for *all* temporal motif counts.
- We perform large scale experiments using datasets with up to billions of temporal edges, showing that ODEN requires a fraction of the time required by state-of-the-art approximation algorithms for single motif counts, and that it reports sharper estimates. We then provide a parallel implementation of ODEN displaying almost linear speedup in many configurations. We also show how ODEN provides novel insights on the dynamics of a real-world temporal network.

4.2 Preliminaries

In this section we introduce the basic notions that we will use throughout the work, and we define the computational problem of counting multiple temporal motifs sharing a common target template graph. Recall the definition of temporal networks.

Definition 4.1. *A* temporal network *is a pair* $T = (V, E)$ *where,* $V = \{v_1, \ldots, v_n\}$ *and* $E = \{(x, y, t) : x, y \in V, x \neq y, t \in \mathbb{R}^+\}$ *with* $|V| = n$ *and* $|E| = m$.

Given $(x, y, t) \in E$, we say that t is the *timestamp* of the directed edge (x, y). Given a temporal network T, by ignoring the timestamps of its edges we obtain the associated *undirected projected static network*, defined as follows.

Definition 4.2. *The* undirected projected static network *of a temporal network* $T = (V, E)$ *is the pair* $G_T = (V, E_{G_T})$ *that is an undirected network, such that* $E_{G_T} = \{\{x, y\} : (x, y, t) \in E\}$.

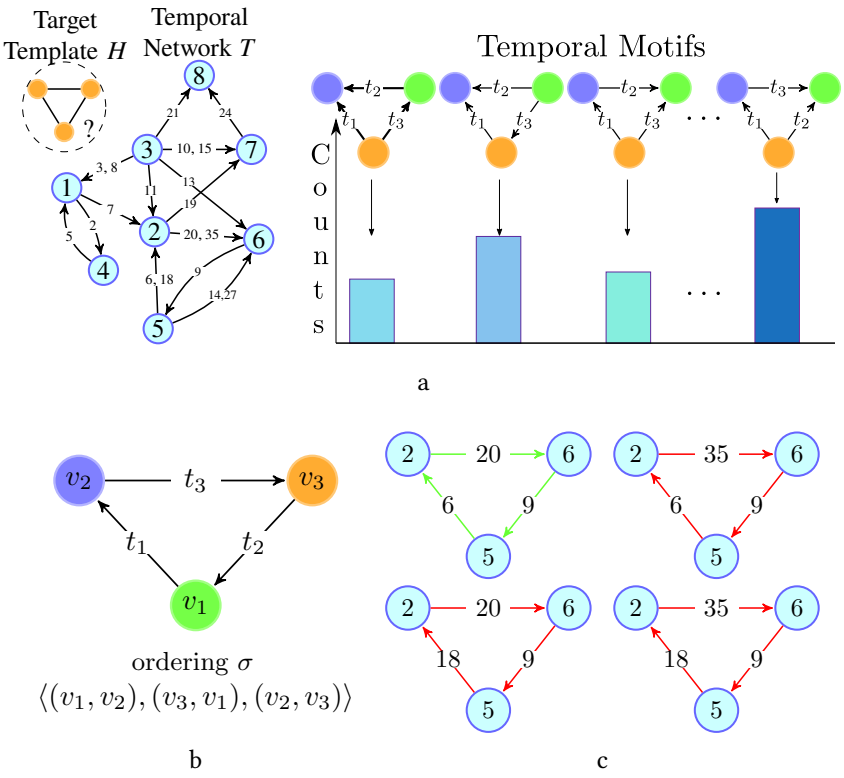


Figure 4.1: (4.1a): Motif template counting problem overview: given a temporal network and a (static) target template, compute the counts of all temporal motifs that map on the template. (4.1b): Temporal motif, with $k = 3, \ell = 3$, and its ordering σ. (4.1c): Sequences of edges of the network in (4.1a) among nodes $\{2, 5, 6\}$ that map topologically on the motif in (4.1b). For $\delta = 15$ only the green sequence is a δ-instance of the motif, since the timestamps respect σ and $t'_\ell - t'_1 = 20 - 6 \leq \delta$. The red sequences are not δ-instances, since they do not respect such constraint or do not respect the ordering σ.

We will often use the term *static network* to denote a network whose edges are without timestamps. Next we introduce the definition of *temporal motifs* as defined by Paranjape et al. [2017] (as seen in Section 2.1.2.1), which are small, connected subgraphs representing patterns of interest.

Definition 4.3. *A* k-node ℓ-edge temporal motif M *is a pair* $M = (\mathcal{K}, \sigma)$ *where* $\mathcal{K} = (V_\mathcal{K}, E_\mathcal{K})$ *is a directed and weakly connected* multigraph *where* $V_\mathcal{K} = \{v_1, \ldots, v_k\}$, $E_\mathcal{K} = \{(x, y) : x, y \in V_\mathcal{K}, x \neq y\}$ *s.t.* $|V_\mathcal{K}| = k$ *and* $|E_\mathcal{K}| = \ell$, *and* σ *is an ordering of* $E_\mathcal{K}$.

Recall that a k-node ℓ-edge temporal motif $M = (\mathcal{K}, \sigma)$ is also identified by the sequence $\langle (x_1, y_1), \ldots, (x_\ell, y_\ell) \rangle$ of edges ordered according to σ; we will often use such representation for a motif M (see Figure (4.1b) for an example). Given a k-node ℓ-edge temporal motif M, the values of k and ℓ are determined by $V_\mathcal{K}$ and $E_\mathcal{K}$. We will therefore use the term *temporal motif*, or simply *motif*, when k and ℓ are clear from context. Given a temporal

motif $M = ((V_{\mathcal{K}}, E_{\mathcal{K}}), \sigma)$, we denote with $G_u[M]$ the undirected graph corresponding to the underlying undirected graph structure of the multigraph $\mathcal{K}$ of M, that is $G_u[M] = (V_{\mathcal{K}}, E^u_M)$ where $E^u_M = \{\{x, y\} : (x, y) \vee (y, x) \in E_{\mathcal{K}}\}$ (i.e., E^u_M is the *set* of undirected edges associated to the multiset $E_{\mathcal{K}}$). Notice that directed edges of the form $(x, y), (y, x)$ as well as multiple directed edges $(x, y), (x, y), \ldots$ from $E_{\mathcal{K}}$ are represented by the same undirected edge $\{x, y\}$ in E^u_M.

For a fixed temporal motif M, we are interested in identifying its realizations in T appearing within at most δ-time *duration*, as captured by the definition of δ-instance of M (already introduced in Section 2.1.2.1).

Definition 4.4. *Given a temporal network $T = (V, E)$ and $\delta \in \mathbb{R}^+$, a time ordered sequence $S = \langle (x'_1, y'_1, t'_1), \ldots, (x'_\ell, y'_\ell, t'_\ell) \rangle$ of ℓ unique temporal edges from T is a* δ-instance *of the temporal motif $M = \langle (x_1, y_1), \ldots, (x_\ell, y_\ell) \rangle$ if:*

1. *there exists a bijection f on the vertices such that $f(x'_i) = x_i$ and $f(y'_i) = y_i$, $i = 1, \ldots, \ell$; and*

2. *the edges of S occur within δ time, i.e., $t'_\ell - t'_1 \leq \delta$.*

As for Chapter 3, the techniques we will develop can be extended to work under more general constraints for a δ-instance, e.g., considering partial orderings or edges with equal timestamps. Exploring different values of $\delta \in \mathbb{R}^+$ in Definition 4.4 often leads to different insights on the temporal network that may be discovered through the analysis of the motifs [Holme and Saramäki, 2012, Panzarasa et al., 2009, Kovanen et al., 2011, Bajardi et al., 2011]. Note that in a δ-instance of the temporal motif $M = (\mathcal{K}, \sigma)$ the edge timestamps must be sorted according to the ordering σ (see Figure (4.1c) for an example). In fact, σ plays a key role in defining a temporal motif, with different orderings of the same multigraph $\mathcal{K}$ reflecting diverse dynamic properties captured by the motif.

For a given directed multigraph $\mathcal{K}$ with $|E_{\mathcal{K}}| = \ell$ edges, in general not all the $\ell!$ orderings of its edges define *distinct* temporal motifs. We therefore introduce the following equivalence relation.

Definition 4.5. *Let M_1 and M_2 be two temporal motifs. Let $M_1 = \langle (x^1_1, y^1_1), \ldots, (x^1_\ell, y^1_\ell) \rangle$, and $M_2 = \langle (x^2_1, y^2_1), \ldots, (x^2_\ell, y^2_\ell) \rangle$ be the sequences of edges of M_1 and M_2, respectively. We say that M_1 and M_2 are* not distinct *(denoted with $M_1 \cong_\tau M_2$) if there exists a bijection g on the vertices such that $g(x^1_i) = x^2_i$ and $g(y^1_i) = y^2_i$, $i = 1, \ldots, \ell$.*

We provide an example of the definition above in Figure 4.2.

Given two networks (undirected or temporal) G, G' we say that $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ (denoted with $G' \subseteq G$) if $V' \subseteq V$ and $E' \subseteq E$. Note that we require a subgraph to be *edge* induced (as presented in Section 2.1.2). To conclude the preliminary notions, we recall the definition of *static graph isomorphism* (seen in Section 2.1.1).

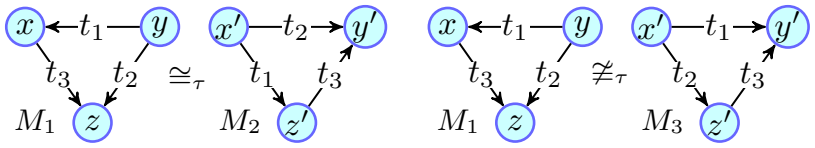


Figure 4.2: (Left): The two motifs are *not distinct*: let $\sigma_1 = \langle (y, x), (y, z), (x, z) \rangle$ and $\sigma_2 = \langle (x', z'), (x', y'), (z', y') \rangle$ corresponding to M_1 and M_2, then the function $f : V^1_{\mathcal{K}} \mapsto V^2_{\mathcal{K}}$ defined by $f(x) = z', f(y) = x', f(z) = y'$ preserves both the topology and the ordering as from Definition 4.5. (Right): The two motifs are *distinct* since there is no map $f : V^1_{\mathcal{K}} \mapsto V^3_{\mathcal{K}}$ preserving both topology and ordering.

Definition 4.6. *Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ we say that the two graphs are* isomorphic*, denoted with $G \sim H$ if and only if there exists a bijection $f : V_G \mapsto V_H$ on the vertices such that $e = (u, v) \in E_G \Leftrightarrow e' = (f(u), f(v)) \in E_H$.*

Let $\mathcal{U}(M, \delta) = \{I : I \text{ is a } \delta\text{-instance of } M\}$ be the *set of (all) δ-instances* of the motif M in T. The *count* of M is $C_M(\delta) = |\mathcal{U}(M, \delta)|$, denoted with C_M when δ is clear from the context.

Given a static undirected graph H, which we call the *target template*, we are interested in solving the problem of computing the number of δ-instances of all temporal motifs with ℓ edges and all corresponding to the same static graph H. More formally, given the *target template* $H = (V_H, E_H)$, which is a simple and connected graph, and $\ell \geq |E_H| \in \mathbb{Z}_+$, let $\mathcal{M}(H, \ell)$ be the set of *distinct* temporal motifs with ℓ edges whose underlying undirected graph structure corresponds to H, that is $\mathcal{M}(H, \ell)$ contains motifs $M_i = ((V^i_{\mathcal{K}}, E^i_{\mathcal{K}}), \sigma_i)$, $i = 1, 2, \ldots$, such that:

1. $G_u[M_i] \sim H$;
2. $|E^i_{\mathcal{K}}| = \ell$;
3. $M_i \ncong_\tau M_j, \forall j \neq i$.

Let us explain intuitively the constrains above. First, H imposes a constraint on the *undirected* static topology the temporal motifs of interest (that are directed subgraphs) should have. That is, it requires all the motifs to have the same underlying graph structure ($G_u[M]$), which must be isomorphic to H. This is a useful way to represent multiple related temporal motifs. For example, in social network analysis by fixing H as an undirected triangle we consider in $\mathcal{M}(H, \ell)$ all temporal motifs that characterize the communication between groups of three friends (i.e., each motif will represent a *different* form of communication among all such groups [Paranjape et al., 2017]). The second constraint requires each motif $M_i \in \mathcal{M}(H, \ell)$ to have *exactly* $\ell \geq |E_H|$ edges, with ℓ provided in input by the user. Fixing the parameter ℓ is motivated by the fact that motifs with different values of ℓ (even with the same target template structure H) reflect different patterns of

interaction (e.g, a group of friends that exchanges $\ell = 3$ or $\ell = 4$ messages). As we will show empirically in Section 4.5.4, such counts vary significantly with ℓ for fixed H and δ. Finally, the third constraint ensures that we only count distinct motifs, i.e., motifs representing different patterns.

We now define the motif template counting problem.

Problem 4.1. ***Motif template counting problem.*** *Given a temporal network T, a static* undirected *target graph $H = (V_H, E_H)$, $\ell \in \mathbb{Z}_+, \ell \geq |E_H|$, and a parameter $\delta \in \mathbb{R}_+$, find the counts $C_{M_i}(\delta)$ of motifs $M_i \in \mathcal{M}(H, \ell), i = 1, \ldots, |\mathcal{M}(H, \ell)|$ in T.*

We now provide an example of the different motifs to be counted for different values of ℓ with a fixed target template H.

Example 4.1. *Let $H = (\{v_1, v_2\}, \{\{v_1, v_2\}\})$, that is, the target template is an edge. Let $e_1 = (v_1, v_2)$ and $e_2 = (v_2, v_1)$. By varying $\ell \in \{2, 3\}$ the motifs in $\mathcal{M}(H, \ell)$, for which we need to compute the counts, are: $M_1 = \langle e_1, e_1 \rangle$ and $M_2 = \langle e_1, e_2 \rangle$ for $\ell = 2$ (i.e., $|\mathcal{M}(H, 2)| = 2$) while $M_1 = \langle e_1, e_1, e_1 \rangle, M_2 = \langle e_1, e_2, e_1 \rangle, M_3 = \langle e_1, e_2, e_2 \rangle, M_4 = \langle e_1, e_1, e_2 \rangle$ for $\ell = 3$ (i.e., $|\mathcal{M}(H, 3)| = 4$).*

Since solving the counting problem *exactly* is NP-Hard in general[1] even for one single temporal motif, we aim at providing high-quality approximations to the motif counts as follows.

Problem 4.2. ***Motif template approximation problem.*** *Given the input parameters of Problem 4.1 and additional parameters $\varepsilon \in \mathbb{R}_+, \eta \in (0, 1)$, compute approximations $C'_{M_i}(\delta)$ of counts $C_{M_i}(\delta)$ of motifs $M_i \in \mathcal{M}(H, \ell), i = 1, \ldots, |\mathcal{M}(H, \ell)|$, such that $\mathbb{P}[\exists i \in \{1, \ldots, |\mathcal{M}(H, \ell)|\} : |C'_{M_i}(\delta) - C_{M_i}(\delta)| \geq \varepsilon C_{M_i}(\delta)] \leq \eta$, that is $C'_{M_i}(\delta)$ is a relative ε-approximation to the count $C_{M_i}(\delta)$ with probability $\geq 1 - \eta$ for all $i = 1, \ldots, |\mathcal{M}(H, \ell)|$ simultaneously.*

Note that the above guarantees are fully characterized by the supremum deviation of the estimators $C'_{M_i}(\delta)$ if these estimates are unbiased as discussed in Section 2.3.2.

4.3 Related Works

Much work has been done on enumerating and approximating k-node motifs in (nontemporal) networks. We refer the interested reader to the surveys by Ribeiro et al. [2019], Yu et al. [2020]. However, such works cannot be easily adapted to temporal motifs since they do not properly account for the temporal information [Paranjape et al., 2017, Holme and Saramäki, 2012]. Many different definitions of temporal networks and temporal patterns have been proposed: here we will focus only on those works that are

[1] The hardness depends on the topology of the motif. For example for triangles and single edges there exist polynomial time-algorithms, even if they are impracticable on very large networks. Interestingly, counting temporal star-shaped motifs is NP-Hard [Liu et al., 2019], while on static networks such motifs can be counted in polynomial time.

relevant for our work, the interested reader may refer to Masuda and Lambiotte [2016], Holme and Saramäki [2019, 2012], Jazayeri and Yang [2020] for a more general overview.

Our work builds on the work of Paranjape et al. [2017] which first introduced the definition of temporal motif used here, and the problem of counting single temporal motifs. The authors provided a general algorithm for counting a single temporal motif by enumerating all the subsequences of edges that map on a single static subgraph. Their approach is not feasible on large datasets since it requires exhaustive enumeration of *all* subgraphs of the undirected projected static network G_T that are isomorphic to the target template H. The authors also proposed efficient algorithms and data-structures for counting 3-node 3-edge motifs, which may be used for the exact counting subroutines within ODEN sampling framework. In addition to the algorithmic contributions, the authors also showed that networks from similar domains tend to exhibit similar temporal motif counts. They also showed how motif counts can provide significant insights on the communication patterns in many networks, highlighting the importance of studying temporal motifs in temporal networks.

Other *exact* algorithms have been proposed for the problem of counting a single motif, or for slightly different problems. Mackey et al. [2018] presented a backtracking algorithm for counting a single temporal motif that can be use for any motif. Boekhout et al. [2019] developed exact algorithms for counting temporal motifs in multilayer temporal networks (i.e., each edge is a tuple (x, y, t, a) with a denoting the layer of each edge), they also discuss efficient data-structures for counting 4-node 4-edge motifs, which may also be adapted for the exact counting subroutines in our sampling framework ODEN. Being exact, both such algorithms do not scale on massive datasets due large time and memory requirements.

Several approximation algorithms have been proposed in recent years for estimating the count of a *single* motif (as already discussed in Chapter 3). Liu et al. [2019] proposed a temporal-partition based sampling approach. Wang et al. [2020] introduced a sampling-based algorithm that selects temporal edges with a fixed probability specified by the user. Lastly, Sarpe and Vandin [2021a] proposed PRESTO, an algorithm based on sampling small windows of the temporal network T. All such algorithms can be used to analyze a single temporal motif but become inefficient as the number of motifs to be counted grows, such as in Problem 4.2. In fact, they cannot leverage the additional information that all motifs $M_1, \ldots, M_{|\mathcal{M}(H,\ell)|}$ must share a common static topology isomorphic to H. As stated in Section 4.1, when analysing a temporal networks it is hard to know *a-priori* which motif is representing important functions for the network, therefore it is common to test all possible orderings σ over one fixed target template H for fixed ℓ, δ [Paranjape et al., 2017, Tu et al., 2019] (as in Problem 4.1) resulting in a time consuming and inefficient procedure. Our approach instead supports the direct analysis of *multiple* temporal motifs, enabling the study of hundreds of temporal motifs on massive networks in a very limited time.

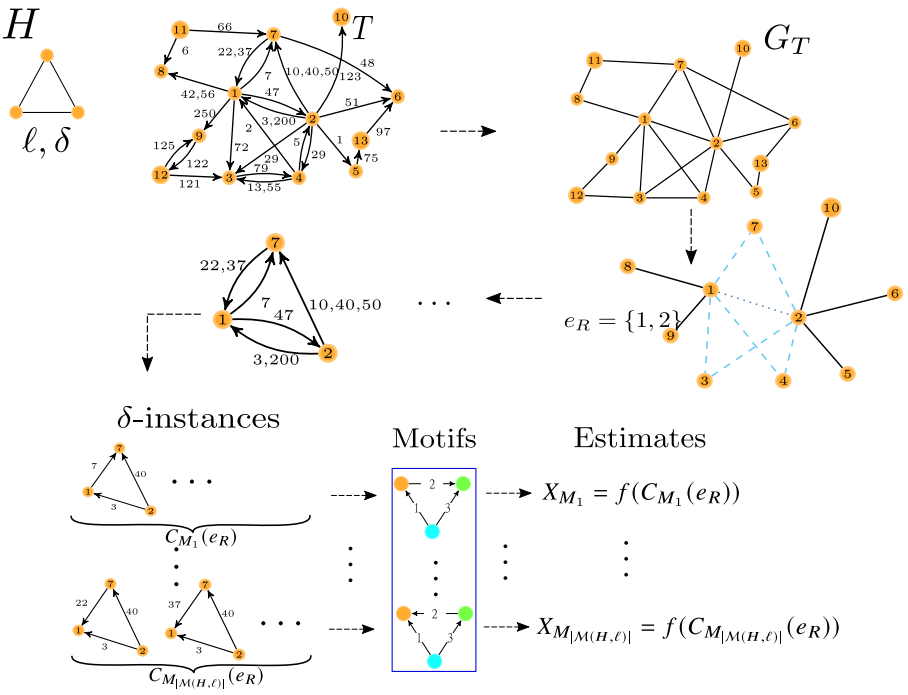


Figure 4.3: Overview of ODEN's approximation strategy. Let H be a triangle, and $\ell = 3, \delta = 40$. ODEN first collects the static projected network G_T, then samples an edge $e_R \in G_T$ randomly ($e_R = \{1, 2\}$ in the figure) and enumerates all the subgraphs of G_T isomorphic to H containing e_R. For each subgraph it collects the corresponding temporal network, counts the δ-instances of the motifs, and combines the different counts to obtain unbiased estimates of motif counts. This procedure is repeated to obtain concentrated estimates.

4.4 ODEN

In this section we present ODEN, our algorithm to address the motif template approximation problem (Problem 4.2). We start in Section 4.4.1 with an overview of ODEN. We then describe the algorithm in Section 4.4.2, analyze its time complexity in Section 4.4.3 and its theoretical guarantees, including an efficiently computable bound on the number of samples required to obtain the desired probabilistic guarantees, in Section 4.4.4.

4.4.1 Overview of ODEN

Our algorithm ODEN estimates of the counts of motifs in $\mathcal{M}(H, \ell)$. The main idea is to avoid the explicit generation all the motifs $M_i \in \mathcal{M}(H, \ell), i = 1, \ldots, |\mathcal{M}(H, \ell)|$ to count them one at the time as it is required by existing algorithms that approximate a single motif count. ODEN instead leverages the fact that the topology of all motifs must to be isomorphic to the target template H, by reusing the computation while estimating the motif counts.

An overview of the main strategy adopted by our algorithm is presented in Figure 4.3. Given the input parameters of Problem 4.2, where H is the

target template, the idea behind our procedure is to consider the undirected static projected graph G_T of the input temporal network T and proceed as follows: i) find a set of subgraphs in the static graph G_T that are isomorphic to H by first *sampling* an edge e_R of G_T with some probability p_{e_R}, where p_{e_R} depends, potentially, on e_R and the temporal network T, and then enumerating all subgraphs of G_T isomorphic to H and containing e_R; ii) for each such subgraph, consider the corresponding temporal subgraph and compute all the counts of the subsequences of ℓ edges occurring within δ-time in such temporal subgraph; iii) for each such subsequence identified, find the corresponding motif in $\mathcal{M}(H, \ell)$, for which the subsequence is a δ-instance of, and update a count for each motif identified; iv) weight each motif count opportunely in order to maintain an unbiased estimate of global motif counts; v) repeat steps i)-iv) a sufficient number of iterations to guarantee the desired (ε, η)-approximation (see Problem 4.2).

4.4.2 Algorithm Description

ODEN is described in Algorithm 4. It first computes $G_T = (V, E_{G_T})$, the undirected projected static graph of T (line 1), and initializes $C_{estimates}$ (line 2) used to store the estimates of motif counts, which are used to compute the estimators $C'_{M_i}, i = 1, \ldots, |\mathcal{M}(H, \ell)|$. Then it repeats s times (line 3) the following procedure: i) pick a random edge e_R from G_T (line 4) according to some probability distribution over the edges of E_{G_T}; ii) enumerate all the subgraphs h of G_T such that $h \sim H$ and $e_R \in h$ (line 5); note that this enumeration step is local to e_R; iii) for each such h (line 6), collect the corresponding temporal graph, i.e., all edges in T for which their static projected edge is an edge of h (line 7), sort the sequence of edges of such graph by increasing timestamps and apply some pruning criteria (lines 8-9); iv) if the sequence is not pruned, then update the estimates of the number of δ-instances of each temporal motif by calling the routine `FastUpdate` (line 10). `FastUpdate` features an efficient implementation of the general algorithm by Paranjape et al. [2017], for which we devised efficient encodings of the motifs within integers through bitwise operations. Such function updates $C_{estimates}$ in order to maintain for each motif the count that will be used to output its unbiased estimate (see Section 4.6.1). Let $C_{M_i}(e)$ be the number of δ-instances in T of $M_i, i = 1, \ldots, |\mathcal{M}(H, \ell)|$ whose undirected projected static network contains edge $e \in G_T$. `FastUpdate` updates the estimate of the counts for each motif M_i by summing its unbiased estimate obtained at the j-th iteration (i.e., $X^j_{M_i} = C_{M_i}(e_R)/(|E_H| p_{e_R})$). Once the procedure is repeated s times, for each motif $M_i \in \mathcal{M}(H, \ell), i = 1, \ldots, |\mathcal{M}(H, \ell)|$, ODEN computes the final estimate $C'_{M_i} = \frac{1}{s} \sum_{j=1}^{s} X^j_{M_i}$ where $X^j_{M_i} = \frac{1}{|E_H|} \sum_{e \in G_T} C_{M_i}(e) X_e / p_e$ is the estimate obtained at the j-th iteration (with X_e being a bernoulli random variable denoting if edge $e \in G_T$ is sampled at the j-th iteration, s.t. $\mathbb{P}[X_e = 1] = p_e$) and outputs it together with the motif (we output σ_i over the node-set V_H) (lines 12-13). We show in Lemma 4.1 that ODEN outputs unbiased estimates for all the motif counts.

Algorithm 4: ODEN

Input: $T = (V, E), H = (V_H, E_H), \delta, s, \ell$
Output: $(M_i, C'_{M_i}), i = 1, \ldots, |\mathcal{M}(H, \ell)|$ where C'_{M_i} is an estimate of C_{M_i} for the motifs in $\mathcal{M}(H, \ell)$.

1 $G_T = (V, E_{G_T}) \leftarrow$ `UndirectedStaticProjection`(T)
2 $C_{estimates} \leftarrow \{\}$
3 **for** $j \leftarrow 1$ *to* s **do**
4 $e_R = \{x_R, y_R\} \leftarrow$ `RandomEdge`$(p(e) : e \in E_{G_T})$
5 $\mathcal{H} \leftarrow \{h \subseteq G_T : h \sim H, \{x_R, y_R\} \in h\}$
6 **foreach** $h \in \mathcal{H}$ **do**
7 $S \leftarrow \{(x, y, t), (y, x, t) \in E : \{x, y\} \in h\}$
8 `SortInPlace`(S) ▷ `By increasing timestamps`
9 **if** *`Pruning criteria are not met`* **then**
10 `FastUpdate`$(\delta, S, C_{estimates}, p(e_R), H)$
11 **foreach** $(M, X_M) \in C_{estimates}$ **do**
12 $C'_M \leftarrow \frac{X_M}{s}$
13 **output** (M, C'_M)

We briefly discuss the pruning criteria used in line 9. Given a candidate temporal graph S for which $G_S \sim H$ holds, we check in linear time if S can contain a δ-instance of a motif or not: since S is already sorted by increasing timestamps (see line 8), we efficiently check if there are at least ℓ edges within δ-time. If not, then we prune the sequence (since by definition a δ-instance of a motif with k-nodes, and ℓ-edges must have ℓ edges occurring within δ-time). We thus avoid calling the subroutine `FastUpdate`, which has an exponential complexity in general (see Section 4.4.3), on S.

We now discuss the probability distribution used to sample a random edge e_R from G_T (line 4), while we describe the subroutine `FastUpdate` that updates the motif estimates at each iteration (line 10) and the algorithms employed for the static enumeration in Section 4.6.1.

Since our final estimate is an average over s samples of the variables $X^j_{M_i}, i = 1, \ldots, |\mathcal{M}(H, \ell)|, j = 1, \ldots, s$, and given that $X^j_{M_i}$ is an unbiased estimate (see Lemma 4.1) the final estimate is also a consistent estimator (i.e., it converges to C_{M_i} as $s \to \infty$) if each edge has a positive probability of being sampled[2]. Thus *any* probability mass assigning positive probabilities on edges can be adopted. We considered different distributions over the edges of E_{G_T}:

1. *Uniform*: $p_e = 1/|E_{G_T}|, e \in E_{G_T}$;
2. *Static degree based*: $p_e = d(e)/(\sum_{e' \in E_{G_T}} d(e')), e \in E_{G_T}$ where $d(e = \{x, y\}) = d(x) + d(y)$ is the degree of the edge as sum of the degree of

[2]More formally it is only necessary to assign to each δ-instance a known positive sampling probability.

its nodes $x, y \in V$ in G_T;

3. *Temporal degree based*: $p_e = \phi(e)/(\sum_{e' \in E_{G_T}} \phi(e'))$ with $\phi(e = \{x, y\}) = |\{t : \exists(x, z, t) \vee (z, x, t) \in E\}| + |\{t : \exists(z, y, t) \vee (y, z, t) \in E, z \neq x\}|, e \in E_{G_T}$;

4. *Temporal edge weight based*: $p_{e=\{x,y\}} = |\{(x, y, t), (y, x, t) \in E\}|/m, e \in E_{G_T}$;

We empirically found the distribution (4) to be the fastest to converge for small number s of iterations, thus we use it in our analysis. We observe that many other candidate distributions can be designed (e.g., combining two of those already listed with weights $\xi, 1 - \xi, \xi \in (0, 1)$) making our framework extremely versatile.

We conclude by summarizing some nice properties of our algorithm: 1) it computes the estimates only for the temporal motifs occurring in the input temporal network T (except for the very unpractical case where the motifs in $\mathcal{M}(H, \ell)$ have all zero counts) without generating all the possible candidates, while existing sampling techniques require to first generate all the candidates and then to execute the algorithms on such candidates, even for motifs with zero counts; 2) it takes advantage of the constraint that all motifs share the same underlying topology (H), saving computation when estimating the different counts; 3) it is trivially parallelizable: all the s iterations can be executed in parallel; 4) it can easily use most of the fast state-of-the-art subgraph enumeration algorithms developed for the exact subgraph isomorphism problem (see Section 4.6.1).

4.4.3 Time Complexity

In this section we briefly describe the time complexity of ODEN. ODEN needs to compute the probabilities $p(e)$ of edges in advance, which requires a $O(|E_{G_T}|)$ preprocessing step. Interestingly, this step does not depend on the target template H, so it can be reused for different target templates H. One of the most expensive steps in Algorithm 4 is the local enumeration to identify the set $\mathcal{H}$ which in general requires exponential time (line 5). For specific topologies this step can be implemented very efficiently with symmetry breaking conditions and min-degree expansion. For example, if H is a triangle this "local" enumeration to $e_R = \{x_R, y_R\}$ can be done in $O(\min(d_{x_R}, d_{y_R}))$ time. Let $|\mathcal{H}^*|$ be the maximum cardinality of a set of subgraphs isomorphic to H and adjacent to an edge in G_T. Let $|S^*|$ denote the maximum cardinality of a set S collected (in line 7) by our algorithm ODEN. Sorting S^* requires $O(|S^*| \log |S^*|)$ time. The subroutine `FastCount` has a complexity dominated by $O((|S^*| + \ell)|E_H|^\ell)$ (see [Paranjape et al., 2017] and Section 4.6.1.1 for more details). So overall the complexity of our procedure is $O(|E_{G_T}| + s(\zeta_{enum} + |\mathcal{H}^*|(|S^*| \log(|S^*|) + |E_H|^\ell(|S^*| + \ell))))$, where ζ_{enum} is the time required by the static enumerator used as subroutine to compute the set $\mathcal{H}^*$. Such step in general is exponential in the number of edges

of $|E_{G_T}|$ and depends on the exact technique used as subroutine. The final complexity accounts for the cycle (in line 3) that is repeated s times. The parallel version of our algorithm, which executes the cycle of line 3 in parallel on ω processing units available, leads to a time complexity of $O(|E_{G_T}| + s/\omega(\zeta_{enum} + |\mathcal{H}^*|(|S^*|\log(|S^*|) + |E_H|^\ell(|S^*| + \ell))))$.

4.4.4 Theoretical Guarantees

In this section we present the theoretical guarantees provided by ODEN.

Recall that our algorithm outputs, for each motif $M_i \in \mathcal{M}(H,\ell), i = 1,\ldots,|\mathcal{M}(H,\ell)|$, the following estimate: $C'_{M_i} = \frac{1}{s}\sum_{j=1}^{s} X^j_{M_i} = \frac{1}{s|E_H|}\sum_{j=1}^{s}\sum_{e\in G_T} C_M(e)X_e/p_e$. Note that the estimates provided by ODEN can be framed in the general framework we introduced in Section 2.2, in particular $\mathcal{X}$ is identified with the set of edges E_{G_T}, and each estimate $X_{M_i} = \hat{X}_f(x)$ hence each function in $\mathcal{F}$ is associated to a temporal motif in $\mathcal{M}(H,\ell)$, note also that X_{M_i} depends on the sampling distribution ρ, i.e., $\rho_e = p_e, e \in E_{G_T}$ as already described.

The following shows that such estimates are unbiased estimates of $C_{M_i}, i = 1,\ldots,|\mathcal{M}(H,\ell)|$. First we recall that $C_{M_i}(e)$ the number of δ-instances of motif $M_i, i = 1,\ldots,|\mathcal{M}(H,\ell)|$ from T whose undirected projected static network contains edge $e \in G_T$, i.e., $C_{M_i}(e) = \sum_{h\subseteq G_T, h\sim H: e\in h} |\mathcal{U}(h,M_i)|, e \in G_T$ where $\mathcal{U}(h,M_i)$ is the set of δ-instances of motif M_i whose static projected graph is $h \subseteq G_T$. Then based on the above it is simple to notice that the following formula holds for each motif $M_i, i = 1,\ldots,|\mathcal{M}(H,\ell)|$: $\sum_{e\in G_T} C_{M_i}(e) = |E_H|C_M$. This relation will be the key for proving the unbiasedness of the estimates provided by ODEN, as we show next.

Lemma 4.1. *For each motif-count pair (M_i, C'_{M_i}) reported in output by ODEN, C'_{M_i} is an unbiased estimate to C_{M_i}, that is $\mathbb{E}[C'_{M_i}] = C_{M_i}$*

Proof. First let us consider the expectation of $X^j_{M_i}, i = 1,\ldots,|\mathcal{M}(H,\ell)|, j = 1,\ldots,s$:

$$\mathbb{E}\left[\frac{1}{|E_H|}\sum_{e\in G_T}\frac{C_{M_i}(e)X_e}{p_e}\right] = \frac{1}{|E_H|}\sum_{e\in G_T}\frac{C_{M_i}(e)\mathbb{E}[X_e]}{p_e} = C_{M_i}$$

where we used the linearity of expectation and the facts that $\mathbb{E}[X_e] = p_e, e \in G_T$, and $\sum_{e\in G_T} C_{M_i}(e) = |E_H|C_{M_i}$; thus $X^j_{M_i}, i = 1,\ldots,|\mathcal{M}(H,\ell)|, j = 1,\ldots,s$ are unbiased estimates of C_{M_i}, combining such result to C'_{M_i} we obtain,

$$\mathbb{E}[C'_{M_i}] = \mathbb{E}\left[\frac{1}{s}\sum_{j=1}^{s} X^j_{M_i}\right] = \frac{1}{s}\sum_{j=1}^{s}\mathbb{E}[X^j_{M_i}] = \frac{sC_{M_i}}{s} = C_{M_i}$$

by the linearity of expectation. □

Let $\alpha = \min_{\{x,y\} \in E_{G_T}} \{|\{(x,y,t),(y,x,t) \in E\}|\}$, i.e., the minimum number of temporal edges of T that map on an edge in G_T. We now give an upper bound to the variance of the estimates provided by Algorithm 4 for each motif reported in output.

Lemma 4.2. *For each motif-count pair (M_i, C'_{M_i}) reported in output by* ODEN*, it holds* $Var[C'_{M_i}] \leq \frac{C^2_{M_i}}{s}\left(\frac{m}{\alpha|E_H|} - 1\right)$.

Proof. We need to bound the variance of the estimate C'_{M_i}, first we rewrite the estimator

$$C'_{M_i} = \frac{1}{s}\sum_{j=1}^{s} \frac{1}{|E_H|} \sum_{e \in G_T} C_{M_i}(e)\frac{X_e}{p_e} = \frac{1}{s}\sum_{j=1}^{s} X^j_{M_i}$$

Since the s variables $X^j_{M_i}, j \in [1,s]$ are independent (edges are drawn independently at each iteration of the outer `for` loop in Algorithm 4), it holds $\text{var}(C'_{M_i}) = \text{var}(\frac{1}{s}\sum_{j=1}^{s} X_{M_i}) = \frac{1}{s}\text{var}(X_{M_i})$ we thus only need to compute the variance of the variable X_{M_i}. Let us recall $\text{var}(X_{M_i}) = \mathbb{E}[X^2_{M_i}] - \mathbb{E}[X_{M_i}]^2 = \mathbb{E}[X^2_{M_i}] - C^2_{M_i}$ by the previous lemma. We will now bound $\mathbb{E}[X^2_{M_i}]$.

$$\mathbb{E}[X^2_{M_i}] = \mathbb{E}\left[\frac{1}{|E_H|^2}\sum_{e_1 \in G_T}\sum_{e_2 \in G_T} C_{M_i}(e_1)C_{M_i}(e_2)\frac{X_{e_1}X_{e_2}}{p_{e_1}p_{e_2}}\right]$$

$$= \frac{1}{|E_H|^2}\sum_{e_2 \in G_T} C^2_{M_i}(e_2)\frac{1}{p_{e_2}} \leq \frac{1}{|E_H|^2}\sum_{e_2 \in G_T} C^2_{M_i}(e_2)\frac{m}{\alpha} =$$

$$= \frac{m}{\alpha|E_H|^2}\sum_{e_2 \in G_T} C^2_{M_i}(e_2) \overset{(1.)}{\leq} \frac{m}{\alpha|E_H|^2}|E_H|C^2_{M_i} = \frac{mC^2_{M_i}}{\alpha|E_H|}$$

where we used the linearity of expectations, the fact that $\mathbb{E}[X_{e_1}X_{e_2}] = p_{e_1}$ only for $e_1 = e_2$ otherwise is 0, a bound on the minimum probability p_e where $p_e \leq \alpha/m, \forall e \in G_T$ for α defined as in Section 4.4.4. In (1.) we used the fact that $C_{M_i}(e) = \lambda_e C_{M_i}, e \in G_T, \lambda_e \in [0,1]$, then $\sum_{e_2 \in G_T} C^2_{M_i}(e_2) = \sum_{e_2 \in G_T} \lambda^2_{e_2} C^2_{M_i} \leq C^2_{M_i}\sum_{e_2 \in G_T}\lambda_{e_2} = |E_H|C^2_{M_i}$ since $\lambda_{e_2} \in [0,1]$ and further $\sum_{e \in G_T}\lambda_e = |E_H|$ by $\sum_{e \in G_T}\lambda_e C_{M_i} = |E_H|C_{M_i}$.

Thus the variance of X_{M_i} is bounded by:

$$\text{Var}(X_{M_i}) \leq \frac{mC^2_{M_i}}{\alpha|E_H|} - C^2_{M_i} = C^2_{M_i}\left(\frac{m}{\alpha|E_H|} - 1\right)$$

combining everything together we obtain that $\text{var}(C'_{M_i}) \leq \frac{C^2_{M_i}}{s}\left(\frac{m}{\alpha|E_H|} - 1\right)$, concluding the proof. □

To give a bound on the number s of samples required by ODEN to output a ε-approximation that holds on all motifs in output with probability $> 1 - \eta$, we combine Bennett's inequality [Bennett, 1962], an advanced result on the concentration of sums for independent random variables as reported in Chapter 3, with a union bound, obtaining the following main result.

Theorem 4.1. *Let s be the number of iterations of* ODEN*, let $\varepsilon \in \mathbb{R}^+$, and $\eta \in (0,1)$. If $s \geq \left(\frac{m}{\alpha|E_H|} - 1\right) \frac{1}{(1+\varepsilon)\ln(1+\varepsilon)-\varepsilon} \ln\left(\frac{2|\mathcal{M}(H,\ell)|}{\eta}\right)$ then*

$$\mathbb{P}[\exists i \in \{1,\dots,|\mathcal{M}(H,\ell)|\} : |C'_{M_i} - C_{M_i}| \geq \varepsilon C_{M_i}] \leq \eta.$$

Proof of Theorem 4.1. Let us fix $M_i, i \in [1, |\mathcal{M}(H,\ell)|]$ we first show a bound to the following probability $\mathbb{P}[|C'_{M_i} - C_{M_i}| \geq \varepsilon C_{M_i}]$. We want to derive such bound through the application of Bennett's inequality to the following summation: $\frac{1}{s}\sum_{j=1}^{s} X^j_{M_i}$, we already know that $\mathbb{E}[X^j_{M_i}] = C_{M_i}$ and $\mathbb{E}[(X^j_{M_i} - C_{M_i})^2] \leq C^2_{M_i}\left(\frac{m}{\alpha|E_H|} - 1\right) = \hat{v}^2_j$ for $j = 1,\dots,s$ it holds:

$$X^j_{M_i} = \frac{1}{|E_H|}\sum_{e\in G_T} C_{M_i}(e)\frac{X_e}{p_e} \leq \frac{1}{|E_H|}\sum_{e\in G_T} C_{M_i}(e)\frac{m}{\alpha} = \frac{mC_{M_i}}{\alpha|E_H|}$$

As proved in Section 3.7.1.2 Bennett's inequality holds even if we only have an upper bound on the variance of the estimates. Therefore let us compute the quantities to apply Bennett's bound (see Chapters 2, 3 for the statement), clearly $B = C_{M_i}(\frac{m}{\alpha|E_H|} - 1)$ combining what we already showed with the unbiasedness of $X^j_{M_i}$, moreover $v \leq \hat{v}^2_j$ since the bound $\hat{v}^2_j$ is equal for each $j \in [1,s]$. Then,

$$\frac{\hat{v}^2_j}{B^2} = \frac{C^2_{M_i}\left(\frac{m}{\alpha|E_H|} - 1\right)}{C^2_{M_i}(\frac{m}{\alpha|E_H|} - 1)^2} = \frac{1}{(\frac{m}{\alpha|E_H|} - 1)}$$

also

$$\frac{tB}{\hat{v}^2_j} = \frac{\varepsilon C_{M_i} C_{M_i}(\frac{m}{\alpha|E_H|} - 1)}{C^2_{M_i}\left(\frac{m}{\alpha|E_H|} - 1\right)} = \varepsilon$$

Combining everything together by Bennett's inequality we obtain,

$$\mathbb{P}\left(\left|\frac{1}{s}\sum_{j=1}^{s} X^j_{M_i} - C_{M_i}\right| \geq \varepsilon C_{M_i}\right) \leq 2\exp\left(-\frac{s}{(\frac{m}{\alpha|E_H|} - 1)}h(\varepsilon)\right) \tag{4.1}$$

Now, let $A_i =$ "$|C'_{M_i} - C_{M_i}| \geq \varepsilon C_{M_i}$", $i = 1,\dots,|\mathcal{M}(H,\ell)|$, namely A_i is the event that the estimate of motif $M_i, i = 1,\dots,|\mathcal{M}(H,\ell)|$ is distant more than εC_{M_i} from C_{M_i}. We already showed that that for an arbitrary A_i inequality (4.1) holds for $\mathbb{P}[A_i]$, so

$$\mathbb{P}\left(\bigcup_{i=1}^{|\mathcal{M}(H,\ell)|} A_i\right) \leq \sum_{i=1}^{|\mathcal{M}(H,\ell)|} \mathbb{P}[A_i] \leq |\mathcal{M}(H,\ell)|2\exp\left(-\frac{s}{(\frac{m}{\alpha|E_H|} - 1)}h(\varepsilon)\right) \leq \eta$$

combining the union bound and the choice of s as in statement. □

4.5 Experimental Evaluation

We implemented ODEN and tested it on several large datasets (see Section 4.5.1 for details on setup, and data). Our experimental evaluation has the following goals: compare ODEN with state-of-the-art algorithms for approximating motif counts (Section 4.5.2); evaluate the scalability of a simple parallel implementation of ODEN (Section 4.5.3); provide a case study highlighting the usefulness of using ODEN (Section 4.5.4) to analyze real-world temporal networks.

4.5.1 Setup, and Datasets

We briefly describe the setup and the large-scale datasets used in our experimental evaluation.

We implemented our algorithm ODEN in C++20 and compiled it under `gcc 9.3` with optimization flag enabled (implementation available at https://github.com/VandinLab/odeN), additional details on the implementation are in Section 4.6.2. We compared ODEN with four different baselines, denoted as PRESTO-A (PR-A), PRESTO-E (PR-E) [Sarpe and Vandin, 2021a], LS [Liu et al., 2019], and ES [Wang et al., 2020]. We used the original implementations available from the authors. We performed all experiments under Ubuntu 20.04 on a machine with 64 cores, Intel Xeon E5-2698 2.3GHz, running each algorithm single threaded and with 300GB of maximum RAM allowed.

The datasets used in our experimental evaluation are reported in Table 4.1, which shows the number of nodes and edges of T, the precision of the timestamps, the timespan of the network, the number $|E_{G_T}|$ of undirected edges in the corresponding undirected projected static network G_T, the maximum degree $d_{\max}$ of a node in G_T and the maximum number $w_{\max}$ of temporal edges that are mapped on the same static edge in G_T. The datasets are from different domains: SO is a network that models interactions from the Stack-Overflow platform [Paranjape et al., 2017], BI is a network of Bitcoin transactions [Liu et al., 2019], RE a network built from comments on the platform Reddit [Liu et al., 2019], and EC is a *bipartite* temporal network build from IPv4 packets exchanged between Chicago and Seattle [Sarpe and Vandin, 2021a]. See the original papers for more details on the networks and the processes they model.

When measuring the running times for the various algorithms we exclude the time to read the dataset. Since ES's implementation supports only values of ℓ up to 4, we do not report results for ES and $\ell > 4$. Unless otherwise stated we used $\delta = 86400$ for SO and RE, $\delta = 43200$ on BI, and $\delta = 50000$ on EC, as done in previous works [Paranjape et al., 2017, Liu et al., 2019, Wang et al., 2020]. Since all algorithms used in our comparison have different parameters and only ODEN counts multiple motifs simultaneously, we used the following procedure to choose the parameters. For a given target template H and ℓ, we run PRESTO-A, PRESTO-E, LS, and ES for each motif

Table 4.1: Datasets used and their statistics. See Section 4.5.1 for details on the statistics reported.

Name	n	m	$\lvert E_{G_T}\rvert$	$d_{\max}$	$w_{\max}$	Precision	Timespan
SO	2.58M	47.9M	28.1M	44K	594	sec	2774 (days)
BI	48.1M	113M	84.3M	2.4M	24.2K	sec	2585 (days)
RE	8.40M	636M	435.3M	0.3M	165K	sec	3687 (days)
EC	11.16M	2.32B	66.8M	0.3M	3.8M	μ-sec	62.0 (mins)

in $\mathcal{M}(H, \ell)$ with fixed parameters, and computed their running time as the sum of the running times required by the single motifs in $\mathcal{M}(H, \ell)$. We then fixed the parameters of ODEN so that its running time would be at most the same as the other methods, or be close to it. All the parameters used in the experiments (including sample sizes) are reported with the source code. To extract the *exact* counts of motifs we used a modified version of the algorithm by Mackey et al. [2018]. We do not report the running times of such algorithm since, even though it employs parallelism, it still runs slower than approximate approaches.

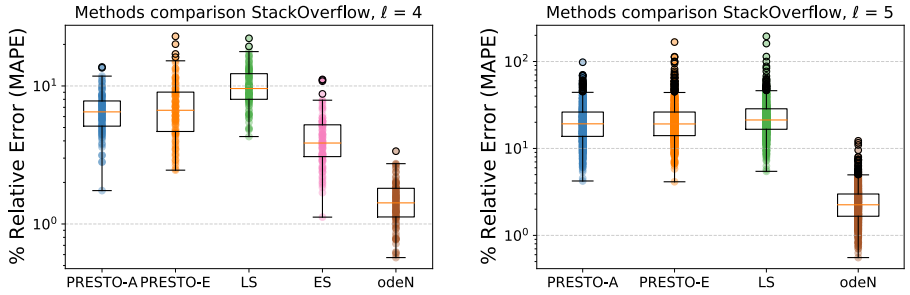
4.5.2 Approximation Quality and Running Time

In this section we compared the quality of the estimates and the running times of ODEN and the baseline sampling approaches.

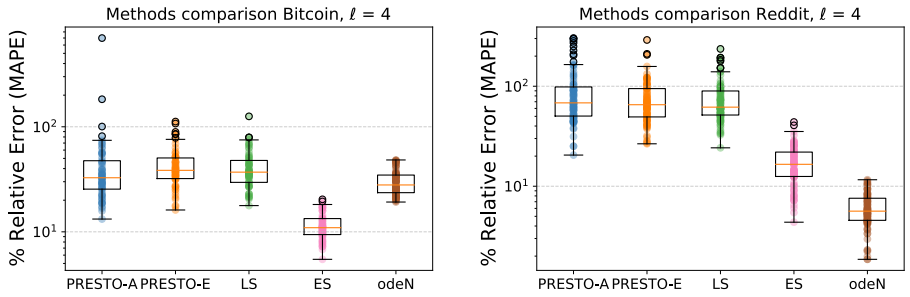
To evaluate the approximations qualities we used the MAPE (Mean Average Percentage Error) metric over ten executions of each algorithm and parameter configuration. The MAPE is computed as follows: let C'_{M_i} be the estimate of $C_{M_i}, i = 1, \ldots, |\mathcal{M}(H, \ell)|$, returned by an algorithm, then the relative error of such estimate is $|C'_{M_i} - C_{M_i}|/C_{M_i}$. The MAPE is the average over the ten runs of the relative errors, in percentage. On each of the ten runs we also measured the running time of each algorithm, for which we will report the arithmetic mean.

We first discuss the quality of the estimates for different datasets when H is a triangle and $\ell \in \{4, 5\}$. For $\ell = 4$ there are $|\mathcal{M}(H, \ell)| = 96$ triangles, while for $\ell = 5, |\mathcal{M}(H, \ell)|$ is 800. So as long as ℓ increases the approximation task becomes more challenging, due to the exponential growth of the number of motifs. We also observe that, to the best of our knowledge, such a huge number of temporal motifs was never tested before on large datasets due to the limitations of existing algorithms, while, as we will show, ODEN renders the approximation task practical even on hundreds of motifs.

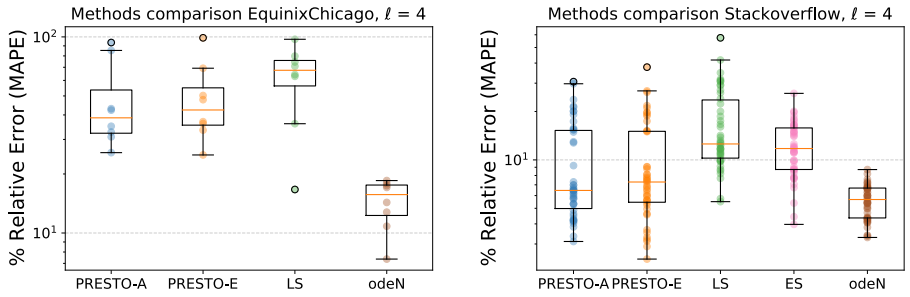
The results on the SO dataset are shown in Figure 4.4a. ODEN provides much sharper estimates than state-of-the-art sampling techniques for single motif estimations on motifs $M_1, \ldots, M_{|\mathcal{M}(H,\ell)|}$: the relative error on $\ell = 4$-edge triangles is bounded by 5%, and for $\ell = 5$-edge triangles (where $|\mathcal{M}(H, \ell)| = 800$) the relative error is bounded by 12% while state-of-the-art algorithms report much less accurate estimates, with twice the relative er-



a



b



c

Figure 4.4: Approximation error on different datasets. (4.4a): SO dataset, H is a triangle, for $\ell = 4$ (left) and $\ell = 5$ (right). (4.4b): H is a triangle, $\ell = 4$, BI dataset (left) and RE dataset (right). (4.4c): EC dataset, H is an edge, $\ell = 4$ (left); SO dataset, H is a square, $\ell = 4$.

ror of ODEN, on each configuration. We report the running times to obtain such estimates in Table 4.2. Interestingly, ODEN is more than 3× faster with $\ell = 4$ than any sampling algorithm and 1.7× faster with $\ell = 5$. For the other datasets, since extracting all the exact counts for $\ell > 4$ is extremely time consuming, requiring up to months of computation, we will not discuss the approximation qualities for $\ell = 5$ (since we do not have the exact counts to evaluate them).

On dataset BI (Figure 4.4b left) ODEN provides more concentrated estimates for the $|\mathcal{M}(H, \ell)| = 96$ triangles than other algorithms but ES, which also has a smaller running time than ODEN. This may be related to the static graph structure of BI, which has some very high-degree nodes (see Table

Table 4.2: Running times (in seconds) to obtain the results in Figure 4.4 (results are showed following the order in Figure 4.4). Under H we report the topolology of H used: T for triangles, E for edges, and S for squares. "-" denotes not applicable, while "✗" denotes out of RAM.

Dataset	ℓ	H	PR-A	PR-E	LS	ES	ODEN
SO	4	T	533.4	537.7	555.5	567.2	**174.4**
SO	5	T	4405	4408	4390	-	**2515**
BI	4	T	2048.6	2065.2	2754.6	**1602.9**	1948.9
RE	4	T	9787.1	10165.8	14289.7	13172.3	**6814.9**
EC	4	E	2581.5	3014.9	2981.9	✗	**1234.3**
SO	4	S	15613.7	16718.7	14344.6	26118.3	**4517.9**

4.1). Therefore ODEN may sample edges with very high degree nodes, introducing an over counting in its estimates. Nonetheless, for higher values of ℓ this issue is amortized over the growing number of motifs $|\mathcal{M}(H, \ell)|$.

On dataset RE (Figure 4.4b right) the estimates by ODEN are all within 13% of relative error and improve significantly over state-of-the-art sampling algorithms, up to one order of magnitude of precision. Such estimates were notably obtained with significantly smaller running time than state-of-the-art sampling algorithms, improving up to 2× the running time of ES and 1.4× over PRESTO (as reported in Table 4.2).

Finally, on the EC datasets, which is a bipartite temporal network with more than 2 billion edges we evaluated the approximation qualities with H being an edge and $\ell = 4$ (for which $|\mathcal{M}(H, \ell)| = 8$), such motifs have fundamental importance in the analysis of temporal networks since they can be seen as building blocks Holme and Saramäki [2019], Zhao et al. [2010]. We report the results on such motifs in Figure 4.4c (left) (ES is not shown since it did not terminate with the allowed memory budget). The estimates of ODEN are well concentrated and within 20% of relative error, while other sampling approaches provide approximations with a relative error up to 90% or more. Moreover, ODEN's results were obtained with a speedup of at least 2× over all the other sampling algorithms, rendering the approximations task feasible in a small amount of time on very large temporal networks.

To illustrate the enormous advantage of ODEN over existing state of the art exact and approximation algorithms, we compared the various algorithms on dataset SO when H is set to be a square and $\ell = 4$, for which $|\mathcal{M}(H, \ell)| = 48$. As Wang et al. [2020] observed, among the 4-edge square motifs there are 16 motifs that do not grow as a single component (i.e., their orderings start with $\langle (1,2)(3,4) \cdots \rangle$). Estimating the counts of such motifs is particularly hard for most of the current state-of-the-art sampling algorithms since they generate a large number of partial matchings, while such aspect does not impact ODEN. The results are shown in Figure 4.4c (right).

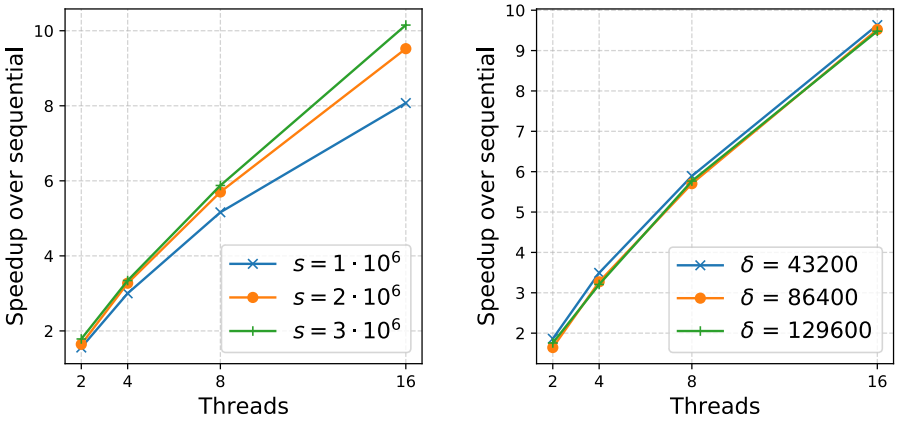


Figure 4.5: Speed-up of ODEN's parallel implementation. (Left): Varying s and fixed δ; (Right) Varying δ and fixed s.

ODEN provides tight approximations under 9% of relative error for *all* four-edge square motifs, while other sampling algorithms fail to provide sharp estimates for some of the motifs. Surprisingly, as shown in Table 4.2, to obtain such estimates ODEN required less than 1.3 hours of computation while the exact computation of the counts required more than two weeks, and ODEN it is at least 3× times faster than all algorithms, and it is 5.4× times faster than ES.

Overall, these results show that our algorithm ODEN achieves much more precise estimates within a significant smaller running time than state of the art sampling algorithms when estimating the counts $C_{M_1}, \ldots, C_{M_{|\mathcal{M}(H,\ell)|}}$ for different values of ℓ and different topologies of the target template H (see Problem 4.1 in Section 4.2).

4.5.3 Parallel Implementation

In this section we briefly describe the advantages of a simple parallel implementation of Algorithm 4. As discussed in Section 4.4.2 the `for` cycle (from line 3) can be trivially parallelized, therefore we implemented such strategy through a *thread pooling* design pattern.

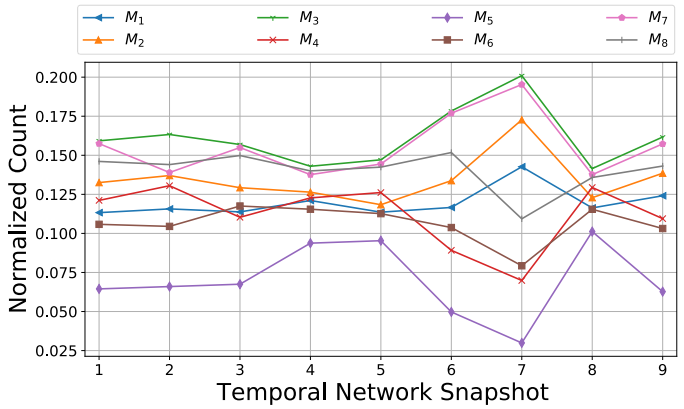
We describe the results obtained with H set to be a triangle, $\ell = 4$, and on the dataset SO; similar results are observed for other datasets. We tested the speedup achieved with $\omega \in \{2, 4, 8, 16\}$ threads over the sequential implementation. Let T_ω the average running time with ω threads over ten execution of ODEN with fixed parameters, with T_1 being the average time for running the algorithm sequentially. We report the value of T_1/T_ω, $\omega \in \{2, 4, 8, 16\}$, i.e., the speedup over the sequential implementation. Fig. 4.5 (Left) shows the speedup across different values of the sample size s, with $\delta = 86400$. We observe an almost linear speedup up to 4 threads and then a slightly worse performance, especially for small sample sizes, that may be related to the time needed to process each sample. Fig. 4.5 (Right)

shows how the speedup changes for $s = 2 \cdot 10^6$ and different values of δ. We note that our algorithm ODEN seems not to be impacted by the value of δ, and always attaining similar performances. Interestingly, as captured by our analysis in Section 4.4.3, the algorithm does not reach a fully linear speedup since we did not parallelized the computation of the sampling probabilities $p(e), e \in E_{G_T}$. As a remark, our parallel implementation is not optimized, and more advanced parallel strategies may substantially increase its speedup.

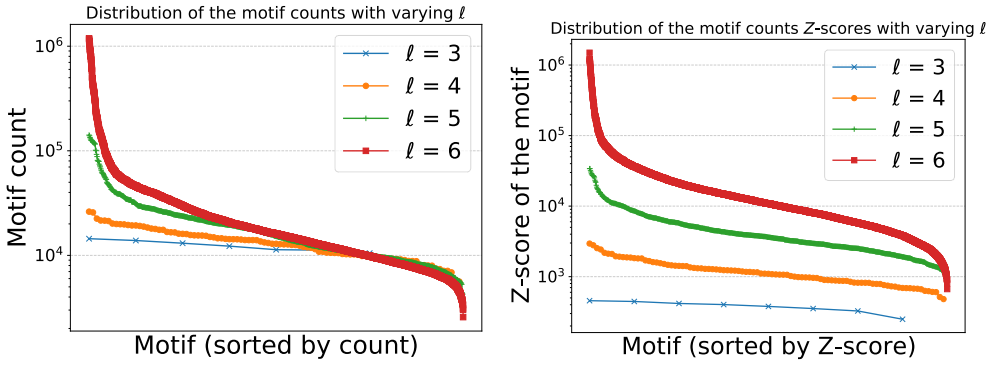
4.5.4 A Case Study

In this section we illustrate how counting multiple motifs, corresponding to the same target template H, with ODEN can be used to extract useful insights from a temporal network. We consider a real-world activity network from Facebook [Viswanath et al., 2009]. In such network, each node represents a user and a temporal edge (u, v, t) indicates that user u posted on v's wall at time t (see the original publication [Viswanath et al., 2009] for more details). The network contains information collected from September 2006 to January 2009. After removing self-loops, the network has n=45.7K nodes, m=826K temporal edges, and $|E_{G_T}|$=179K static (undirected) edges. We will fist show how analyzing the motif counts obtained with ODEN provides complementary insights to those in Viswanath et al. [2009], that relied on mostly static analyses. We then conclude by discussing how the counts of the network evolve by varying only the parameter ℓ (i.e., fixing H, δ), showing that such counts surprisingly differ with different values of such parameter.

In the original paper Viswanath et al. [2009], partitioned the Facebook network in nine different snapshots (obtaining nine projected static networks), with each snapshot spanning 90 days of interactions in the network. The authors observed that consecutive snapshots have small resemblance, i.e., on average only 45% of the edges are preserved through consecutive snapshots. The authors also observed that despite this difference all the snapshots have similar, almost invariant, structural properties in terms of their clustering coefficient, average degree distribution, and others. We used ODEN (with $\varepsilon = 1, \eta = 0.1$) to compare the temporal networks associated to the snapshots by computing the counts of the 8 temporal motifs in $\mathcal{M}(H, \ell = 3)$ with H being a triangle and $\delta = 86400 = 1$ day. On each snapshot, after extracting the motif counts, we computed for each motif M its normalized count on the snapshot as $C_M / \sum_{i=1}^{8} C_{M_i}$. The results are reported in Fig. (4.6a) (see Appendix 4.6.3 for a visual representation of the motifs). Interestingly, even if in Viswanath et al. [2009] the authors highlight small resemblance through different snapshots, the counts of the motifs are stable across the different snapshots, especially by looking at the first three and the last two snapshots. Surprisingly on snapshots 6 and 7, which correspond to the period of observation of mid-2008, we observe that there is a significant variation in the motif counts w.r.t. the previous months. This is the period where the authors of [Viswanath et al., 2009] observed a change in Facebook's interface (that led to a drop in the growth of the network)



a



b c

Figure 4.6: (4.6a): Counts of the motifs in $\mathcal{M}(H, 3)$ with H a triangle on each temporal network corresponding to one snapshot in Viswanath et al. [2009]. (4.6b): Counts on the full Facebook network with varying ℓ. (4.6c): Z-scores of the motif counts with varying ℓ.

that seems to be correlated to the variation on the motif counts. Even more surprisingly, this aspect is not captured by a static analysis of the snapshots as performed in Viswanath et al. [2009]. Thus, our temporal motifs analysis through ODEN is able to capture a variation in the growth of the network that the static analysis cannot highlight. (We discuss how the motifs and their counts can be used to characterize the activity on the network in Appendix 4.6.3).

We then analyzed how the different motif counts of the whole network change by varying the parameter ℓ. We fixed H a triangle and run ODEN with $\varepsilon = 1, \eta = 0.1, \delta = 86400$. The results are shown in Figure (4.6b). We observe that the counts of $M_1, \ldots, M_{|\mathcal{M}(H,\ell)|}$ vary significantly by increasing ℓ. For $\ell = 3$ almost all the motifs have the same counts, while for larger ℓ there are some motifs with very high counts (i.e., overrepresented) and some other motifs that are underrepresented. Overall the highest counts range from 10^4 to 10^6 from $\ell = 3$ up to $\ell = 6$. To understand if these counts increase only by chance, we performed a widely used statistical test (e..g, [Gauvin et al., 2018, Kovanen et al., 2013]) by computing the Z-scores of

the different motif counts under the following null model [Milo, 2004]. We generated 500 random networks by the timeline shuffling random model [Gauvin et al., 2018], which redistributes all the timestamps by fixing the *directed* projected static network. For each motif $M_i, i = 1, \ldots, |\mathcal{M}(H, \ell)|$ we computed a Z-score that is defined as follows: let C_{M_i} be the count of the motif in the original network and let $C^1_{M_i}, \ldots, C^{500}_{M_i}$ be its counts on the j-th random network $j \in \{1, \ldots, 500\}$. The Z-score is computed as, $Z_{M_i} = (C_{M_i} - \sum_{j=1}^{500} C^j_{M_i}/500)/\text{std}(C^1_{M_i}, \ldots, C^{500}_{M_i})$ where std$(\cdot)$ denotes the standard deviation. The results are in Figure (4.6c), and they show that the counts in Figure (4.6b) are very significant and not due to random fluctuations (higher Z-scores indicate that such motif counts are significantly more frequent in T than in the networks permutated randomly). Interestingly, the Z-scores in Figure (4.6c) follow a similar law to the counts in Figure (4.6b), with the highest Z-scores increasing significantly every time ℓ increases. Notably the highest Z-scores of motifs with $\ell = 6$ are more than 3 orders of magnitude larger than the Z-scores of motifs with $\ell = 3$. (We discuss some of the significant motifs in Section 4.6.3).

4.6 Additional Material

4.6.1 ODEN's Subroutines

4.6.1.1 `FastUpdate` and its Subroutines

We now discuss the `FastUpdate` routine that is called in line 10 of Algorithm 4 to keep $C_{estimates}$ updated. The `FastUpdate` subroutine is shown in Algorithm 5. $C_{estimates}$ maintains the weighted counts of the motif sequences identified, therefore to keep it updated we first count the δ-instances of $M_i, i = 1, \ldots |\mathcal{M}(H, \ell)|$ within the sampled temporal network i.e. S, and then rescale each count opportunely. Such routine will feature two main aspects, i) an efficient adaptation of the algorithm by Paranjape et al. Paranjape et al. [2017] and ii) an efficient encoding of the various sequences representing the motifs occurrences within integers that will allow for fast operations (comparisons to distinguish between different motifs and fast updates to the data structures).

We now discuss how `FastUpdate` counts all the δ-instances in S. First observe that we already know that $G_S \sim H$, and that S can be rewritten as $S = (((x_1, y_1), t_1), \ldots, ((x_\ell, y_\ell), t_\ell)$. We first compute the set $E_{unique} = \{(x, y) : ((x, y), t) \in S\}$ and we assign to each edge in E_{unique} a unique identifier (lines 4-5). Then we run an efficient implementation of the algorithm by Paranjape et al. [2017] that computes through dynamic programming the counts of all the subsequences of edges (x, y) s.t. $(x, y, t) \in S$ having length ℓ and occurring within δ-time (lines 6-10). In Algorithm 6 we show our implementation of the subroutines needed to execute lines 6-10 (see the original paper by Paranjape et al. [2017] for full details and correctness). Intuitively, lines 6-10 of Algorithm 5 scan the input sequence S linearly, maintaining in

Algorithm 5: `FastUpdate`

Input: $\delta, S, C_{estimates}, p(e_R), H$

1 $E_{unique} \leftarrow \{(x, y) : (x, y, t) \in S\}$
2 $Map_{id} \leftarrow \{\}$, $E_{rev} \leftarrow []$, $Map_{counts} \leftarrow \{\}$, $start \leftarrow 1$
3 $\texttt{id} \leftarrow 0$
4 **foreach** $e \in E_{unique}$ **do**
5 $\quad E_{rev}[\texttt{id}] \leftarrow e$, $Map_{id}\{e\} \leftarrow \texttt{id++}$
6 **foreach** $(x, y, t) \in S$ **do**
7 $\quad$ **while** $t - t_{start} > \delta$ **do**
8 $\quad\quad \texttt{Decrement}(Map_{id}[(x_{start}, y_{start})], Map_{counts})$
9 $\quad\quad start \leftarrow start + 1$
10 $\quad \texttt{Increment}(Map_{id}[(x, y)], Map_{counts})$
11 **foreach** *key* $\bar{k}$ *of length* $\ell \in Map_{counts}.keys$ **do**
12 $\quad M' \leftarrow \texttt{ReconstructMotif}(\bar{k}, E_{rev})$
13 $\quad$ **if** $G_u[M'] \sim H$ **then**
14 $\quad\quad M_i \leftarrow \texttt{EncodeAndClassifyMotif}(M')$
15 $\quad\quad X_{M_i} \leftarrow Map_{counts}\{\bar{k}\}/(|E_H|p(e_R))$
16 $\quad\quad X'_{M_i} \leftarrow C_{estimates}\{M_i\}$
17 $\quad\quad C_{estimates}\{M_i\} \leftarrow X'_{M_i} + X_{M_i}$

memory information about the edges within δ time from the processed one. Through such scan the algorithm updates Map_{counts} to keep the counts of the sequences having at most ℓ edges over the set E_{unique}. Starting the cycle in line 11, Map_{counts} contains the counts of all the ℓ subsequences of edges from S over the set E_{unique}. We highlight that we assign to each static edge of S an ID of b bits. This allows us to encode each sequence up to $j = 1, \ldots, \ell$ edges, occurring within δ time, in an integer using $j \cdot b$ bits through bitwise operations ("<<" denotes right shift and "|" denotes bitwise or) to allow for fast updates to Map_{counts}.

To obtain the estimates of motifs $M_1, \ldots, M_{|\mathcal{M}(H,\ell)|}$, for each ℓ sequence of edges identified we reconstruct the corresponding graph and thus the motif M' that the sequences is an instance of in line 12 (the multigraph is given by the edges ID's while the ordering of the edges is given by the sequence itself). We then check if $G_u[M']$ is isomorphic to H (constraint (1) from Problem 4.1). If so we encode the motif in a sequence of $2b\ell$ bits that allows us to classify such motif (line 14) in order to distinguish between distinct motifs (recall we want $M_i \not\cong_\tau M_j, i \neq j$). The encoding is computed as follows: given $M' = \langle (x_1, y_1), \ldots, (x_\ell, y_\ell) \rangle$ we assign to each node an incremental ID according to its first appearance in M' and we obtain the final encoding as $\langle \texttt{ID}(x_1)\texttt{ID}(y_1) \ldots \texttt{ID}(x_\ell)\texttt{ID}(y_\ell) \rangle$. It is easily seen that two motifs M_1, M_2 share the same encoding iff it holds $M_1 \cong_\tau M_2$ as desired, given that the motifs are *directed* and the definition of distinct motifs accounts for the ordering in which edges appear. We provide an example below.

Algorithm 6: Subroutines of `FastUpdate`

Function `Increment(id, `Map_{counts}`)`

1 **foreach** $\bar{k} \in$ `SortByDecLength(`$Map_{counts}.keys$`)` **do**
2 **if** $\bar{k}.length < \ell$ **then**
3 $\kappa \leftarrow (\bar{k} << b)|\texttt{id}$
4 $Map_{counts}[\kappa] \leftarrow Map_{counts}[\kappa] + Map_{counts}[\bar{k}]$
5 $Map_{counts}[\texttt{id}] \leftarrow Map_{counts}[\texttt{id}] + 1$

Function `Decrement(id, `Map_{counts}`)`

6 $Map_{counts}[\texttt{id}] \leftarrow Map_{counts}[\texttt{id}] - 1$
7 **foreach** $\bar{k} \in$ `SortByIncLength(`$Map_{counts}.keys$`)` **do**
8 **if** $\bar{k}.length < \ell - 1$ **then**
9 $\kappa \leftarrow (\texttt{id} << (\bar{k}.length \cdot b))|\bar{k}$
10 $Map_{counts}[\kappa] \leftarrow Map_{counts}[\kappa] - Map_{counts}[\bar{k}]$

Example 4.2. *Let us consider M_1, M_2, and M_3 from Figure 4.2. Consider $\sigma_1 = \langle (y,x),(y,z),(x,z)\rangle$, then by assigning an incremental ID to each node according to its first appearance in σ_1 we get $\texttt{ID}(y) = 1, \texttt{ID}(x) = 2, \texttt{ID}(z) = 3$ so the final encoding of M_1 is $\langle 121323\rangle$. Following a similar procedure the encoding of M_2 is $\langle 121323\rangle$, while the encoding M_3 is $\langle 121332\rangle$. The encodings of M_1 and M_2 coincide while differing from the one of M_3 as desired.*

After this step we update the global data structure $C_{estimates}$ by summing to each motif's estimate, its count in S divided by $|E_H|p(e_R)$ where $p(e_R)$ is the probability of edge e_R of being sampled (lines 15-17), which we prove in Section 4.4.4 to be the correct weighting schema to output an unbiased estimate.

4.6.1.2 Exact Subgraph Enumeration

In this section we briefly discuss the algorithms for subgraph enumeration that can be adapted to our Algorithm 4 (in line 5). Unfortunately we cannot easily use the algorithms for extracting k-node motifs mentioned in Section 4.3 *as is*, since they do not provide the local enumeration step required by ODEN.

In fact, the problem most related to the exact enumeration we require is the *labelled query graph matching problem.* In such setting one is provided a labelled query graph $H = (V_H, E_H, L_H)$, and a labelled graph $G = (V, E, L)$ (where labels can be colors for example, see Lee et al. [2012]), L may be defined both on edges or vertices. The problem requires to find all the subgraphs $h' \subseteq G$ isomorphic to H, which could be either induced or not but must preserve the labelling properties (i.e., if $(x,y) \in E$ is mapped to $(x',y') \in H$ then $(L(x), L(y)) = (L_H(x'), L_H(y'))$). To explain how we take advantage of the algorithms developed for the problem above we need to introduce the following definitions (adapted from Pashanasangi and Seshadhri

[2019]).

Definition 4.7. *Let $H = (V_H, E_H)$ be an undirected graph, an* automorphism *is a bijection $\pi : V_H \mapsto V_H$ such that $(x, y) \in E_H$ iff $(\pi(x), \pi(y)) \in E_H$.*

Definition 4.8. *Let $H = (V_H, E_H)$ be an undirected graph, we say that two edges $e = (x, y), e' = (x', y') \in E_H$ belong to the same edge-orbit iff there exists an automorphism that maps e on e'.*

In order to adapt the algorithms for the labelled query graph matching problem we proceed in the following way: 1) colour the nodes of G_T with a fixed colour (say red) 2) Once sampled $e_R \in G_T$, colour its endpoint nodes with a different colour (say blue), call the map from the last two points L_{G_T}; 3) compute the different edge-orbits of the pattern H (by enumerating the automorphisms of H) and for each edge-orbit choose an edge, colour its endpoint nodes with the same colour assigned to e_R, and keep the colour on the other edges the same as G_T, call this map L_H; 4) run an algorithm for the labelled query graph matching problem with graph $G_T = (V_T, E_{G_T}, L_{G_T})$ and pattern $H = (V_H, E_H, L_H)$ 5) the desired subgraphs ($\mathcal{H}$) are the union over the different edge-orbits enumeration steps.

4.6.2 Implementation Details

In this section we provide additional implementation details, complementing the description of Section 4.5.1.

In our implementation, we used two main structures: first, an adjacency list[3], that allows to query for an edge between $u, v \in V$ in $O(\log(\min(d_u, d_v)))$. Second, we used a hashmap to store for each static directed edge the timestamps of the temporal edges that map on that edge, leading to $O(1)$ complexity of querying for the timestamps of a static edge in G_T. The initialization of such structures is done in $O(1)$ per each processed temporal edge while loading the dataset, by knowing the number of nodes n. Many state of the art algorithms exist for the local enumeration of motifs (e.g., [Sun et al., 2020, Ren and Wang, 2015, Han et al., 2013]), we provide in our code a general algorithm based on the algorithm VF2++ [Jüttner and Madarasi, 2018]. However, instead of using the general procedure described in Section 4.6.1, in our test we relied on a simple algorithm that locally enumerates the subgraphs containing an edge $e = \{x, y\}$ isomorphic to H: for triangles the algorithm runs in $O(\min(d_x, d_y) \log(n))$, while when H is a square the algorithm runs in $O(\min(d_x, d_y) d_{max} \log(n))$, with d_{max} the maximum degree of a node in G_T.

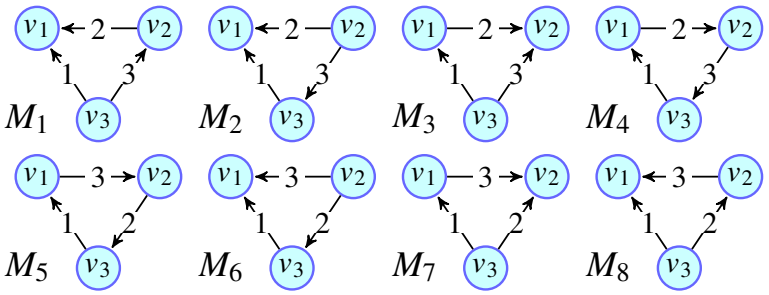


Figure 4.7: Graphical representation of the motifs in Figure (4.6a).

4.6.3 Case Study - Motif analysis

Motifs on the Snapshots of the Facebook Network.

Thanks to our analysis of Section 4.5.4 we are able to characterize the user behaviour on the Facebook network of wall posts by looking at different motifs (topology and their orderings) and their counts. We first show in Fig. 4.7 the motifs corresponding to the labels of Figure (4.6a) in Section 4.5.4. Then, let $H = \{v_1, v_2, v_3\}$ be a triangle, the most frequent motifs (i.e., those with the highest normalized counts on each snapshot) seem to share a common pattern: a first node (v_3) after posting on v_1's (or v_2's) wall triggers v_1 (or v_2's) to post on the remaining node's wall with v_1 posting also on such node's wall to close the triangle, as captured by motifs M_3, M_7 and M_8. Observe that by identifying the users that mostly act as v_3 in the occurrences of such frequent motifs one is able to identify, for example, the nodes more engaged in spreading most of the information over the Facebook network in a short period of time (recall that we set δ to one day). Not surprisingly motif M_5 is the less frequent one since its occurrences require node v_2 to post on v_3'wall before receiving the post from v_2 therefore without being "triggered" by such node, that received the post from v_3. Interestingly, without considering the orderings of occurrence among such patterns we will not be able to distinguish between the most frequent motifs and the least frequent ones since for example M_4 and M_5 have the same static *directed* graph structure but they have very different counts on the different snapshots of the Facebook network.

Motifs with Varying ℓ - Frequent vs Infrequent.

In this Section we briefly discuss the properties and show visually the motifs with highest and lowest Z-scores obtained in Section 4.5.4 on the Facebook wall post network for $\ell = 6$. The motifs are reported in Figure 4.8, where we report the 4-top motifs ranked by Z-score on the top and the 4-lowest motifs by Z-scores on the bottom. Note, that the top 4 motifs share a similar structure, both temporal and topological. Interestingly in the original paper

[3]We used the one provided by SNAP: https://github.com/snap-stanford/snap, more efficient implementations can be also adopted improving the global running times.

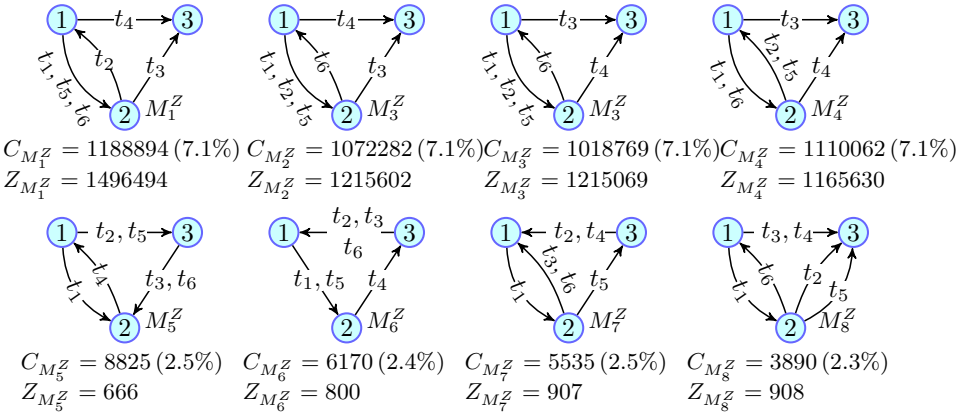


Figure 4.8: Graphical representation of the 4 motifs with highest (top) and lowest (bottom) Z-scores in Figure (4.6c) for $\ell = 6$. For each motif we report the exact count (which we computed for such representation) and the relative error in the approximation obtained with ODEN in brackets, we additionally report each Z-score of the motif as obtained from Section 4.5.4 (i.e., by using only ODEN).

by Viswanath et al. [2009] the authors noted that there were very few pair of nodes that exchanged more than 5 messages (with median 2). The most frequent temporal motifs seem to involve a pair of highly active nodes (which exchanged many messages between them, i.e., more than 4) and another third node that is reached by such pair of nodes. We unfortunately do not have the original messages to understand better the information captured by such frequent motifs (since we do not have the original posts), but it is really surprising that the top 4 motifs all share similar properties especially in the orderings of their edges. Additionally, it seems that triangles involving nodes that are pairwise very active seem to be the rarest type of interaction as captured by the 4 motifs with lowest Z-score, reported in Figure 4.8 bottom.

Chapter 5

Estimating Temporal Betweenness Centralities in Temporal Networks

In this chapter we present approximate algorithms for obtaining high-quality estimates to the temporal betweenness centrality values of the nodes in a temporal network. We then empirically evaluate the algorithms on several temporal networks, and show how our proposed algorithm "ONBRA" can be used to obtain results on datasets on which exact algorithm fails to scale the computation.

5.1 Introduction

The study of centrality measures is a fundamental primitive in the analysis of networked datasets [Borgatti and Everett, 2006, Newman, 2010], and plays a key role in social network analysis [Das et al., 2018]. A centrality measure informally captures how important a node is for a given network according to *structural* properties of the network. Central nodes are crucial in many applications such as analyses of co-authorship networks [Liu et al., 2005, Yan and Ding, 2009], biological networks [Wuchty and Stadler, 2003, Koschützki and Schreiber, 2008], and ontology summarization [Zhang et al., 2007].

One of the most important centrality measures is the betweenness centrality [Freeman, 1977, 1978], which informally captures the fraction of *shortest paths* going through a specific node. The betweenness centrality has found applications in many scenarios such as community detection [Fortunato, 2010], link prediction [Ahmad et al., 2020], and network vulnerability analysis [Holme et al., 2002]. The exact computation of the betweenness centrality of each node of a network is an extremely challenging task on modern networks, both in terms of running time and memory costs. Therefore, sampling algorithms have been proposed to compute provable high-quality approximations of the betweenness centrality values, while remarkably reducing the computational costs associated to this problem [Riondato and Kornaropoulos, 2016, Riondato and Upfal, 2018, Brandes and Pich, 2007].

Modern networks, additionally to being large, have also richer information about their edges. In particular, one of the most important and easily accessible information is the *time* at which edges occur. Such networks are often called *temporal networks* [Holme and Saramäki, 2019]. The analysis of temporal networks provides novel insights compared to the insights that would be obtained by the analysis of static networks (i.e., networks without temporal information), as, for example, in the study of subgraph patterns [Paranjape et al., 2017, Kovanen et al., 2011], community detection [Lehmann, 2019], and network clustering [Fu et al., 2020]. As well as for static networks, the study of the temporal betweenness centrality in temporal networks aims at identifying the nodes that are visited by a high number of *optimal* temporal paths [Holme and Saramäki, 2012, Buß et al., 2020]. In temporal networks in fact, the definition of optimal paths has to consider the information about the timing of the edges, making the possible definitions of optimal paths much more richer than in static networks [Rymar et al., 2021].

In this work, a temporal path is valid if it is time respecting, i.e., if all the edges defining the path occur at increasing timestamps (see Figures 5.1b-5.1c). Additional to paths we will also consider *walks* that, intuitively, are time respecting paths where a node can appear multiple times across the edges defining the walk. We considered two different optimality criteria for temporal paths (or walks), chosen for their relevance [Holme and Saramäki, 2012][1]: (i) shortest temporal path (STP) criterion, a commonly used criterion for which a path is optimal if it uses the minimum number of interactions to connect a given pair of nodes; (ii) shortest restless temporal walk (RTW) criterion, for which a walk is optimal if, in addition to being shortest, all its consecutive interactions occur at most within a given user-specified time duration parameter $\delta \in \mathbb{R}^+$ (see Figure 5.1c). The RTW criterion finds application, for example, in the study of spreading processes over complex networks [Pan and Saramäki, 2011], where information about the timing of consecutive interactions is fundamental. The exact computation of the temporal betweenness centrality under the STP and RTW optimality criteria becomes impractical (both in terms of running time and memory usage) for even moderately-sized networks. Furthermore, as well as for static networks, obtaining a high-quality approximation of the temporal betweenness centrality of a node is often sufficient in many applications. Thus, we propose ONBRA, the *first* algorithm to compute rigOrous estimatioN of temporal Betweenness centRality values in temporAl networks[2], providing sharp guarantees on the quality of its output. As for many data-mining algorithms, ONBRA's output is function of two parameters: $\varepsilon \in (0, 1)$ controlling the estimates' accuracy; and $\eta \in (0, 1)$ controlling the confidence. The algorithmic problems arising from accounting for temporal information are really challenging to deal with compared to the static network scenario, al-

[1]The general schema of ONBRA can be adapted to other optimality criteria, as we will discuss.

[2]https://vec.wikipedia.org/wiki/Onbra.

though ONBRA shares a high-level sampling strategy similar to the work by Riondato and Upfal [2018]. Finally, we show that in practice our algorithm ONBRA, other than providing high-quality estimates while reducing computational costs, it also enables analyses that cannot be otherwise performed with existing state-of-the-art algorithms. Our main contributions are the following:

- We propose ONBRA, the first sampling-based algorithm that outputs high-quality approximations of the temporal betweenness centrality values of the nodes of a temporal network. ONBRA leverages on an advanced data-dependent and variance-aware concentration inequality to provide sharp probabilistic guarantees on the quality of its estimates. ONBRA is able to compute high-quality estimates for the temporal betweenness centrality values of the nodes defined both on paths or walks under many different optimality criteria.
- We show how to adapt ONBRA for the temporal betweenness centrality defined on paths under the STP criterion and on walks under the RTW criterion. In particular, we devise specific algorithms to be used within ONBRA to address the computation of the estimates under the STP and RTW criteria.
- We perform an extensive experimental evaluation with several goals: (i) under the STP criterion, show that studying the temporal betweenness centrality provides novel insights compared to the static betweenness; (ii) under the STP criterion, show that ONBRA provides high-quality estimates, while significantly reducing the computational costs compared to the state-of-the-art exact algorithm, and that it enables the study of large datasets that cannot practically be analyzed by the existing exact algorithm; (iii) show that ONBRA is able to estimate the temporal betweenness centrality under the RTW optimality criterion by varying the value of $\delta \in \mathbb{R}^+$, for which no practical algorithms currently exist.

5.2 Preliminaries

In this section we introduce the fundamental notions needed throughout the development of our work and formalize the problem of approximating the temporal betweenness centrality of the nodes in a temporal network.

We start by recalling the definition of temporal networks.

Definition 5.1. *A* temporal network T *is a pair* $T = (V, E)$*, where* V *is a set of* n *nodes (or vertices), and* $E = \{(u, v, t) : u, v \in V, u \neq v, t \in \mathbb{R}^+\}$ *is a set of* m *directed edges*[3][4].

[3] ONBRA can be easily adapted to work on *undirected* temporal networks with minor modifications.

[4] Without loss of generality we assume the edges $(u_1, v_1, t_1), \ldots, (u_m, v_m, t_m)$ to be sorted

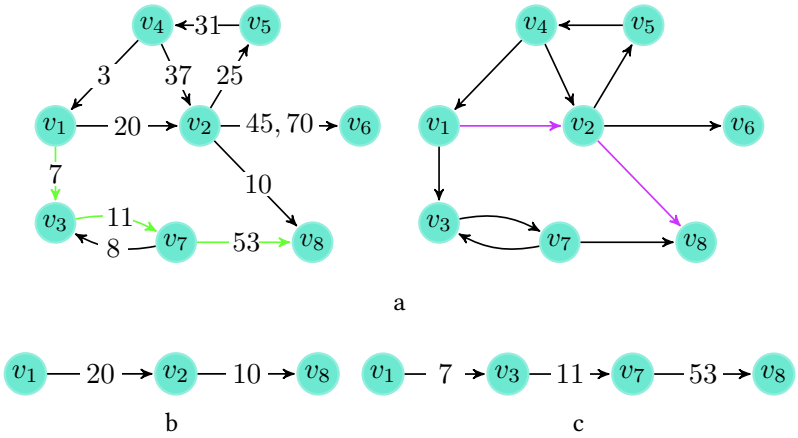


Figure 5.1: (5.1a): (left) a temporal network T with $n = 8$ nodes and $m = 12$ edges, (right) its associated static network G_T obtained from T by removing temporal information. A shortest *temporal* path cannot be identified by a shortest path in the static network: e.g., the shortest paths from node v_1 to node v_8, respectively coloured in green in T and purple in G_T, are different. (5.1b): A path that is not time respecting. (5.1c): A time respecting path that is also shortest in T. With $\delta \geq 42$ such path is also shortest δ-restless walk.

Each edge $e = (u, v, t) \in E$ of the network represents an interaction from node $u \in V$ to node $v \in V$ at time t, which is the *timestamp* of the edge. Figure 5.1a (left) provides an example of a temporal network T. Next, we define *temporal paths*.

Definition 5.2. *Given a temporal network T, a* temporal path P *is a sequence* $\mathsf{P} = \langle e_1 = (u_1, v_1, t_1), e_2 = (u_2, v_2, t_2), \ldots, e_k = (u_k, v_k, t_k)\rangle$ *of k edges of T ordered by increasing timestamps*[5]*, i.e., $t_i < t_{i+1}, i \in \{1, \ldots, k-1\}$, such that the node v_i of edge e_i is equal to the node u_{i+1} of the consecutive edge e_{i+1}, i.e., $v_i = u_{i+1}, i \in \{1, \ldots, k-1\}$, and each node $v \in V$ is visited by* P at most *once.*

A shortest temporal *walk* is a temporal path were we *drop* the constraint of each node $v \in V$ of being visited at most once. A walk $\mathsf{W} = \langle e_1 = (u_1, v_1, t_1), e_2 = (u_2, v_2, t_2), \ldots, e_k = (u_k, v_k, t_k)\rangle$ is said *δ-restless* if $t_{i+1} - t_i \leq \delta$ for $i \in \{1 \ldots, k-1\}, \delta \in \mathbb{R}^+$. Given a temporal path P made of k edges, we define its length as $\ell_{\mathsf{P}} = k$. An example of temporal path P of length $\ell_{\mathsf{P}} = 3$ is given by Figure 5.1c. Given a *source* node $s \in V$ and a *destination* node $z \in V, z \neq s$, a *shortest* temporal path (resp. *shortest* δ-restless walk) between s and z is a temporal path (resp. δ-restless temporal walk) $\mathsf{P}_{s,z}$ of length $\ell_{\mathsf{P}_{s,z}}$ such that in T there is no temporal path (resp. δ-restless temporal walk) $\mathsf{P}'_{s,z}$ connecting s to z of length $\ell_{\mathsf{P}'_{s,z}} < \ell_{\mathsf{P}_{s,z}}$. Given a temporal shortest path (resp. shortest δ-restless walk)$\mathsf{P}_{s,z}$ connecting s and z, we define $\texttt{Int}(\mathsf{P}_{s,z}) = \{w \in V | \exists (u, w, t) \vee (w, v, t) \in \mathsf{P}_{s,z}, w \neq s, z\} \subset V$ as the set of nodes *internal*

by increasing timestamps.

[5]Our work can be easily adapted to deal with non-strict ascending timestamps (i.e., with $\leq$ constraints) for the STP criterion.

to the path $\mathsf{P}_{s,z}$. Let $\sigma_{s,z}^{sh}$ (resp. $\sigma_{s,z}^{rstw}$) be the number of shortest temporal paths (resp. shortest δ-restless walks) between nodes s and z. Given a node $v \in V$, we denote with $\sigma_{s,z}^{sh}(v)$ (resp. $\sigma_{s,z}^{srtw}(v)$)the number of shortest temporal paths (resp. shortest δ-restless walks) $\mathsf{P}_{s,z}$ connecting s and z for which v is an internal node, i.e., $\sigma_{s,z}^{\{sh,rstw\}}(v) = |\{\mathsf{P}_{s,z} | v \in \texttt{Int}(\mathsf{P}_{s,z})\}|$ where $\mathsf{P}_{s,z}$ will be clear from the context if referred to a shortest temporal path or a shortest δ-restless temporal walk. Now we introduce the *temporal betweenness centrality* of a node $v \in V$, which intuitively captures the fraction of *optimal* temporal paths (or walks) visiting v.

Definition 5.3. *Given an optimality criterion OPT over paths (or walks) we define the* temporal betweenness centrality $b(v)$ *of a node $v \in V$ as*

$$b(v) = \frac{1}{n(n-1)} \sum_{s,z \in V,\ s \neq z} \frac{\sigma_{s,z}^{OPT}(v)}{\sigma_{s,z}^{OPT}}. \tag{5.1}$$

In the above Equation (5.1) we will use "sh" instead OPT to denote optimal paths under the STP criterion (i.e., shortest temporal paths) and "$srtw$" to denote optimal walks under the RTW criterion (i.e., shortest δ-restless temporal walks). Let $B(T) = \{(v, b(v)) : v \in V\}$ be the set of pairs composed of a node $v \in V$ and its temporal betweenness value $b(v)$. Since the exact computation of the set $B(T)$ using state-of-the-art exact algorithms [Buß et al., 2020, Rymar et al., 2021] is impractical on even moderately-sized temporal networks (see Section 5.5 for experimental evaluations), in our work we aim at providing high-quality approximations of the temporal betweenness centrality values of all the nodes of the temporal network. That is, we compute the set $\tilde{B}(T) = \{(v, \tilde{b}(v)) : v \in V\}$, where $\tilde{b}(v)$ is an accurate estimate of $b(v)$, controlled by two parameters $\varepsilon, \eta \in (0, 1)$, (accuracy and confidence). We want $\tilde{B}(T)$ to be an *absolute (ε, η)-approximation set* of $B(T)$, as commonly adopted in data-mining algorithms (e.g., in [Riondato and Upfal, 2018, Riondato and Vandin, 2020]): that is, $\tilde{B}(T)$ is an approximation set such that

$$\mathbb{P}\left[\sup_{v \in V} |\tilde{b}(v) - b(v)| \leq \varepsilon\right] \geq 1 - \eta.$$

Note that in an absolute (ε, η)-approximation set, for each node $v \in V$, the estimate $\tilde{b}(v)$ of the temporal betweenness value deviates from the actual value $b(v)$ of at most ε, with probability at least $1 - \eta$. Finally, let us state the main computational problem addressed in this work.

Problem 5.1. *Given a temporal network T and two parameters $(\varepsilon, \eta) \in (0, 1)^2$, compute the set $\tilde{B}(T)$, i.e., an absolute (ε, η)-approximation set of $B(T)$.*

5.3 Related Works

Given the importance of the betweenness centrality for network analysis, many algorithms have been proposed to compute it in different scenarios. In

this section we focus on those scenarios most relevant to our work, grouped as follows.

Approximation Algorithms for Static Networks. Recently, many algorithms to approximate the betweenness centrality in static networks have been proposed, most of them employ randomized sampling approaches [Riondato and Kornaropoulos, 2016, Riondato and Upfal, 2018, Brandes and Pich, 2007]. The existing algorithms differ from each other mainly for the sampling strategy they adopt and for the probabilistic guarantees they offer. Among these works, the one that shares similar ideas to our work is by Riondato and Upfal [2018], where the authors proposed to sample pairs of nodes $(s, z) \in V^2$, compute all the shortest paths from s to z, and update the estimates of the betweenness centrality values of the nodes internal to such paths. The authors developed a suite of algorithms to output an (ε, η)-approximation set of the set of betweenness centrality values. Their work cannot be easily adapted to temporal networks. In fact, static and temporal paths in general are not related in any way, and the temporal scenario introduces many novel challenges: (i) computing the optimal temporal paths, and (ii) updating the betweenness centrality values. Therefore, our algorithm ONBRA employs the idea of the estimator provided by Riondato and Upfal [2018], while using novel algorithms designed for the context of temporal networks. Furthermore, the probabilistic guarantees provided by our algorithm ONBRA leverage on the variance of the estimates, differently from Riondato and Upfal [2018] that used bounds based on the Rademacher averages. Our choice to use a empirical based variance-aware concentration inequality is motivated by the recent interest in providing sharp guarantees employing the *empirical variance* of the estimates [Cousins et al., 2021, Pellegrina and Vandin, 2021].

Algorithms for Dynamic Networks. In this setting the algorithm keeps track of the betweenness centrality value of each node for every timestamp $t_1, \ldots, t_m$ observed in the network [Lee et al., 2012, Hanauer et al., 2021]. Note that this is extremely different from estimating the temporal betweenness centrality values in temporal networks. In the dynamic scenario the paths considered are *not* required to be time respecting. For example, in the dynamic scenario, if we consider the network in Figure 5.1a (left) at any time $t > 20$, the shortest path from v_1 to v_8 is the one highlighted in purple in Figure 5.1a (right). Instead, in the temporal setting such path is not time respecting. We think that it is very challenging to adapt the algorithms for dynamic networks to work in the context of temporal networks, which further motivates us to propose ONBRA.

Exact Algorithms for Temporal Networks. First of all we note that algorithms for temporal betweenness centrality are strictly related to algorithms for counting temporal paths (or walks) for which significant work has been done [Thejaswi et al., 2020, Casteigts et al., 2021, Xuan et al., 2003, Enright et al., 2022, Michail, 2016]. Related to our definition of paths Wu et al. [2014] discussed several conditions and algorithms to find shortest, foremost, fastest, and reverse foremost paths in temporal networks. Their algo-

rithms which are state-of-the-art for counting such paths do not enumerate *all* optimal temporal paths (or walks) according to the optimality criterion adopted as required for computing the temporal betweenness centrality but rather adopt several pruning conditions to speed-up the computation, in fact computing *all* fastest paths is #*P*-Hard as proved by Buß et al. [2020]. Michail [2016] discussed algorithms that can be used for computing paths (or walks) through static expansions, which become inefficient when temporal networks are supplemented with fine-grained information as in our scenario. Additionally Casteigts et al. [2021] proved that identifying restless temporal *paths* is NP-Complete in temporal networks, but such problem is known to be solvable in polynomial time on shortest restless *walks*, as considered in this work.

Several exact approaches have been proposed in the literature for temporal betweenness centrality [Tsalouchidou et al., 2020, Alsayed and Higham, 2015, Kim and Anderson, 2012]. The algorithm most relevant to our work was presented by Buß et al. [2020], where the authors extended the well-known Brandes algorithm [Brandes, 2001] to the temporal network scenario considering the STP criterion (among several other criteria). They showed that the time complexity of their algorithm is $O(n^3(t_m - t_1)^2)$, which is often impractical on even moderately-sized networks. Recently, Rymar et al. [2021] discussed conditions on temporal paths under which the temporal betweenness centrality can be computed in polynomial time, showing a general algorithm running in $O(n^2m(t_m - t_1)^2)$ even under the RTW criterion, which is again very far from being practical on modern networks.

We conclude by observing that, to the best of our knowledge, no approximation algorithms exist for estimating the temporal betweenness centrality in temporal networks.

5.4 ONBRA

In this section we discuss ONBRA, our novel algorithm for computing high-quality approximations of the temporal betweenness centrality values of the nodes of a temporal network. We first discuss the sampling strategy used in ONBRA, then we present the algorithm and all its subroutines required for ONBRA to work under the STP and RTW criteria, and finally we prove the theoretical guarantees on the quality of the estimates of ONBRA.

5.4.1 Sampling Strategy

In this section we discuss the sampling strategy adopted by ONBRA that is independent of the optimality criterion of the paths (or walks) considered. However, for the sake of presentation, we discuss the sampling strategy for the STP-based temporal betweenness centrality estimation, then we discuss how to adapt ONBRA to work for the temporal betweenness defined on the RTW as optimality criterion.

ONBRA samples *pairs* of nodes (s, z) and computes all the shortest temporal paths from s to z. More formally, let $\mathcal{D} = \{(u, v) \in V^2 : u \neq v\}$, and $\ell \in \mathbb{N}, \ell \geq 2$ be a user-specified parameter. ONBRA first collects ℓ pairs of nodes $(s_i, z_i)_i, i = 1, \ldots, \ell$, sampled uniformly at random from $\mathcal{D}$. Next, for each pair (s, z) it computes $\mathcal{P}_{s,z} = \{\mathrm{P}_{s,z} : \mathrm{P}_{s,z} \text{ is shortest}\}$, i.e., the set of shortest temporal paths from s to z. Then, for each node $v \in V$ s.t. $\exists \mathrm{P}_{s,z} \in \mathcal{P}_{s,z}$ with $v \in \texttt{int}(\mathrm{P}_{s,z})$, i.e., for each node v that is internal to a shortest temporal path of $\mathcal{P}_{s,z}$, ONBRA computes the estimate $\tilde{b}'(v) = \sigma^{sh}_{s,z}(v)/\sigma^{sh}_{s,z}$, which is an unbiased estimator of the temporal betweenness centrality value $b(v)$ as we show in Lemma 5.1 (i.e., $\mathbb{E}[\tilde{b}'(w)] = b(v)$). Note, that the estimates obtained by ONBRA can be stated in the framework we presented in Section 2.2. In particular, $\mathcal{X}$ is identified with $\mathcal{D}$, and ρ is uniform over $\mathcal{X}$, given $x = (s, z)$ an element sampled from $\mathcal{X}$, $\mathcal{F}$ is as follows, $f_v(x) = \sigma^{sh}_{s,z}(v)/\sigma^{sh}_{s,z}, v \in V$, and $\hat{X}_{f_v}(x) = f_v(x), v \in V$.

Lemma 5.1. *Let $v \in V$, then $\tilde{b}'(v)$ is an unbiased estimator of $b(v)$.*

Proof. Let X_{sz} be a Bernoulli random variable that takes value 1 if the pair of nodes $(s, z) \in \mathcal{D}$ is sampled, and 0 otherwise. Since $\mathbb{E}[X_{sz}] = 1/(n(n-1))$, then by the linearity of expectation,

$$\mathbb{E}[\tilde{b}'(v)] = \frac{1}{n(n-1)} \sum_{s,z \in V s \neq z} \frac{\sigma^{sh}_{s,z}(v)}{\sigma^{sh}_{s,z}} \frac{\mathbb{E}[X_{sz}]}{1/n(n-1)} = b(v).$$

□

Finally, after processing the ℓ pairs of nodes randomly selected, ONBRA computes for each node $v \in V$ the (unbiased) estimate $\tilde{b}(v)$ of the actual temporal betweenness centrality $b(v)$ by averaging $\tilde{b}'(v)$ over the ℓ sampling steps: $\tilde{b}(v) = 1/\ell \sum_{i=1}^{\ell} \tilde{b}'(v)_i$, where $\tilde{b}'(v)_i$ is the estimate of $b(v)$ obtained by analyzing the i-th sample, $i \in [1, \ell]$. We will discuss the theoretical guarantees on the quality of the estimates $\tilde{b}(v), v \in V$ in Section 5.4.5.

5.4.2 Algorithm Description

Sampling Algorithm: ONBRA

ONBRA is presented in Algorithm 7. In line 1 we first initialize the set $\mathcal{D}$ of objects to be sampled, where each object is a pair of distinct nodes from V. Next, in line 2 we initialize the matrix $\tilde{B}$ of size $|V| \cdot \ell$ to store the estimates of ONBRA for each node at the various iterations, needed to compute their empirical variance and the final estimates. Then we start the main loop (line 3) that will iterate ℓ times. In such loop we first select a pair (s, z) sampled uniformly at randomly from $\mathcal{D}$ (line 4). We then compute all the shortest temporal paths from s to z by executing Algorithm 8 (line 5), which is described in detail later in this section. Such algorithm computes *all* the shortest temporal paths from s and z adopting some pruning criteria to speed-up the computation. If at least one STP between s and z exists (line 6), then

Algorithm 7: ONBRA.

Input: Temporal network $T = (V, E)$, $\eta \in (0, 1)$, $\ell \geq 2$
Output: Pair $(\varepsilon', \tilde{B}(T))$ s.t. $\tilde{B}$ is an absolute (ε', η)-approximation set of $B(T)$.

1 $\mathcal{D} \leftarrow \{(u, v) \in V \times V, u \neq v\}$
2 $\tilde{B}_{v,:} \leftarrow \vec{0}_\ell, \forall v \in V$
3 **for** $i \leftarrow 1$ **to** ℓ **do**
4 $(s, z) \leftarrow$ `uniformRandomSample(`$\mathcal{D}$`)`
5 `SourceDestinationSTPComputation(`T, s, z`)`
6 **if** reached(z) **then**
7 `updateSTPEstimates(`$\tilde{B}, i$`)`
8 $\tilde{B}(T) \leftarrow \{(v, 1/\ell \sum_{i=1}^{\ell} \tilde{B}_{v,i}) : v \in V\}$
9 $\varepsilon' \leftarrow \sup_{v \in V} \left\{ \sqrt{\frac{2\mathbf{V}(\tilde{B}_{v,:}) \ln(4n/\eta)}{\ell}} + \frac{7 \ln(4n/\eta)}{3(\ell-1)} \right\}$
10 **return** $(\varepsilon', \tilde{B}(T))$

for each node $v \in V$ internal to a path in $\mathcal{P}_{s,z}$ we update the corresponding estimate to the current iteration by computing $\tilde{b}'(v)_i$ using Algorithm 9 (line 7). While in static networks this step can be done with a simple recursive formula [Riondato and Upfal, 2018], in our scenario we need a specific algorithm to deal with the more challenging fact that a node may appear at different distances from a given source across different shortest temporal paths. We will discuss in detail such algorithm later in this section. At the end of the ℓ iterations of the main loop, ONBRA computes: (i) the set $\tilde{B}(T)$ of unbiased estimates (line 8); (ii) and a tight bound ε' on $\sup_{v \in V} |\tilde{b}(v) - b(v)|$, which leverages the empirical variance $\mathbf{V}(\tilde{B}_{v,:})$ of the estimates (line 9). We observe that ε' is such that the set $\tilde{B}(T)$ is an absolute (ε', η)-approximation set of $B(T)$. We discuss the computation of such bound in Section 5.4.5. Finally, ONBRA returns $(\varepsilon', \tilde{B}(T))$.

5.4.3 Shortest Temporal Path Betweenness

We now describe the subroutines employed in Algorithm 7 focusing on the STP criterion. Then, in Section 5.4.4, we discuss how to address the estimation of the temporal betweenness defined under the RTW criterion.

Source-Destination Shortest Paths Computation.

We start by introducing some definitions needed through this section. First, we say that a pair $(v, t) \in V \times \{t_1, \ldots, t_m\}$ is a *vertex appearance* (VA) if $\exists (u, v, t) \in E$. Next, given a VA (v, t) we say that a VA (w, t') is a *predecessor* of (v, t) if $\exists (w, v, t) \in E, t' < t$. Finally, given a VA (v, t) we define its set of *out-neighbouring VAs* as $\mathcal{N}^+(v, t) = \{(w, t') : \exists (v, w, t') \in E, t < t'\}$.

We now describe Algorithm 8 that computes the shortest temporal paths

Algorithm 8: Source-Destination STP computation.

Input: $T = (V, E)$, source node s, destination node z

```
1  for v ∈ V do
2      dist_v ← −1; σ_v ← 0
3  for (u, v, t) ∈ E do
4      σ_{v,t} ← 0; P_{v,t} ← ∅; dist_{v,t} ← −1
5  dist_s ← 0; dist_{s,0} ← 0
6  σ_s ← 1; σ_{s,0} ← 1; d_z^min ← ∞
7  Q ← empty queue; Q.enqueue((s, 0))
8  while !Q.empty() do
9      (v, t) ← Q.dequeue()
10     if (dist_{v,t} < d_z^min) then
11         for (w, t') ∈ N^+(v, t), do
12             if dist_{w,t'} = −1 then
13                 dist_{w,t'} ← dist_{v,t} + 1
14                 if dist_w = −1 then
15                     dist_w ← dist_{v,t} + 1
16                     if w = z then
17                         d_z^min ← dist_w
18                 Q.enqueue((w, t'))
19             if dist_{w,t'} = dist_{v,t} + 1 then
20                 σ_{w,t'} ← σ_{w,t'} + σ_{v,t}
21                 P_{w,t'} ← P_{w,t'} ∪ {(v, t)}
22                 if dist_{w,t'} = dist_w then
23                     σ_w ← σ_w + σ_{v,t}
```

between a source node s and a destination node z (invoked in ONBRA at line 5). Such computation is optimized to prune the search space once found the destination z. The algorithm initializes the data structures needed to keep track of the shortest temporal paths that, starting from s, reach a node in V, i.e., the arrays dist$[\cdot]$ and $\sigma[\cdot]$ that contain for each node $v \in V$, respectively, the minimum distance to reach v and the number of shortest temporal paths reaching v (line 2). In line 4 we initialize dist$[\cdot,\cdot]$ that keeps track of the minimum distance of a VA from the source s, $\sigma[\cdot,\cdot]$ that maintains the number of shortest temporal paths reaching a VA from s, and P keeping the set of predecessors of a VA across the shortest temporal paths explored. After initializing the values of the data structures for the source s and $d_z^{\min}$ keeping the length of the minimum distance to reach z (lines 5-6), we initialize the queue Q that keeps the VAs to be visited in a BFS fashion in line 7 (observe that, since the temporal paths need to be time-respecting, all the paths need to account for the time at which each node is visited). Next, the algorithm explores the network in a BFS order (line 8), extracting

a VA (v, t) from the queue, which corresponds to a node and the time at which such node is visited, and processing it by collecting its set $\mathcal{N}^+(v, t)$ of out-neighbouring VAs (lines 9-11). If a VA (w, t') was not already explored (i.e., it holds $\text{dist}_{w,t'} = -1$), then we update the minimum distance $\text{dist}_{w,t'}$ to reach w at time t', the minimum distance dist_w of the vertex w if it was not already visited, and, if w is the destination node z, we update $\text{d}_z^{\min}$ (lines 12-17). Observe that the distance $\text{d}_z^{\min}$ to reach z is used as a *pruning criterion* in line 10 (clearly, if a VA appears at a distance greater than $\text{d}_z^{\min}$ then it cannot be on a shortest temporal path from s to z). After updating the VAs to be visited by inserting them in Q (line 18), if the current temporal path is shortest for the VA (w, t') analyzed, we update the number $\sigma_{w,t'}$ of shortest temporal paths leading to it, its set $P_{w,t'}$ of predecessors, and the number σ_w of shortest temporal paths reaching the node w (lines 19-23).

Recall that for static graphs, if a shortest path leading from s to z contains one node w then also the path leading from s to w needs to be shortest (i.e., let s, z be two nodes if we have that $\langle s, \ldots, w, \ldots, z\rangle$ is shortest then the two paths $\langle s, \ldots, w\rangle$ and $\langle w, \ldots, z\rangle$ will be shortest). In order to prove the correctness of Algorithm 8 we need to prove an analogous property of temporal paths (that can be phrased in the more general framework by Rymar et al. [2021] of *prefix-compatibility*).

Property 5.1. *Given a shortest temporal path* $\text{P}_{\text{s,z}}$ *from* $s \in V$ *to* $z \in V$*, let* (w, t') *be a VA s.t., it appears in* $\text{P}_{\text{s,z}}$*, then the temporal (sub)path leading from* s *to* w *at time* t' *is shortest.*

Proof. By contradiction, assume that a temporal path $\text{P}_{\text{s,z}} = \langle(s, \cdot, t_1), \ldots (\cdot, w, t'), \ldots, (\cdot, z, t_2)\rangle$ is shortest, then suppose that the temporal sub-path $\text{P}_1 = \langle(s, \cdot, t_1), \ldots (\cdot, w, t')\rangle$ is not shortest, this means that there exists a path $\text{P}_2 = \langle(s, \cdot, t_1), \ldots (\cdot, w, t')\rangle$ such that $|\text{P}_2| < |\text{P}_1|$. That is, the composition of P_2 and the path after P_1 in $\text{P}_{\text{s,z}}$ forms a path that is shorter than $\text{P}_{\text{s,z}}$ contradicting the initial assumption. □

Lemma 5.2. *Algorithm 8 computes exactly* $\sigma_{w,t}$ *for each vertex appearance internal to path in* $\mathcal{P}_{s,z}$*, and* $\sigma_w, \forall w : w \in P, P \in \mathcal{P}_{s,z} \vee w = z$.

Proof. First observe that during the execution of Algorithm 8 the following properties are maintained, $\text{dist}_{w,t}$ maintains the *minimum* distance to reach node w at time t with a path starting from s. This is immediate to observe since the algorithm exploits nodes (and therefore paths) through a BFS, hence nodes are visited in increasing order of distance from s. Hence, the first time a vertex appearance is processed (i.e., $\text{dist}_{w,t} = -1$) the algorithm is guaranteed that the distance of such VA is minimum by Property 5.1. Additionally, if a node is reached trough a path having minimum distance $\sigma_{w,t'}$ is updated accordingly by adding $\sigma_{v,t}$, i.e., the number of shortest paths going through the VA preceding (w, t'). Hence the algorithm computes exactly the desired quantities $\sigma_{w,t}$ and $\sigma_w \forall w : w \in P, P \in \mathcal{P}_{s,z} \vee w = z$. Additionally, the truncation occurs only for VAs that dist more than $d_z^{\min}$ and such VAs cannot contribute to optimal paths by Property 5.1. □

Algorithm 9: Update betweenness estimates - STP.

Input: $\tilde{B}, i$.
for $(u, v, t) \in E$ **do**
  $\sigma^z_{v,t} \leftarrow 0$; $M_{v,t} \leftarrow$ False
$R \leftarrow$ empty queue;
foreach $t : (\sigma_{z,t} > 0)$ **do**
  for $(w, t') \in P_{z,t}$ **do**
    $\sigma^z_{w,t'} \leftarrow \sigma^z_{w,t'} + 1$
    if $!M_{w,t'}$ **then**
      R.enqueue((w, t')); $M_{w,t'} \leftarrow$ True
while $!R$.empty() **do**
  $(w, t) \leftarrow R$.dequeue()
  if $w \neq s$ **then**
    $\tilde{B}_{w,i} \leftarrow \tilde{B}_{w,i} + \sigma^z_{w,t} \cdot \sigma_{w,t}/\sigma_z$
    for $(w', t') \in P_{w,t}$ **do**
      $\sigma^z_{w',t'} \leftarrow \sigma^z_{w',t'} + \sigma^z_{w,t}$
      if $!M_{w',t'}$ **then**
        R.enqueue((w', t')); $M_{w',t'} \leftarrow$ True

Lemma 5.3. *Algorithm 8 runs in* $O(m \log(d^+_{\max}))$.

Proof. Note that each vertex appearance (w, t') is processed at most once and the number of such VAs is $O(m)$. To find $(w, t') \in \mathcal{N}^+(v, t)$ of a given VA (v, t) it requires $O(\log(d^+(v)))$ through our data-structures, where $d^+(v) = |\{t : \exists(v, w, t) \in E\}|$ Additionally, for each VA that is processed the algorithm performs $O(1)$ number of operations. Therefore the final complexity follows. □

Update Estimates – STP criterion

Now we describe Algorithm 9, which updates the temporal betweenness estimates of each node internal to a path in $\mathcal{P}_{s,z}$ already computed (through Algorithm 8). With Algorithm 8 we computed for each VA (w, t) the number $\sigma_{w,t}$ of shortest temporal paths from s reaching (w, t). Now, in Algorithm 9 we need to combine such counts to compute the total number of shortest temporal paths leading to each VA (w, t) appearing in a path in $\mathcal{P}_{s,z}$, allowing us to compute the estimate of ONBRA for each node w.

At the end of Algorithm 8 there are in total $|\mathcal{P}_{s,z}|$ shortest temporal paths reaching z from s. Now we need to compute, for each node w internal to a path in $\mathcal{P}_{s,z}$ and for each VA (w, t), the number $\sigma^z_{w,t}$ of shortest temporal paths leading from w to z at a time greater that t. Then, the fraction of paths containing the node w is computed with a simple formula, i.e., $\sum_t \sigma^z_{w,t} \cdot \sigma_{w,t}/\sigma_z$, where $\sigma_z = |\mathcal{P}_{s,z}|$. The whole procedure is described in Algorithm

9. We start by initializing $\sigma^z_{v,t}$ that stores for each VA (v,t) the number of shortest temporal paths reaching z at a time greater than t starting from v, and a boolean matrix M that keeps track for each VA if it has already been considered (line 2). In line 3 we initialize a queue R that will be used to explore the VAs appearing along the paths in $\mathcal{P}_{s,z}$ in reverse order of distance from s starting from the destination node z. Then we initialize $\sigma^z_{w,t'}$ for each VA reaching z at a given time t' (line 6), and we insert each VA in the queue only one time (line 8). The algorithm then starts its main loop exploring the VAs in decreasing order of distance starting from z (line 9). We take the VA (w,t) to be explored in line 10. If w differs from s (i.e., w is an internal node), then we update its temporal betweenness estimate by adding $\sigma^z_{w,t} \cdot \sigma_{w,t}/\sigma_z$ (line 12). As we did in the initialization step, then we process each predecessor (w',t') of (w,t) across the paths in $\mathcal{P}_{s,z}$ (line 13), update the count $\sigma^z_{w',t'}$ of the paths from the predecessor to z by summing the number $\sigma^z_{w,t}$ of paths passing through (w,t) and reaching z (line 14), and we enqueue the predecessor (w',t') only if it was not already considered (lines 15-16). So, the algorithm terminates by having properly computed for each node $v \in V, v \neq s,z$ the estimate $\tilde{b}'(v)_i$ for each iteration $i \in [1,\ell]$.

Proposition 5.1. *Algorithm 9 computes correctly $\tilde{b}'(v)_i = \sigma_{sz}(v)/\sigma_{sz}$ for each node $v \in V$ internal to a path between s and z at each iteration $i \in [1,\ell]$ for the STP criterion.*

Proof. First observe that the correctness follows if $\sigma^z_{w,t}$ contains the number of paths in $\mathcal{P}_{s,z}$ that reach z *after* the VA (w,t), i.e., paths of the form $\langle(\cdot,w,t),\ldots,(\cdot,z,t')\rangle$ s.t. $t' > t$, since if this is correct $\tilde{b}'(v)_i$ can be computed by:

$$\tilde{b}'(w)_i = \sum_{t:\sigma^z_{w,t}>0} \frac{\sigma^z_{w,t}\sigma_{w,t}}{\sigma_z}. \tag{5.2}$$

The terms $\sigma^z_{w,t}$ can be computed recursively by exploiting the sets $P_{w,t}$ where $w \neq s$ and $t : \sigma_{w,t} > 0$. Starting from z, fix a node $w \neq z$ such that $\exists \mathsf{P} \in \mathcal{P}_{s,z} : (w,z,t) \in \mathsf{P}$ (i.e., w dists one edge from z on an optimal shortest temporal path) we have that $\sigma^z_{w,t} = \sum_{t':P_{z,t'}\neq\emptyset} |\{(w,t) : (w,t) \in P_{z,t'}\}|$. Then for the other nodes $w \in \mathtt{int}(P_{s,z}), P_{s,z} \in \mathcal{P}_{s,z}, w \neq s,z$ we have that $\sigma^z_{w,t} = \sum_{w':\exists t',(w,t)\in P_{w',t'}} \sum_{t':P_{w',t'}\neq\emptyset} \sigma^z_{w',t'}$ and this is computed recursively in line 14. We explore VAs in reverse order of distance from s, hence thanks to Property 5.1 we know that once we process a VA, this can appear only at a fixed distance from s. Since VAs are explored in decreasing order, once we process a VA (w,t) the value $\sigma^z_{w,t}$ has been correctly computed by evaluating the queue, since it is based on VAs preceding (w,t) in the queue, hence we can update the betweenness of the node w by summing $\sigma^z_{w,t} \cdot \sigma_{w,t}/\sigma_z$, that contributes to the general summation in Equation (5.2). Hence as the exploration of the algorithm concludes all the estimates $\tilde{b}'(w)_i$ are computed correctly. □

Lemma 5.4. *Algorithm 9 runs in $O(md^-_{\max})$.*

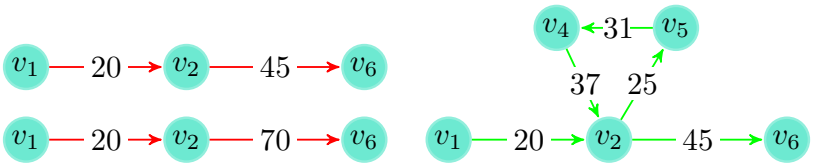


Figure 5.2: Considering the temporal network in Figure 5.1 and $\delta = 10$, the paths from node v_1 to node v_6 on the left are not shortest δ-restless since both violate the timing constraint (i.e., $45 - 20, 70 - 20 > \delta$). Instead, the walk on the right is shortest and meets the timing constraint with $\delta = 10$: so, it is a shortest δ-restless walk.

Proof. Note that again each vertex appearance (w, t) is processed at most once and the number of such VAs is $O(m)$. The maximum size of a set $P_{w,t}$ is $O(d^-(w))$, where $d^-(w) = |\{t : \exists(v, w, t) \in E\}|$ Additionally, for each VA in the set $P_{w,t}$ that is processed the algorithm performs $O(1)$ number of operations. Therefore the final complexity follows. □

Lemma 5.5. *The running time of* ONBRA *under the STP criterion is bounded by* $O(\ell[m(\log(d^+_{\max}) + d^-_{\max}) + n])$

Proof. This is immediately obtained by combining Lemma 5.3, Lemma 5.4, and recalling that such algorithms are executed on $O(\ell)$ samples, and the fact that to evaluate the bound providing ε', for each node in the temporal network we need $O(\ell)$ time, hence the additional $O(n\ell)$ time complexity. □

5.4.4 Shortest Restless Temporal Walk Betweenness

In this section we present the algorithms that are used in ONBRA when considering the RTW criterion for the optimal walks to compute the temporal betweenness centrality values, i.e., when defining Equation (5.1) for shortest δ-restless temporal walks hence when we set $OPT = srtw$.

Recall that, in such scenario, a temporal walk $\mathcal{W} = \langle e_1 = (u_1, v_1, t_1),\ e_2 = (u_2, v_2, t_2), \ldots, e_k = (u_k, v_k, t_k)\rangle$ is considered optimal if and only if $\mathcal{W}$, additionally to being *shortest*, is such that, given $\delta \in \mathbb{R}^+$, it holds $t_{i+1} \leq t_i + \delta$ for $i = 1, \ldots, k-1$. Considering the RTW criterion, we need to relax the definition of shortest temporal paths and, instead, consider *shortest temporal walks*. We provide an intuition of why we need such requirement in Figure 5.2. Given $\delta \in \mathbb{R}^+$, we refer to a shortest temporal walk as *shortest δ-restless temporal walk*.

In order to properly work under the RTW criteria, ONBRA needs novel algorithms to compute the optimal walks and update the betweenness estimates. Note that to compute the shortest δ-restless temporal walks we can use Algorithm 8 provided that we add the condition $t' - t \leq \delta$ in line 11.

More interestingly, the biggest computational problem arises when updating the temporal betweenness values of the various nodes on the optimal walks. Note that, to do so, we cannot use Algorithm 9 because it does not account for cycles (i.e, when vertices appear multiple times across a walk). We

Algorithm 10: Update betweenness estimates - RTW.

```
   Input: B̃, i.
1  for (u, v, t) ∈ E do
2      σ^z_{v,t} ← 0; u_{v,t} ← 0
3  R ← empty queue;
4  foreach t : σ_{z,t} > 0 do
5      for (w, t') ∈ P_{z,t} do
6          R.enqueue(⟨(w, t'), {z}⟩);
7  while !R.empty() do
8      ⟨(w, t), S⟩ ← R.dequeue()
9      if w ≠ s then
10         if w ∉ S then
11             B̃_{w,i} ← B̃_{w,i} + σ^z_{w,t} · σ_{w,t}/(σ_z · u_{w,t})
12         S' ← S ∪ {w}
13         for (w', t') ∈ P_{w,t} do
14             σ^z_{w',t'} ← σ^z_{w',t'} + σ^z_{w,t}/u_{w,t}
15             u_{w',t'} ← u_{w',t'} + 1
16             R.enqueue(⟨(w', t'), S'⟩);
```

therefore introduce Algorithm 10 that works in the presence of cycles. The main intuition behind Algorithm 10 is that we need to recreate backwards all the optimal walks obtained through the RTW version of Algorithm 8. For each walk we will maintain a set that keeps track of the nodes already visited up to the current point of the exploration of the walk, updating a node's estimate if and only if we see such node for the first time. This is based on the simple observation that a cycle cannot alter the value of the betweenness centrality of a node on a fixed walk, allowing us to account only once for the node's appearance along the walk.

We now describe Algorithm 10 by discussing its differences with Algorithm 9. In line 2, instead of maintaining a matrix keeping track of the presence of a VA in the queue, we now initialize a matrix $\mathsf{u}[\cdot,\cdot]$ that keeps the number of times a VA is in the queue. The queue, initialized in line 3, keeps elements of the form $\langle\cdot,\cdot\rangle$, where the first entry is a VA to be explored and the second entry is the set of nodes already visited backwards along the walk leading to such vertex appearance. While visiting backwards each walk, we check if the nodes are visited for the first time on such walk: if so, we update the betweenness values by accounting for the number of times we will visit such VA across other walks (lines 10-11). Next, we update the set of nodes visited (line 12). Finally, we update the count $\sigma^z_{w',t'}$ of the walks leading from the predecessor (w',t') of the current VA (w,t) to z (line 14), the number $\mathsf{u}_{w',t'}$ of times such predecessor will be visited (line 15), and enqueue the predecessor (w',t') to be explored, together with the additional

information of the set S′ of nodes explored up to that point. To conclude, note that Algorithm 10 is more expensive than Algorithm 9 since it recreates all the optimal walks, while Algorithm 9 avoids such step given the absence of cycles. In fact, we just briefly highlight that Lemma 5.3 holds also with the modification on RTWs. Instead Algorithm 10 runs in $O(m|\mathcal{W}|^* d^-_{\max})$ where $|\mathcal{W}|^* = \max\{|\mathcal{W}_{s,z}| : s, z \in V^2, s \neq z\}$ hence the final time complexity of ONBRA is bounded by $O(\ell[m(\log(d^+_{\max} + |\mathcal{W}|^* d^-_{\max}) + n)])$.

5.4.5 ONBRA – Theoretical Guarantees

In order to address Problem 5.1, ONBRA bounds the deviation between the estimates $\tilde{b}(v)$ and the actual values $b(v)$, for every node $v \in V$. To do so, we leverage on the so called *empirical Bernstein bound*, which we adapted to ONBRA.

Given a node $v \in V$, let $\tilde{B}_{v,:} = (\tilde{b}'(v)_1, \tilde{b}'(v)_2, \ldots, \tilde{b}'(v)_\ell)$, where $\tilde{b}'(v)_i$ is the estimate of $b(v)$ by analysing the i-th sample, $i \in \{1, \ldots, \ell\}$. Let $\mathbf{V}(\tilde{B}_{v,:})$ be the *empirical variance* of $\tilde{B}_{v,:}$:

$$\mathbf{V}(\tilde{B}_{v,:}) = \frac{1}{\ell(\ell-1)} \sum_{1 \leq < i < j \leq \ell} (\tilde{b}'(v)_i - \tilde{b}'(v)_j)^2.$$

We use the *empirical Bernstein bound* to limit the deviation between $\tilde{b}(v)$'s and $b(v)$'s, which represents Corollary 5 of Maurer and Pontil [2009] adapted to our framework, since Corollary 5 of Maurer and Pontil [2009] is formulated for generic random variables taking values in $[0, 1]$ and for an arbitrary set of functions.

Theorem 5.1 (Corollary 5, [Maurer and Pontil, 2009]). *Let $\ell \geq 2$ be the number of samples, and $\eta \in (0, 1)$ be the confidence parameter. Let $\tilde{b}'(v)_i$ be the estimate of $b(v)$ by analysing the i-th sample, $i \in \{1, \ldots, \ell\}$ and $v \in V$. Let $\tilde{B}_{v,:} = (\tilde{b}'(v)_1, \tilde{b}'(v)_2, \ldots, \tilde{b}'(v)_\ell)$, and $\mathbf{V}(\tilde{B}_{v,:})$ be its empirical variance. With probability at least $1 - \eta$, and for every node $v \in V$, we have that*

$$|\tilde{b}(v) - b(v)| \leq \sqrt{\frac{2\mathbf{V}(\tilde{B}_{v,:}) \ln(4n/\eta)}{\ell}} + \frac{7 \ln(4n/\eta)}{3(\ell - 1)}.$$

The right hand side of the inequality of the previous theorem differs from Corollary 5 in the work of Maurer and Pontil [2009] by a factor of 2 in the arguments of the natural logarithms, since in [Maurer and Pontil, 2009] the bound is not stated in the symmetric form reported in Theorem 5.1. Finally, the result about the guarantees on the quality of the estimates provided by ONBRA follows.

Corollary 5.1. *Given a temporal network T, the pair $(\varepsilon', \tilde{B}(T))$ in output from* ONBRA *is such that, with probability $> 1 - \eta$, it holds that $\tilde{B}(T)$ is an absolute (ε', η)-approximation set of $B(T)$.*

Observe that Corollary 5.1 is independent of the structure of the optimal paths considered by ONBRA, therefore such guarantees hold for both the criteria considered in our work, and possibly many others.

Table 5.1: Datasets used and their statistics.

Name	n	m	Granularity	Timespan
HighSchool2012 (HS)	180	45K	20 sec	7 (days)
CollegeMsg	1.9K	59.8K	1 sec	193 (days)
EmailEu	986	332K	1 sec	803 (days)
FBWall (FB)	35.9K	199.8K	1 sec	100 (days)
Sms	44K	544.8K	1 sec	338 (days)
Mathoverflow	24.8K	390K	1 sec	6.4 (years)
Askubuntu	157K	727K	1 sec	7.2 (years)
Superuser	192K	1.1M	1 sec	7.6 (years)

5.5 Experimental Evaluation

In this section we present our experimental evaluation that has the following goals: (i) motivate the study of the temporal betweenness centrality by showing two real world temporal networks on which the temporal betweenness provides novel insights compared to the static betweenness computed on their associated static networks; (ii) assess, considering the STP criterion, the accuracy of the ONBRA's estimates, and the benefit of using ONBRA instead of the state-of-the-art exact approach [Buß et al., 2020], both in terms of running time and memory usage; (iii) finally, show how ONBRA can be used on a real world temporal network to analyze the RTW-based betweenness centrality values.

5.5.1 Setup

We implemented ONBRA in C++20 and compiled it using `gcc 9`. The code is publicly available[6]. All the experiments were performed sequentially on a 72 core Intel Xeon Gold 5520 @ 2.2GHz machine with 1008GB of RAM available. The real world datasets we used are described in Table 5.1, which are mostly social or message networks from different domains. Such datasets are publicly available online[7]. For detailed descriptions of such datasets we refer to the links reported and [Paranjape et al., 2017]. To obtain the FBWall dataset we cut the last 200K edges from the original dataset [Viswanath et al., 2009], which has more than 800K edges. Such cut is done to allow the exact algorithm to complete its execution without exceeding the available memory.

5.5.2 Temporal vs Static Betweenness

In this section we assess that the temporal betweenness centrality of the nodes of a temporal network provides novel insights compared to its static

[6] https://github.com/iliesarpe/ONBRA.

[7] http://www.sociopatterns.org/ and https://snap.stanford.edu/temporal-motifs/data.html.

Table 5.2: Static vs temporal top-k nodes Jaccard similarity $J(k)$. We also report the size of the intersection.

Name	$J(25)$	$J(50)$
HS	0.28 (11)	0.56 (36)
FB	0.22 (9)	0.18 (15)

version. To do so, we computed for two datasets, from different domains, the exact ranking of the various nodes according to their betweenness values. The goal of this experiment is to compare the two rankings (i.e., temporal and static) and understand if the relative orderings are preserved, i.e., verify if the most central nodes in the static network are also the most central nodes in the temporal network. To this end, given a temporal network $T = (V, E)$, let $G_T = (V, \{(u, v) : \exists(u, v, t) \in E\})$ be its associated static network. We used the following two real world networks: (i) HS, that is a temporal network representing a face-to-face interaction network among students; (ii) and FB, that is a Facebook user-activity network [Viswanath et al., 2009] (see Table 5.1 for further details).

We first computed the exact temporal and static betweenness values of the different nodes of the two networks. Then, we ranked the nodes by descending betweenness values. We now discuss how the top-k ranked nodes vary from temporal to static on the two networks. We report in Table 5.2 the Jaccard similarity between the sets containing the top-k nodes of the static and temporal networks. In particular, given T and G_T, let $S_T^k = \{v_1, \ldots, v_k\}$ be the top-k nodes ranked by their temporal betweenness values and let $S_{G_T}^k = \{v'_1, \ldots, v'_k\}$ be the top-k nodes ranked by their static betweenness values. We report in Table 5.2 the Jaccard similarity $J(k) = |S_T^k \cap S_{G_T}^k| / |S_T^k \cup S_{G_T}^k|$ for two different values of k. On HS, for $k = 25$, only 11 nodes are top ranked in both the rankings, which means that less than half of the top-25 nodes are central if only the static information is considered. The value of the intersection increases to 36 for $k = 50$, since the network has only 180 nodes. More interestingly, also on the Facebook network only few temporally central nodes can be detected by considering only static information: only 9 over the top-25 nodes and 15 over the top-50 nodes. In order to better visualize the top-k ranked nodes, we show their betweenness values in Figure 5.3a: note that there are many top-k temporally ranked nodes having small static betweenness values, and vice versa.

These experiments show the importance of studying the temporal betweenness centrality, which provides novel insights compared to the static version.

5.5.3 Accuracy and Resources of ONBRA

In this section we first assess the accuracy of the estimates $\tilde{B}(T)$ provided by ONBRA considering only the STP criterion, since for the RTW criterion no

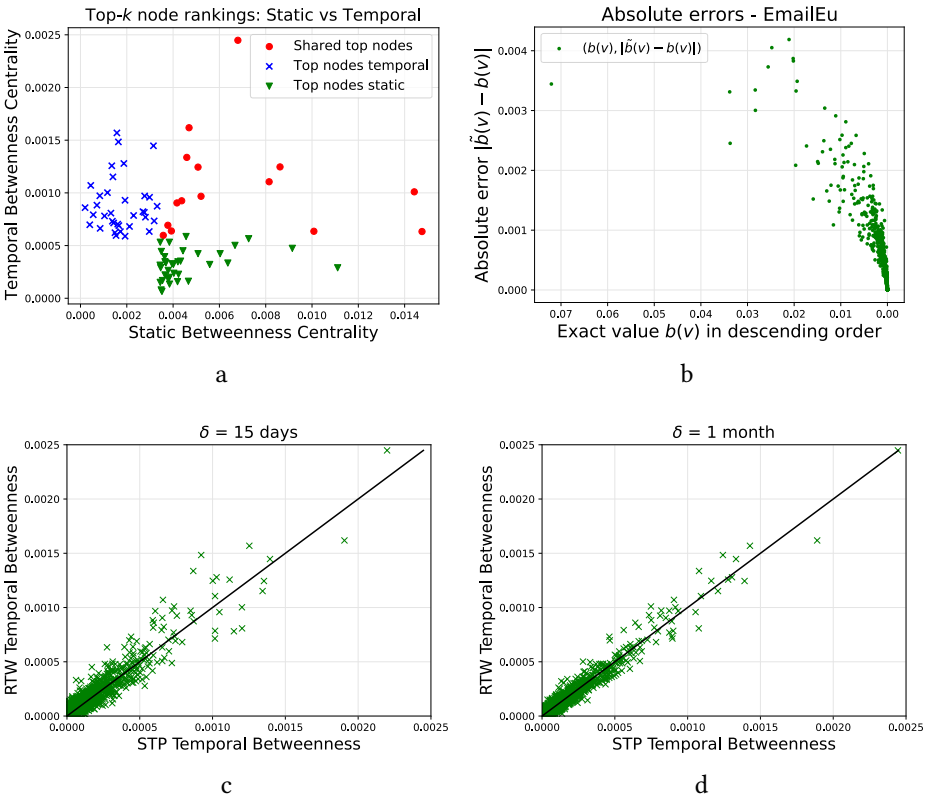


Figure 5.3: (5.3a): static and temporal betweenness values of the top-50 ranked nodes of the dataset FB; (5.3b): for dataset EmailEu, the deviations (or absolute errors) $|\tilde{b}(v) - b(v)|$ between the estimates $\tilde{b}(v)$ and the actual values $b(v)$ of the temporal betweenness centrality, for decreasing order of $b(v)$; (5.3c,5.3d): comparison between the temporal betweenness values based on STP and RTW, for δ=15 days (left) and δ=1 month (right).

implemented exact algorithm exists. Then, we show the reduction of computational resources induced by ONBRA compared to the exact algorithm by Buß et al. [2020].

To assess ONBRA's accuracy and its computational cost, we used four datasets, i.e., CollegeMsg, EmailEu, Mathoverflow, and FBWall. We first executed the exact algorithm, and then we fix $\eta = 0.1$ and ℓ properly for ONBRA to run within a fraction of the time required by the exact algorithm. The results we now present, which are described in detail in Table 5.3, are all averaged over 10 runs (except for the RAM peak, which is measured over one single execution of the algorithms).

Remarkably, even using less than 1% of the overall pairs of nodes as sample size, ONBRA is able to estimate the temporal betweenness centrality values with very small average deviations between $4 \cdot 10^{-6}$ and $5 \cdot 10^{-4}$, while obtaining a significant running time speed-up between $\approx$1.5$\times$ and $\approx$4$\times$ with respect to the exact algorithm [Buß et al., 2020]. Additionally, the amount of RAM memory used by ONBRA is significantly smaller than the exact al-

Table 5.3: For each dataset, the average and maximum deviation between the estimate $\tilde{b}(v)$ and the actual temporal betweenness value $b(v)$ over all nodes v and 10 runs, respectively *Avg. Error* and $\sup_{v \in V} |b(v) - \tilde{b}(v)|$, the theoretical upper bound ε', the *Sample rate* (%) of pairs of nodes we sampled, the running time t_{EXC} and peak RAM memory MEM_{EXC} required by the exact approach [Buß et al., 2020], the running time t_{ONBRA} and peak RAM memory MEM_{ONBRA} required by ONBRA. The symbol ✗ denotes that the exact computation of [Buß et al., 2020] is not able to conclude on our machine.

Dataset	Avg. Error	$\sup_{v \in V} \lvert b(v) - \tilde{b}(v) \rvert$	ε'	Sample rate (%)	t_{EXC} (sec)	t_{ONBRA} (sec)	MEM_{EXC} (GB)	$\text{MEM}_{\text{ONBRA}}$ (GB)
CollegeMsg	$1.74 \cdot 10^{-4}$	$6.38 \cdot 10^{-3}$	$2.27 \cdot 10^{-2}$	0.083	231	**148**	12.0	**0.13**
EmailEu	$4.69 \cdot 10^{-4}$	$1.35 \cdot 10^{-2}$	$6.15 \cdot 10^{-2}$	0.093	7211	**1808**	23.9	**2.1**
Mathoverflow	$6.35 \cdot 10^{-6}$	$2.1 \cdot 10^{-3}$	$5.38 \cdot 10^{-3}$	0.005	79492	**36983**	1004.3	**6.8**
FBWall	$4.25 \cdot 10^{-6}$	$5.89 \cdot 10^{-4}$	$2.13 \cdot 10^{-3}$	0.003	11489	**3145**	738.0	**11.1**
Askubuntu	✗	✗	$6.92 \cdot 10^{-3}$	0.00006	✗	**35585**	>1008	**20.3**
Sms	✗	✗	$1.54 \cdot 10^{-3}$	0.00231	✗	**13020**	>1008	**16.2**
Superuser	✗	✗	$1.02 \cdot 10^{-2}$	0.00003	✗	**41856**	>1008	**16.7**

gorithm by Buß et al. [2020]: e.g., on the Mathoverflow dataset ONBRA requires only 6.8 GB of RAM peak, which is 147× less than the 1004.3 GB required by the exact state-of-the-art algorithm [Buß et al., 2020]. Furthermore, in all the experiments we found that the maximum deviation is distant at most one order of magnitude from the theoretical upper bound ε' guaranteed by Corollary 5.1. Surprisingly, for two datasets (EmailEu and Mathoverflow) the maximum deviation and the upper bound ε' are even of the same order of magnitude. Therefore we can conclude that the guarantees provided by Corollary 5.1 are often very sharp. In addition, ONBRA's accuracy is demonstrated by the fact that the deviation between the actual temporal betweenness centrality value of a node and its estimate obtained using ONBRA is about one order of magnitude less than the actual value, as we show in Figure 5.3b and Figure 5.4 (in Section 5.6).

Finally, we show in Table 5.3 that on the large datasets Askubuntu, Sms, and Superuser the exact algorithm [Buß et al., 2020] is not able to conclude the computation on our machine (denoted with ✗) since it requires more than 1008GB of RAM. Instead, ONBRA provides estimates of the temporal betweenness centrality values in less than 42K (sec) and 21 GB of RAM memory.

To conclude, ONBRA is able to estimate the temporal betweenness centrality with high accuracy providing rigorous and sharp guarantees, while significantly reducing the computational resources required by the exact algorithm by Buß et al. [2020].

5.5.4 ONBRA on RTW-based Betweenness

In this section we discuss how ONBRA can be used to analyze real world networks by estimating the centrality values of the nodes for the temporal betweenness under the RTW criterion.

We used the FB network, on which we computed a tight approximation of the temporal betweenness values ($\varepsilon' < 10^{-4}$) of the nodes for different

values of δ, i.e., δ=1 day, δ=15 days, and δ=1 month. For δ=1 day, we found only 4 nodes with temporal betweenness value different from 0, which is surprising since it highlights that the information spreading across wall posts through RTWs in 2008 on Facebook required more than 1 day of time between consecutive interactions (i.e., slow spreading). We present the results for the other values of δ in Figures 5.3c and 5.3d, comparing them to the (exact) STP-based betweenness. Interestingly, 15 days are still not sufficient to capture most of the betweenness values based on STPs of the different nodes, while with δ=1 month the betweenness values are much closer to the STP-based values. While this behaviour is to be expected with increasing δ, finding such values of δ helps to better characterize the dynamics over the network. To conclude, ONBRA also enables novel analyses that cannot otherwise be performed with existing tools.

5.6 Additional Material

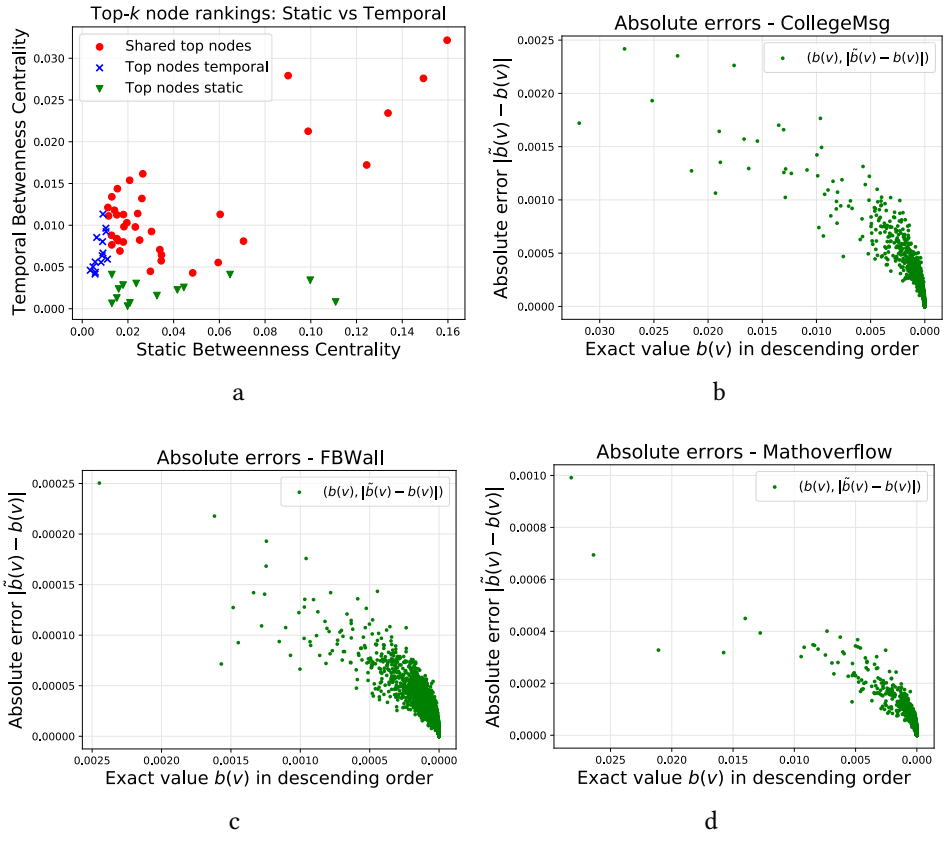


Figure 5.4: (5.4a): static and temporal betweenness values of the top-50 ranked nodes of the dataset HS; (5.4b),(5.4c), and (5.4d): respectively for datasets CollegeMsg, FBWall, and Mathoverflow, the deviations (or absolute errors) $|\tilde{b}(v) - b(v)|$ between the estimates $\tilde{b}(v)$ and the actual values $b(v)$ of the temporal betweenness centrality, for decreasing order of $b(v)$.

Chapter 6

Conclusion

In this chapter we summarize the contributions achieved in this thesis, by commenting each chapter separately. We then conclude with some final remarks and possible extensions for the works and topics presented in this thesis.

In Chapter 3 we addressed the problem of computing a count of a temporal motif through exact and approximate algorithms. To this end, we introduced a novel exact algorithm than can be used, combined with existing state-of-the-art algorithms for exact enumeration, to parallelize the computation of temporal motif counts. This algorithm renders therefore practical the *exact* computation of a count of a temporal motif in a temporal network. We then extensively discussed the problem of computing an *approximate count* of a temporal motif introducing PRESTO, a simple yet practical randomized algorithm for the rigorous approximation of temporal motif counts. We introduced two variants of PRESTO that differ on how the sampled subnetworks are collected inside the algorithm, and for each variant we derived bounds on the number of samples for the algorithms to concentrate within desired accuracy with controlled probability. Our extensive experimental evaluation shows that our proposed exact algorithm enables the computation of temporal motif counts when these cannot be computed with existing state-of-the-art exact algorithms. Our algorithm is able in fact to efficiently save significant amount of resources, both time and memory, which coupled with its parallel implementation provides a novel fundamental tool for the exact computation of motif counts. We then evaluated our randomized approximate algorithm PRESTO, showing that it provides more accurate results on large real-world networks and is much more scalable than the state-of-the-art sampling approaches. In particular, PRESTO often uses a significantly smaller amount of resources when compared with existing state-of-the-art approximate approaches providing more accurate estimates, and is able to parallelize efficiently its computation in a parallel environment. With our algorithms we therefore contributed towards enabling the practical computation of arbitrary temporal motif counts in a temporal network.

In Chapter 4, starting from an existing gap in the literature, we introduced a novel problem that requires computing the counts of multiple tem-

poral motifs, all sharing the same static topology. This is motivated by the fact that without prior knowledge on the ordering of a temporal motif, it is common to count all the motifs as in the problem we proposed. We then introduced ODEN, our sampling algorithm to obtain rigorous, high-quality, probabilistic approximations of the counts of multiple motifs with the same static topology in large temporal networks. ODEN samples edges on the static projection of the input temporal network, and carefully identifies and weights each instance of the temporal motifs containing the sampled edge, then ODEN employs concentration bounds to provide rigorous probabilistic guarantees on its output. Our experimental evaluation shows that ODEN allows to accurately estimate counts of hundreds of temporal motifs in large networks in a fraction of the time required by state-of-the-art algorithms, since these approaches do not account for the novel problem we introduced. We believe that our algorithm ODEN will be of practical interest in the analysis of temporal networks, complementing many of the existing tools and helping in understanding complex networked systems and their patterns.

In Chapter 5 we presented ONBRA, the first algorithm that provides high-quality approximations of the temporal betweenness centrality values of the nodes in a temporal network, with rigorous probabilistic guarantees. ONBRA is a sampling algorithm that collects randomly sampled pairs of nodes, and computes all the optimal paths (or walks) connecting the sampled nodes. We adapted ONBRA to work under two different optimality criteria for the paths (and walks) on which the temporal betweenness centrality is defined: shortest paths and shortest δ-restless temporal walks (STP, RTW) criteria. To provide its rigorous probabilistic guarantees ONBRA employs advanced tail bounds based on the *empirical* variance of the estimates considered. To the best of our knowledge, ONBRA is the first algorithm enabling a practical computation of the temporal betweenness centrality, especially under the RTW criterion. Our experimental evaluation on real world networks shows that ONBRA provides high-quality estimates with tight guarantees, while remarkably reducing the computational costs compared to the state-of-the-art by Buß et al. [2020], enabling analyses that would not otherwise be possible to perform. In fact, ONBRA thanks to its scalability can address the computation of the temporal betweenness centrality on datasets where such centrality measure cannot be computed with existing techniques, providing therefore a fundamental tool for processing temporal networks.

We now discuss possible future directions for the various chapters.

For Chapter 3 there are several interesting directions for future research, such as improving the approximate algorithms by dealing with datasets with different edge distributions and temporal motif distributions over temporal networks. Additionally, the theoretical guarantees of our proposed algorithm PRESTO can be improved by finding novel ways to bound specific quantities used for proving the concentration of PRESTO's final estimator.

For Chapter 4 future directions include, devising better edge probability distributions for ODEN and choosing such distribution based on the charac-

teristics of the dataset, since different datasets can have very different temporal edges distributions (e.g., with skewed behaviours as seen in Chapter 3) and, thus, there may not exist a unique distribution that is effective for all temporal networks. Another direction of future research is the derivation of improved bounds for the number of samples required by ODEN, using for example statistical learning theory concepts, such as Pseudodimensions or Rademacher averages. In fact it has been shown in many data mining problems that leveraging more advanced techniques, such as based on VC-dimension of Rademacher averages can improve the results based on union bounds. Even if for the problem proposed in this chapter we do not expect that such results can improve significantly the sample sizes we obtained, given the complicated structure of the problem. Another interesting direction is to extend our work on coloured temporal networks, where the number of temporal motifs combinatorially depend on the number of colours of the networks.

For Chapter 5 there are several interesting directions that could be explored in the future, such as dealing with different optimality criteria for the paths or walks considered, and employing sharper concentration inequalities to provide tighter guarantees on the quality of the estimates. In particular, leveraging statistical learning theory concepts that have been practically used for the problem of static betweenness [Pellegrina and Vandin, 2021, Cousins et al., 2021], even though in some cases the bound we propose are already close to optimum, as shown in practice by our experimental evaluation.

Bibliography

The CAIDA UCSD Anonymized Internet Traces – February 17, 2011. URL https://www.caida.org/data/passive/passive_dataset.xml.

Iftikhar Ahmad, Muhammad Usman Akhtar, Salma Noor, and Ambreen Shahnaz. Missing link prediction using common neighbor and centrality based parameterized algorithm. 10(1), jan 2020. doi: 10.1038/s41598-019-57304-y.

Nesreen K Ahmed, Nick Duffield, Jennifer Neville, and Ramana Kompella. Graph sample and hold: A framework for big-graph analytics. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1446–1455, 2014.

Ahmad Alsayed and Desmond J. Higham. Betweenness in time dependent networks. 72:35–48, mar 2015. doi: 10.1016/j.chaos.2014.12.009.

Paolo Bajardi, Alain Barrat, Fabrizio Natale, Lara Savini, and Vittoria Colizza. Dynamical patterns of cattle trade movements. *PLoS ONE*, 6(5): e19869, may 2011. doi: 10.1371/journal.pone.0019869.

V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *Advances in Data Analysis and Classification, 2011. Volume 5, Number 2, 129-145*, October 2003.

Federico Battiston, Vincenzo Nicosia, Mario Chavez, and Vito Latora. Multilayer motif analysis of brain networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(4):047404, 2017.

Jeffrey Baumes, Mark K. Goldberg, Mukkai S. Krishnamoorthy, Malik Magdon-Ismail, and Nathan Preston. Finding communities by clustering a graph into overlapping subgraphs. In Nuno Guimarães and Pedro T. Isaías, editors, *AC 2005, Proceedings of the IADIS International Conference on Applied Computing, Algarve, Portugal, February 22-25, 2005, Volume 1*, pages 97–104. IADIS, 2005.

Caleb Belth, Xinyi Zheng, and Danai Koutra. Mining persistent activity in continually evolving networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, jul 2020. doi: 10.1145/3394486.3403136.

George Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45, mar 1962. doi: 10.1080/01621459.1962.10482149.

Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

Hanjo D Boekhout, Walter A Kosters, and Frank W Takes. Efficiently counting complex multilayer temporal motifs in large-scale networks. *Computational Social Networks*, 6(1):1–34, 2019.

Stephen P. Borgatti and Martin G. Everett. A graph-theoretic perspective on centrality. *Soc. Networks*, 28(4):466–484, 2006. doi: 10.1016/j.socnet.2005.11.005.

Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration Inequalities*. Oxford University Press, feb 2013. doi: 10.1093/acprof:oso/9780199535255.001.0001.

Ulrik Brandes. A faster algorithm for betweenness centrality. 25(2):163–177, jun 2001. doi: 10.1080/0022250x.2001.9990249.

Ulrik Brandes and Christian Pich. Centrality estimation in large networks. *Int. J. Bifurc. Chaos*, 17(7):2303–2318, 2007. doi: 10.1142/S0218127407018403.

Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. Counting graphlets: Space vs time. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 557–566, 2017.

Marco Bressan, Stefano Leucci, and Alessandro Panconesi. Motivo. *Proceedings of the VLDB Endowment*, 12(11):1651–1663, jul 2019. doi: 10.14778/3342263.3342640.

Sebastian Buß, Hendrik Molter, Rolf Niedermeier, and Maciej Rymar. Algorithmic aspects of temporal betweenness. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 2084–2092. ACM, 2020. doi: 10.1145/3394486.3403259.

Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, jun 2021. doi: 10.1007/s00453-021-00831-w.

Matteo Ceccarello, Carlo Fantozzi, Andrea Pietracaprina, Geppino Pucci, and Fabio Vandin. Clustering uncertain graphs. *Proceedings of the VLDB Endowment*, 11(4):472–484, dec 2017. doi: 10.1145/3186728.3164143.

Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*. ACM Press, 2011. doi: 10.1145/2020408.2020579.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. ACM Press, 1971. doi: 10.1145/800157.805047.

Cyrus Cousins, Chloe Wohlgemuth, and Matteo Riondato. Bavarian: Betweenness centrality approximation with variance-aware rademacher averages. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 196–206. ACM, 2021. doi: 10.1145/3447548.3467354.

Kousik Das, Sovan Samanta, and Madhumangal Pal. Study on centrality measures in social networks: a survey. *Soc. Netw. Anal. Min.*, 8(1):13, 2018. doi: 10.1007/s13278-018-0493-2.

Lorenzo De Stefani, Erisa Terolli, and Eli Upfal. Tiered sampling: An efficient method for approximate counting sparse motifs in massive graph streams. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 776–786. IEEE, 2017.

Ying Ding. Scientific collaboration and endorsement: Network analysis of coauthorship and citation networks. *Journal of Informetrics*, 5(1):187–203, jan 2011. doi: 10.1016/j.joi.2010.10.008.

Jessica Enright, Kitty Meeks, and Hendrik Molter. Counting temporal paths. February 2022.

David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Graph Algorithms and Applications I*, pages 283–309. WORLD SCIENTIFIC, mar 2002. doi: 10.1142/9789812777638_0014.

Santo Fortunato. Community detection in graphs. 486(3-5):75–174, feb 2010. doi: 10.1016/j.physrep.2009.11.002.

Linton C. Freeman. A set of measures of centrality based on betweenness. 40(1):35, mar 1977. doi: 10.2307/3033543.

Linton C. Freeman. Centrality in social networks conceptual clarification. 1 (3):215–239, jan 1978. doi: 10.1016/0378-8733(78)90021-7.

Dongqi Fu, Dawei Zhou, and Jingrui He. Local motif clustering on time-evolving graphs. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 390–400. ACM, 2020. doi: 10.1145/3394486.3403081.

Zhongqiang Gao, Chuanqi Cheng, Yanwei Yu, Lei Cao, Chao Huang, and Junyu Dong. Scalable motif counting for large-scale temporal graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, may 2022. doi: 10.1109/icde53745.2022.00244.

Laetitia Gauvin, Mathieu Génois, Márton Karsai, Mikko Kivelä, Taro Takaguchi, Eugenio Valdano, and Christian L. Vestergaard. Randomized reference models for temporal networks. June 2018.

M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12): 7821–7826, jun 2002. doi: 10.1073/pnas.122653799.

Saket Gurukar, Sayan Ranu, and Balaraman Ravindran. COMMIT. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, may 2015. doi: 10.1145/2723372.2737791.

Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. $\text{Turbo}_{\text{iso}}$: towards ultrafast and robust subgraph isomorphism search in large graph databases. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 337–348. ACM, 2013. doi: 10.1145/2463676.2465300.

Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms. *CoRR*, abs/2102.11169, 2021. URL https://arxiv.org/abs/2102.11169.

Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58 (301):13–30, March 1963. ISSN 0162-1459, 1537-274X. doi: 10.1080/01621459.1963.10500830. URL http://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500830.

Petter Holme and Jari Saramäki. Temporal networks. *Physics Reports*, 519 (3):97–125, oct 2012. doi: 10.1016/j.physrep.2012.03.001.

Petter Holme and Jari Saramäki, editors. *Temporal Network Theory*. Springer International Publishing, 2019. doi: 10.1007/978-3-030-23495-9.

Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. 65(5):056109, may 2002. doi: 10.1103/physreve.65.056109.

J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing - STOC '74*. ACM Press, 1974. doi: 10.1145/800119.803896.

Y. Hulovatyy, H. Chen, and T. Milenković. Exploring the structure and function of temporal networks with dynamic graphlets. *Bioinformatics*, 31(12): i171–i180, jun 2015. doi: 10.1093/bioinformatics/btv227.

Ali Jazayeri and Christopher C Yang. Motif discovery algorithms in static and temporal networks: A survey. *Journal of Complex Networks*, 8(4), aug 2020. doi: 10.1093/comnet/cnaa031.

Alpár Jüttner and Péter Madarasi. VF2++ - an improved subgraph isomorphism algorithm. *Discret. Appl. Math.*, 242:69–81, 2018. doi: 10.1016/j.dam.2018.02.018.

Hyoungshick Kim and Ross Anderson. Temporal node centrality in complex networks. 85(2):026107, feb 2012. doi: 10.1103/physreve.85.026107.

Dirk Koschützki and Falk Schreiber. Centrality analysis methods for biological networks and their application to gene regulatory networks. 2: GRSB.S702, jan 2008. doi: 10.4137/grsb.s702.

Chrysanthi Kosyfaki, Nikos Mamoulis, Evaggelia Pitoura, and Panayiotis Tsaparas. Flow motifs in interaction networks. October 2018.

L. Kovanen, K. Kaski, J. Kertesz, and J. Saramaki. Temporal motifs reveal homophily, gender-specific patterns, and group talk in call sequences. *Proceedings of the National Academy of Sciences*, 110(45):18070–18075, oct 2013. doi: 10.1073/pnas.1307941110.

Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. Temporal motifs in time-dependent networks. *CoRR*, abs/1107.5646, 2011. URL http://arxiv.org/abs/1107.5646.

Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*. ACM Press, 2006. doi: 10.1145/1150402.1150476.

Rohit Kumar and Toon Calders. 2scent. *Proceedings of the VLDB Endowment*, 11(11):1441–1453, jul 2018. doi: 10.14778/3236187.3269460.

Johannes Köbler, Uwe Schöning, and Jacobo Torán. Introduction. In *The Graph Isomorphism Problem*, pages 1–4. Birkhäuser Boston, 1993. doi: 10.1007/978-1-4612-0333-9_1.

Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, 8(1), oct 2018. doi: 10.1007/s13278-018-0537-7.

Min-Joong Lee, Jungmin Lee, Jaimie Yejean Park, Ryan Hyun Choi, and Chin-Wan Chung. QUBE: a quick algorithm for updating betweenness

centrality. In Alain Mille, Fabien Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 351–360. ACM, 2012. doi: 10.1145/2187836.2187884.

Sune Lehmann. Fundamental structures in dynamic communication networks. *CoRR*, abs/1907.09966, 2019. URL http://arxiv.org/abs/1907.09966.

Da Lei, Xuewu Chen, Long Cheng, Lin Zhang, Satish V. Ukkusuri, and Frank Witlox. Inferring temporal motifs for travel pattern analysis using large scale smart card data. *Transportation Research Part C: Emerging Technologies*, 120:102810, nov 2020. doi: 10.1016/j.trc.2020.102810.

Bar Light. Concentration inequalities using higher moments information. June 2020.

Paul Liu, Austin R. Benson, and Moses Charikar. Sampling methods for counting temporal motifs. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, page 294–302, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359405. doi: 10.1145/3289600.3290988. URL https://doi.org/10.1145/3289600.3290988.

Penghang Liu, Valerio Guarrasi, and A. Erdem Sariyuce. Temporal network motifs: Models, limitations, evaluation. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021. doi: 10.1109/tkde.2021.3077495.

Xiaoming Liu, Johan Bollen, Michael L. Nelson, and Herbert Van de Sompel. Co-authorship networks in the digital library research community. *Inf. Process. Manag.*, 41(6):1462–1480, 2005. doi: 10.1016/j.ipm.2005.03.012.

Antonio Longa, Giulia Cencetti, Bruno Lepri, and Andrea Passerini. An efficient procedure for mining egocentric temporal motifs. *Data Mining and Knowledge Discovery*, 36(1):355–378, nov 2021. doi: 10.1007/s10618-021-00803-2.

Patrick Mackey, Katherine Porterfield, Erin Fitzhenry, Sutanay Choudhury, and George Chin Jr. A chronological edge-driven approach to temporal subgraph isomorphism. January 2018.

Shmoolik Mangan and Uri Alon. Structure and function of the feed-forward loop network motif. *Proceedings of the National Academy of Sciences*, 100 (21):11980–11985, 2003.

Naoki Masuda and Renaud Lambiotte. *A Guide to Temporal Networks*. WORLD SCIENTIFIC (EUROPE), apr 2016. doi: 10.1142/q0033.

Andreas Maurer and Massimiliano Pontil. Empirical bernstein bounds and sample variance penalization, 2009.

Othon Michail. An Introduction to Temporal Graphs: An Algorithmic Perspective. *Internet Mathematics*, 12(4):239–280, apr 2016. doi: 10.1080/15427951.2016.1177801.

Tijana Milenković and Nataša Pržulj. Uncovering biological network function via graphlet degree signatures. *Cancer Informatics*, 6:CIN.S680, jan 2008. doi: 10.4137/cin.s680.

R. Milo. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, oct 2002. doi: 10.1126/science.298.5594.824.

R. Milo. Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–1542, mar 2004. doi: 10.1126/science.1089167.

Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

Mark Newman. *Networks*. Oxford University Press, mar 2010. doi: 10.1093/acprof:oso/9780199206650.001.0001.

Raj Kumar Pan and Jari Saramäki. Path lengths, correlations, and centrality in temporal networks. *Physical Review E*, 84(1):016105, jul 2011. doi: 10.1103/physreve.84.016105.

Pietro Panzarasa, Tore Opsahl, and Kathleen M. Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, may 2009. doi: 10.1002/asi.21015.

Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. Motifs in temporal networks. In Maarten de Rijke, Milad Shokouhi, Andrew Tomkins, and Min Zhang, editors, *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017*, pages 601–610. ACM, 2017. doi: 10.1145/3018661.3018731.

Ha-Myung Park, Francesco Silvestri, U Kang, and Rasmus Pagh. Mapreduce triangle enumeration with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1739–1748, 2014.

Noujan Pashanasangi and C. Seshadhri. Efficiently counting vertex orbits of all 5-vertex subgraphs, by EVOKE. *CoRR*, abs/1911.10616, 2019. doi: 10.1145/3336191.3371773. URL http://arxiv.org/abs/1911.10616.

Noujan Pashanasangi and C. Seshadhri. Faster and generalized temporal triangle counting, via degeneracy ordering. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. ACM, aug 2021. doi: 10.1145/3447548.3467374.

Tiago P. Peixoto and Laetitia Gauvin. Change points, memory and epidemic spreading in temporal networks. 8(1), oct 2018. doi: 10.1038/s41598-018-33313-1.

Leonardo Pellegrina and Fabio Vandin. SILVAN: estimating betweenness centralities with progressive sampling and non-uniform rademacher bounds. *CoRR*, abs/2106.03462, 2021. URL https://arxiv.org/abs/2106.03462.

Alexandra Porter, Baharan Mirzasoleiman, and Jure Leskovec. Analytical models for motifs in temporal networks. In *Companion Proceedings of the Web Conference 2022*. ACM, apr 2022. doi: 10.1145/3487553.3524669.

N. Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, jan 2007. doi: 10.1093/bioinformatics/btl301.

Natasa Pržulj, Derek G Corneil, and Igor Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 2004.

Xuguang Ren and Junhu Wang. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. *Proc. VLDB Endow.*, 8(5): 617–628, 2015. doi: 10.14778/2735479.2735493. URL http://www.vldb.org/pvldb/vol8/p617-ren.pdf.

Pedro Ribeiro, Pedro Paredes, Miguel EP Silva, David Aparicio, and Fernando Silva. A survey on subgraph counting: concepts, algorithms and applications to network motifs and graphlets. *arXiv preprint arXiv:1910.13011*, 2019.

Pedro Ribeiro, Pedro Paredes, Miguel E. P. Silva, David Aparicio, and Fernando Silva. A survey on subgraph counting. *ACM Computing Surveys*, 54(2):1–36, mar 2022. doi: 10.1145/3433652.

Matteo Riondato and Evgenios M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. *Data Min. Knowl. Discov.*, 30 (2):438–475, 2016. doi: 10.1007/s10618-015-0423-0.

Matteo Riondato and Eli Upfal. ABRA: approximating betweenness centrality in static and dynamic graphs with rademacher averages. *ACM Trans. Knowl. Discov. Data*, 12(5):61:1–61:38, 2018. doi: 10.1145/3208351.

Matteo Riondato and Fabio Vandin. MiSoSouP: Mining Interesting Subgroups with Sampling and Pseudodimension. *ACM Transactions on Knowledge Discovery from Data*, 14(5):1–31, jun 2020. doi: 10.1145/3385653.

Ryan A. Rossi, Nesreen K. Ahmed, Aldo Carranza, David Arbour, Anup Rao, Sungchul Kim, and Eunyee Koh. Heterogeneous graphlets. *ACM Transactions on Knowledge Discovery from Data*, 15(1):1–43, jan 2021. doi: 10.1145/3418773.

Maciej Rymar, Hendrik Molter, André Nichterlein, and Rolf Niedermeier. Towards classifying the polynomial-time solvability of temporal betweenness centrality. In Lukasz Kowalik, Michal Pilipczuk, and Pawel Rzazewski, editors, *Graph-Theoretic Concepts in Computer Science - 47th International Workshop, WG 2021, Warsaw, Poland, June 23-25, 2021, Revised Selected Papers*, volume 12911 of *Lecture Notes in Computer Science*, pages 219–231. Springer, 2021. doi: 10.1007/978-3-030-86838-3_17. URL https://doi.org/10.1007/978-3-030-86838-3_17.

Diego Santoro and Ilie Sarpe. ONBRA: Rigorous estimation of the temporal betweenness centrality in temporal networks. In *Proceedings of the ACM Web Conference 2022*. ACM, apr 2022. doi: 10.1145/3485447.3512204.

Ilie Sarpe and Fabio Vandin. Presto: Simple and scalable sampling techniques for the rigorous approximation of temporal motif counts. *SIAM International Conference on Data Mining*, 2021a. doi: 10.1137/1.9781611976700.17.

Ilie Sarpe and Fabio Vandin. odeN: Simultaneous Approximation of Multiple Motif Counts in Large Temporal Networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. ACM, oct 2021b. doi: 10.1145/3459637.3482459.

Alice C. Schwarze and Mason A. Porter. Motifs for processes on networks. July 2020.

Shai S. Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 31(1):64–68, apr 2002. doi: 10.1038/ng881.

Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In David A. Van Dyk and Max Welling, editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, volume 5 of *JMLR Proceedings*, pages 488–495. JMLR.org, 2009. URL http://proceedings.mlr.press/v5/shervashidze09a.html.

Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. Triest: Counting local and global triangles in fully dynamic streams with fixed memory size. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(4):1–50, 2017.

Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 687–696, 2007.

Shixuan Sun, Xibo Sun, Yulin Che, Qiong Luo, and Bingsheng He. Rapidmatch: a holistic approach to subgraph query processing. *Proceedings of the VLDB Endowment*, 14:176–188, 2020. ISSN 2150-8097. doi: 10.14778/3425879.3425888.

Nishil Talati, Haojie Ye, Sanketh Vedula, Kuan-Yu Chen, Yuhan Chen, Daniel Liu, Yichao Yuan, David Blaauw, Alex Bronstein, Trevor Mudge, and Ronald Dreslinski. Mint: An accelerator for mining temporal motifs. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, oct 2022. doi: 10.1109/micro56248.2022.00089.

Suhas Thejaswi, Juho Lauri, and Aristides Gionis. Restless reachability problems in temporal graphs. October 2020.

Surya T. Tokdar and Robert E. Kass. Importance sampling: a review. *WIREs Computational Statistics*, 2(1):54–60, dec 2009. doi: 10.1002/wics.56.

Ioanna Tsalouchidou, Ricardo Baeza-Yates, Francesco Bonchi, Kewen Liao, and Timos Sellis. Temporal betweenness centrality in dynamic graphs. *Int. J. Data Sci. Anal.*, 9(3):257–272, 2020. doi: 10.1007/s41060-019-00189-x.

Charalampos E Tsourakakis, U Kang, Gary L Miller, and Christos Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 837–846, 2009.

Kun Tu, Jian Li, Don Towsley, Dave Braines, and Liam D Turner. Network classification in temporal networks using motifs. *arXiv preprint arXiv:1807.03733*, 2018.

Kun Tu, Jian Li, Don Towsley, Dave Braines, and Liam D. Turner. gl2vec. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM, aug 2019. doi: 10.1145/3341161.3342908.

Johan Ugander, Lars Backstrom, and Jon Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1307–1318, 2013.

Bimal Viswanath, Alan Mislove, Meeyoung Cha, and P. Krishna Gummadi. On the evolution of user interaction in facebook. In Jon Crowcroft and Balachander Krishnamurthy, editors, *Proceedings of the 2nd ACM Workshop on Online Social Networks, WOSN 2009, Barcelona, Spain, August 17, 2009*, pages 37–42. ACM, 2009. doi: 10.1145/1592665.1592675.

Jingjing Wang, Yanhao Wang, Wenjun Jiang, Yuchen Li, and Kian-Lee Tan. Efficient sampling algorithms for approximate temporal motif counting. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. ACM, oct 2020. doi: 10.1145/3340531.3411862.

Virginia Vassilevska Williams, Joshua R. Wang, Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, dec 2014. doi: 10.1137/1.9781611973730.111.

Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proceedings of the VLDB Endowment*, 7(9):721–732, may 2014. doi: 10.14778/2732939.2732945.

Jiajing Wu, Jieli Liu, Weili Chen, Huawei Huang, Zibin Zheng, and Yan Zhang. Detecting mixing services via mining bitcoin transaction network with hybrid motifs. *arXiv preprint arXiv:2001.05233*, 2020.

Stefan Wuchty and Peter F. Stadler. Centers of complex networks. 223(1): 45–53, jul 2003. doi: 10.1016/s0022-5193(03)00071-7.

B. Bui Xuan, A. Ferreira, and A. Jarry. Cimputing Shortest, Fastest, Aand Foremost Journeys In Dynamic Networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, apr 2003. doi: 10.1142/s0129054103001728.

Erjia Yan and Ying Ding. Applying centrality measures to impact analysis: A coauthorship network analysis. *J. Assoc. Inf. Sci. Technol.*, 60(10):2107–2118, 2009. doi: 10.1002/asi.21128.

Ömer Nebil Yaveroğlu, Noël Malod-Dognin, Darren Davis, Zoran Levnajic, Vuk Janjic, Rasa Karapandza, Aleksandar Stojmirovic, and Nataša Pržulj. Revealing the hidden language of complex networks. *Scientific reports*, 4: 4547, 2014.

Hao Yin, Austin R. Benson, and Jure Leskovec. Higher-order clustering in networks. *Physical Review E*, 97(5):052306, may 2018. doi: 10.1103/physreve.97.052306.

Shuo Yu, Yufan Feng, Da Zhang, Hayat Dino Bedru, Bo Xu, and Feng Xia. Motif discovery in networks: A survey. *Computer Science Review*, 37: 100267, 2020.

Xiang Zhang, Gong Cheng, and Yuzhong Qu. Ontology summarization based on rdf sentence graph. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 707–716. ACM, 2007. doi: 10.1145/1242572.1242668.

Qiankun Zhao, Yuan Tian, Qi He, Nuria Oliver, Ruoming Jin, and Wang-Chien Lee. Communication motifs. In *Proceedings of the 19th ACM international conference on Information and knowledge management - CIKM '10*. ACM Press, 2010. doi: 10.1145/1871437.1871694.

Songfeng Zheng. An improved bennett's inequality. *Communications in Statistics - Theory and Methods*, 47(17):4152–4159, oct 2017. doi: 10.1080/03610926.2017.1367818.

Bo Zong, Xusheng Xiao, Zhichun Li, Zhenyu Wu, Zhiyun Qian, Xifeng Yan, Ambuj K. Singh, and Guofei Jiang. Behavior query discovery in system-generated temporal graphs. *Proceedings of the VLDB Endowment*, 9(4): 240–251, dec 2015. doi: 10.14778/2856318.2856320.