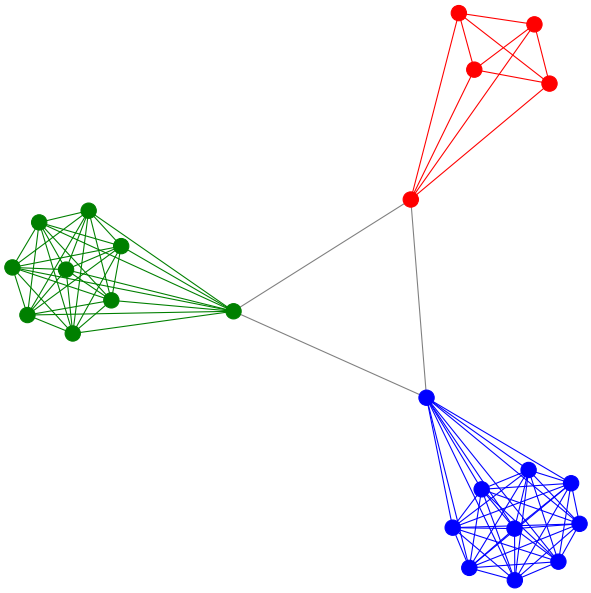
CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Parallel and Distributed Motif Discovery in Temporal Networks

A feasibility study applying thread parallelism and community structure

Master's thesis in Computer Science and Engineering

STEFAN MARTON

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Master's thesis 2024

Parallel and Distributed Motif Discovery in Temporal Networks

A feasibility study applying thread parallelism and community structure

STEFAN MARTON

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Parallel and Distributed Motif Discovery in Temporal Networks
A feasibility study applying thread parallelism and community structure
STEFAN MARTON

Supervisor: Ahmed Ali-Eldin Hassan, Department of Computer Science and Engineering
Examiner: Peter Damaschke, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An undirected graph separated into clusters.

Typeset in LaTeX
Gothenburg, Sweden 2024

Parallel and Distributed Motif Discovery in Temporal Networks
A feasibility study applying thread parallelism and community structure
STEFAN MARTON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Temporal networks are used to model complex systems in topics such as epidemiology, finance, and computer networks. Motifs are subgraphs of a directed graph that are representative of the structure of that particular graph. Motifs have been extended to temporal networks. Motif discovery is a computationally hard problem; in fact sub-problems are NP-hard problems. In this thesis, we explore state-of-the-art temporal network motif discovery algorithms, and how they can be parallelized on a multi-threaded system and distributed across multiple systems. We select Kovanen's definition of temporal network motif. We implement a simple approach to thread parallelism to demonstrate the potential for parallelism of the algorithm, and find that the parallelizable proportion $p > 0.89$, which implies great potential for parallelism. We utilize community structure of the graph the temporal network represents to divide the network into work packages for distributed computation. In doing so, we encounter and report numerous challenges in distribution of the exact solution approach.

Keywords: temporal network, motif, temporal motif, parallel computing, distributed computing.

Acknowledgements

I would like to thank my thesis supervisor, Associate Professor Ahmed Ali-Eldin Hassan, for giving me the opportunity to do research, and providing invaluable guidance throughout the research. Your patience, motivation and enthusiasm has been invaluable to this research. I am equally grateful to my examiner, Professor Peter Damaschke, who generously dedicated time and effort to review and evaluate my work. Your tireless commitment to rigor and precision has been vital in refining the content and ensuring the academic integrity of my work.
I thank Chalmers University of Technology and its dedicated faculty and staff, who have created a nurturing atmosphere conducive to learning and research. In this environment, I also met the love of my life, my wife. Special appreciation goes to Professor Ulf Assarsson. Your mentorship throughout my academic journey has truly made a difference.
I would like to thank my employer, KTC Product AB, for their support throughout the duration of my academic pursuits.
I would like to express my deepest gratitude to my dear father, Albert Marton, and mother, Bodil Marton, for your endless patience, boundless love and support.
I thank my dear wife, Lakshmi Salelkar, for always being by my side, finding ways to challenge me, contributing to my personal growth and broadening my horizons. I look forward to seeing you wear the laurel wreath and referring to you as my Doctor.
Finally, this journey has been long and arduous, and it would not have been possible for me to endure without the grace of God, and his love as demonstrated through his people.

Stefan Marton, Uppsala, March 2024

Contents

List of Figures

List of Tables

1
Introduction

Various fields of research make use of graphs as a mathematical model. For example, in the social sciences graphs have been used to study communication networks, dominance structures, and relations of influence [1]. In the field of Computer Science, a *graph* refers to a type of data structure, composed of *nodes* (or *vertices*) and *edges*, in which each edge is connected to two nodes. If the direction of the edge is significant, the graph is referred to as a *directed graph*. A subset of the edges and nodes of a graph, which in itself would form a graph, is referred to as a *subgraph*. The number of edges connected to a node is referred to as the *degree* of the node.
Temporal networks [2] are a class of data structures extending the concept of a directed graph to include information about time-dependent interactions. Temporal networks are comprised of two main components: nodes and *events*. Similar to graphs, nodes often represent objects or actors, but distinctly from graphs, events model time-dependent interactions and therefore also contain information not only about which origin node is interacting with which destination node but also about when that interaction begins and ends.
Temporal networks have been used to model diverse phenomena such as the spread of contagion between individuals over time [3], stock markets [4] and interactive WiFi users [5].
Certain types of graphs representing network structures, such as those modeling social networks or communication networks, may form clusters referred to as *communities* [6]. For such graphs, the number of edges connecting the nodes within a community is significantly larger than the number of edges connecting nodes that belong to different communities. We will refer to such graphs as *community-structured graphs*.
Network motifs are subgraphs of a particular directed graph, the shape of which recurs at a higher frequency than statistically expected [7]. Hence motifs identify structures that are significant to the graph being studied.
Multiple authors have attempted to extend the concept of network motifs to temporal networks [8]–[12]. Likewise, these definitions of a *temporal network motif* attempt to identify structures that are significant to the temporal network being studied, but they differ in their definition of what makes a particular structure a valid motif to be considered.
Discovering network motifs entails comparing the number of occurrences of a particular structure in the graph to the number of occurrences of said structure in randomly connected graphs, each *random graph* consisting of the same number of nodes, and each node with the same degree as that of the graph being studied [7]. Motif discovery is computationally hard since matching a given graph pattern to

a certain motif is in itself an NP-hard problem [13]. State-of-the-art motif discovery algorithms exploit application-specific properties of the networks under study to increase performance [13].
To the best of our knowledge, all methods proposed to find motifs in temporal networks are likewise based on the subgraph matching problem. Further temporal networks generally contain additional data compared to a non-temporal directed graph when representing the same network. Hence, exponential growth in complexity based on this additional data could make the computational time required infeasible. The subgraph matching problem is NP-hard [13], meaning it may have exponential time bounds in the size of the graph.
Attempts at defining temporal network motif have been mostly aimed towards finding a definition that allows for more efficient computation. Literature review did not reveal any proposed model of temporal network motifs that accounted for the duration of events. Further, according to Liu, Guarrasi, and Sariyuce [8] "devising an ultimate unifying model would be too ambitious due to the diverse characteristics of temporal networks," hence application-specific models, exploiting properties of the network, may both increase performance and yield more useful results [8].
Kovanen, Karsai, Kaski, *et al.* [9] define temporal network motifs in a way that reduces the number of subgraph matching problems to linear in the size of the network [9]. Technically, this does not change the complexity class of the problem, as it remains NP-hard. In practice these large exponential factors in time complexity lead to very long execution times on real hardware. This places a high demand on computational resources.
However, the clock frequencies of CPUs are no longer increasing quickly [14], leading to slowed increase in performance of classical algorithms with single-threaded execution. According to Navarro, Hitschfeld-Kahler, and Mateu "Unfortunately, sequential implementations will no longer run faster by just buying better hardware. They must be re-designed as a parallel algorithm that can scale its performance as more processors are available" [15].
Parallel algorithms may take multiple forms, but two broad categories are *parallelization* through multiple threads of execution on a single computer system and *distribution* of computation across multiple systems.
Further, temporal network datasets, depending on the domain, may be very large. For example, in September 2009, Cha, Haddadi, Benevenuto, *et al.* crawled the publicly accessible part of the Twitter social network and discovered the profiles of 54 million users, 2 billion links between their profiles, and 1.7 billion public messages [16]. Considering this dataset only represents a small part of the total number of events that may occur between users in the social network, we can see that such a dataset can grow very large, and become infeasible to process on a single computer.

1.1 Research objectives

The purpose of this study is to investigate techniques for increasing the performance of motif discovery in temporal networks, and in particular to explore the opportunities for parallelization and distribution of computation. In doing so, this could potentially benefit computer network development, edge computing, finance market

stability, and medicine.

1.2 Research scope

From our literature review, we find that the most widely used definition of temporal network motif in applied studies is that of Kovanen [9]. Hence, we will focus on analyzing and improving said algorithm through the use of parallelization and distribution techniques. Furthermore, we aim to explore the opportunities for applying the algorithm to larger datasets than is feasible on a particular single computer using distributed computing over several computers.
In the general case, it is not trivial to subdivide a temporal network into subgraphs and apply Kovanen's algorithm to these subgraphs. Therefore, we delimit our study to community-structured networks, which can be divided into communities by removing relatively few edges. The results from applying Kovanen's algorithm to these communities may, in themselves, be feasible to combine into a good approximation of the results for the entire network. Moreover, it may be possible to compute the motif statistics for the motifs crossing the minimum-cut boundary between the clusters and recombine the statistics to produce an accurate result. We aim to explore these approaches for distributing the algorithm over community-structured networks.

1.3 Research questions

In this study, the overarching research question is: *"How can we apply parallel and/or distributed computing methods to Kovanen's algorithm to reduce the execution time of discovering temporal network motifs, and/or be able to discover such motifs in larger networks?"* Owing to the broad, exploratory nature of this research question, we deconstruct it into a sequence of questions with more delimited scope as mentioned in the previous section.
RQ 1. What types of algorithmic approaches exist in the literature to parallelize or distribute network motif discovery?
RQ 2. How do temporal network motifs differ from network motifs from an algorithmic viewpoint?
RQ 3. What level of parallelism does the computational problem that Kovanen's algorithm is attempting to solve have?
RQ 4. What are the challenges in distributing temporal motif discovery?
RQ 5. How can we design a distributed algorithm, applying community detection, to allow execution on larger networks?

1.4 Thesis outline

In the following Chapter 2, we first introduce the theoretical foundation of the discussion of this thesis. This is followed by literature review to address our delimitation, RQ 1, and RQ 2 in sequence. Further, we discuss our position within the

broader literature and conclude with our findings on engaging with the literature, in the form of discussion and implications, especially in relation to RQ 4 and RQ 5. This is the point at which we depart from the literature review. In Chapter 3, we provide a brief overview of Kovanen's definition of a temporal network motif and Kovanen's algorithm for motif discovery. We then commence towards investigating the two tracks of parallel and distributed computation independently. The two tracks are addressed in two separate chapters, namely Chapter 4 focusing on addressing RQ 3, and Chapter 5 addressing RQ 4 and RQ 5. For each of these chapters, its sections will discuss the method, implementation, and results, then conclude with a discussion. Chapter 6 concludes the thesis with a discussion on how our findings relate to the broad literature and our suggestions for future work.

2
Background

In this chapter, stemming from the literature review, we first introduce the graph-theoretical constructs that enable us to discuss the computations used in the thesis. We proceed to define what it means for a network to be community-structured and briefly discuss algorithms that can be used for community discovery. Next, in the pursuit of RQ 1, we address network motif discovery, briefly explain the original algorithm, and describe algorithms of different types to parallelize and/or distribute the computation of network motif discovery. Further, we address temporal network motifs at a conceptual level and discuss why the concrete definition of the concept is problematic. Moreover, we discuss the semantics of a temporal network motif and how it relates to the network motif of Milo, Shen-Orr, Itzkovitz, *et al.* [7]. In doing so, we address RQ 2. The Background section concludes with our findings on engaging with the literature, in the form of discussion and implications.

2.1 Parallel vs distributed computing

Parallel and distributed computing both refer to ways to apply multiple computing resources simultaneously to the solution of a particular problem set, to reduce computation time.

Parallel computing is a general term, but it often refers to thread-parallel computing, in which multiple computing cores of a single processor, or multiple processors in a single computer, are applied to perform a computation collectively. This form of parallelism generally involves shared memory. This is the interpretation of parallel computing we will adopt for this study.

In contrast, according to Haas, "A distributed system consists of a finite collection of processes which communicate with one another exclusively through messages." [17] In other words, distributed computing refers to dividing a computational problem so as to simultaneously execute a computation on multiple distinct systems operating autonomously. These systems generally do not share memory and communicate through message-passing, often over a computer network.

2.2 Connected induced subgraph

Consider a graph $G = (V, E)$, where V is the set of nodes (vertices), and E the set of edges.

Definition 2.2.1 (Subgraph)**.** A graph $G' = (V', E')$ is a subgraph of G, or $G' \subseteq G$ if $V' \subseteq V$ and $E' \subseteq E$.
Definition 2.2.2 (Induced subgraph)**.** If $G' \subseteq G$ and G' contains *all* the edges $(x, y) \in E$ with $x, y \in V'$, then G' is an *induced subgraph* of G.
In other words an *induced subgraph* contains all the edges it could, given the set of nodes, while still being a subgraph.
Definition 2.2.3 (Connected graph)**.** G is *connected* if it is non-empty and any two of its vertices are linked by a path in G.
In other words, $E \neq \emptyset$ and between any pair of nodes $v_i, v_j \in V$, v_j must be reachable from v_i, by following edges in E.
Definition 2.2.4 (Connected Induced Subgraph (CIS))**.** A *CIS* is an *induced subgraph* that is also *connected*.

2.3 Graph isomorphism

Consider two graphs, G_1 and G_2.
Definition 2.3.1 (Isomorphic graph)**.** G_1 and G_2 are isomorphic if there exists a one-to-one mapping of the vertices of G_1 onto the vertices of G_2 such that adjacency is preserved.
For a more formal definition, we introduce notation.
Definition 2.3.2 ($\simeq$)**.** Let $G_1 \simeq G_2$ mean that G_1 is *isomorphic* to G_2.
Definition 2.3.3 ($V(G)$ and $E(G)$)**.** Let the nodes and edges of a graph G be $V(G), E(G)$ respectively, i.e.: $G = (V(G), E(G))$.
Lemma 2.3.1 (π)**.** *Let π be a bijective function $\pi : V(G_1) \to V(G_2)$.*
Definition 2.3.4 ($G_1 \simeq G_2$)**.**

$$G_1 \simeq G_2 \;\Leftrightarrow\; \exists \pi.\, \forall u, v \in V(G_1).\; ((u, v) \in E(G_1) \Leftrightarrow (\pi(u), \pi(v)) \in E(G_2))$$

Lemma 2.3.2. *$\simeq$ is commutative, i.e.: $G_1 \simeq G_2 \Leftrightarrow G_2 \simeq G_1$.*
Definition 2.3.5 (Isomorphism)**.** If $G_1 \simeq G_2$ then G_1 is an *isomorphism* of G_2.
Remark. And vice versa, since $G_2 \simeq G_1$ (Lemma 2.3.2).

2.3.1 Automorphism groups

Definition 2.3.6 (Automorphism)**.** An *automorphism*, or a *symmetry*, of a graph G is an isomorphism from G to G itself.
Roughly, G' is an automorphism of G if $G' \simeq G, V(G') = V(G)$.
Definition 2.3.7 (Automorphism group)**.** An *automorphism group* of G is a mathematical *group* where the elements is the set of all permutations of $V(G)$ which form automorphisms.
For a formal definition, we introduce notation.
Lemma 2.3.3 (σ)**.** *Let σ be a bijective function $\sigma : V(G) \to V(G)$, representing a permutation of the nodes.*
Definition 2.3.8 ($\mathrm{Sym}(G)$)**.** Let the automorphism group of graph G be $\mathrm{Sym}(G)$.
We can then define the relationship between the individual automorphism and its automorphism group as:

Definition 2.3.9.

$$G' \text{ is an automorphism of } G \Leftrightarrow$$
$$\exists \sigma \in \mathrm{Sym}(G).\ \forall u, v \in V(G).\ ((u,v) \in E(G) \Leftrightarrow (\sigma(u), \sigma(v)) \in E(G'))$$

2.3.2 Time complexity of graph isomorphism

The problem of determining whether $G_1 \simeq G_2$, given $|V(G_1)| = |V(G_2)|$, $|E(G_1)| = |E(G_2)|$, is in complexity class NP [18]. According to Grohe and Schweitzer, "Determining the precise computational complexity of [graph isomorphism] has been regarded a major open problem in theoretical computer science" [18]. Babai proposed an algorithm of quasi-polynomial time complexity $\left(\exp\left((\log n)^{O(1)}\right)\right)$ [19]. The Babai algorithm is not considered practically useful [18], due in part to its complexity of implementation, and in practice algorithms with exponential worst-case time bounds, but lower typical execution times are used, such as those of *Nauty* and *Bliss* [18].

2.3.3 Canonical labeling

Definition 2.3.10 (Canonical labeling)**.** Let L be a function over the domain of graphs, with arbitrary codomain. Let G, G' be graphs. If $L(G) = L(G') \Leftrightarrow G \simeq G'$ then L is a *canonical labeling*.
Producing a canonical labeling of graphs involves computing labels that are unique to each automorphism group and identical for each graph in the group.
Nauty [20], [21] is an open-source computer software to compute canonical labelings of graphs. We will not elaborate on the exact algorithm used. However, at its core, Nauty relies on probabilistic properties of graphs. Essentially, the algorithm backtracks through a search tree, progressively guessing a labeling and testing its validity. It prunes this search tree of already discovered automorphisms. This makes analysis of the run-time complexity of the algorithm difficult. Miyazaki found that for particular, already known to be difficult, types of graphs, Nauty explores $\Omega(n^2)$ search tree nodes. Miyazaki designed a new pathological type of graph requiring the exploration of $\Omega(c^n)$ nodes [22]. The worst-case time complexity is believed to be exponential, but for most graphs, it runs in polynomial time. According to Miyazaki, "[Nauty] is widely considered to be the fastest practical graph isomorphism package available" [22].
Bliss [23], [24] is based on the same algorithm as Nauty but introduces optimizations to make it particularly efficient on sparse graphs, allowing earlier termination of the evaluation of each search tree node. Bliss is implemented in C++.

2.4 Communities in graphs

The *density* D of a graph is defined to be $D = 1$ if all possible edges that could exist between its nodes are present in the graph, and $D = 0$ if the graph has no edges. A directed graph potentially has one edge in each direction between each distinct pair of nodes in the graph, and hence:

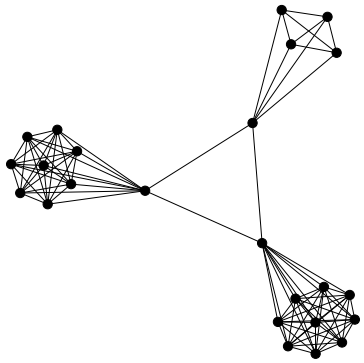


Figure 2.1: An undirected graph which has the community structure property

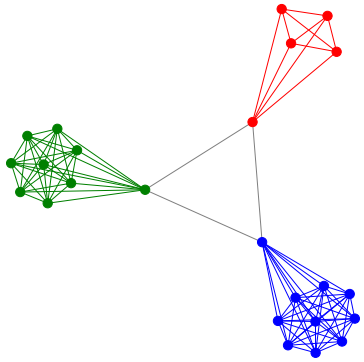


Figure 2.2: The graph from Figure 2.1, after partitioning communities into subgraphs. Each community is distinguished by assigning its member nodes a distinct color. Grey edges are not included in any community subgraph, but connect communities

Definition 2.4.1 (Graph density)**.** The density D of a directed graph $G = (V, E)$ is:

$$D = \frac{|E|}{|V|\,(|V| - 1)}$$

where $|V|$ is the number of nodes and $|E|$ is the number of edges in the graph.
A graph can be more or less *dense* than another graph, meaning that its *density* is respectively higher or lower.

Definition 2.4.2 (Community structure)**.** A graph is said to have the *community structure* property if within the graph there exist subgraphs that are more dense than the graph itself [6].
Hence, if one were to replace these subgraphs with nodes, the resulting graph would be less dense than the original graph. Fortunato and Castellano popularized referring to these subgraphs as *communities* [25].
An example of a graph with a community structure is illustrated in Figure 2.1. This graph can easily be partitioned into subgraphs such that only a few edges connect the subgraphs. In Figure 2.2 the same graph has been partitioned, and the different communities are visualized by assigning different colors to the nodes.

2.4.1 Community detection algorithms

Different types of algorithms exist for detecting communities within graphs. Two categories of algorithms to accomplish this are straightforward to describe: those which iteratively construct communities, and those which iteratively break the network down into smaller communities.
The Girvan-Newman algorithm [26] iteratively removes the edges from the original network. At each step, the removed edge is the one which best fits a heuristic called *betweenness centrality*. This heuristic attempts to measure the edge being unlikely to form part of a community. For any given edge, *betweenness centrality* is estimated by the *number of shortest paths* between any two nodes in the network that pass through said edge. The algorithm produces a tree-like hierarchy of community structure. Girvan-Newman is most optimal in the case that the network is sparse, in which case it may have a time complexity in $O(n^3)$ [26], but generally, it has a time complexity in $O(n \cdot m^2)$ [27], where $n = |V|\,,\ m = |E|$.
The Louvain method [6] iteratively constructs communities. The method starts with communities of single nodes, one for each node in the original network. At each iteration, for each node, a heuristic is calculated to determine whether the node should be grouped with one of its neighbors in the original network. When no improvement according to the heuristic can be made, the algorithm constructs a network in which the discovered communities form nodes, and weighted edges represent the number of connections between the communities. These two steps repeat until no improvement is made.
The Leiden method [28] is an improved version of the Louvain method, with improved accuracy in the selection of community membership, higher level of refinement in each algorithm step, and faster run time. "The Leiden algorithm is considerably more complex than the Louvain algorithm" [28]. Hence, we will not describe this method in detail.
Our literature review did not uncover any conclusive time complexities for either the Louvain method or the Leiden method. According to Xu, Ren, and Sun, the Louvain method has a time complexity in $O(n \log n)$ [27]; however according to Zhang, Fei, Song, *et al.* the time complexity is approximately "$o(m + n)$ [sic]" [29].

2.5 Network motifs

Milo, Shen-Orr, Itzkovitz, *et al.* [7] introduce a *network motif* as a subgraph of a particular directed graph, the shape of which occurs more frequently than statistically expected [7]. More precisely, a size n network motif is a graph of n nodes that is isomorphic (see Section 2.3) to one or more subgraphs of the input graph, for which a *null hypothesis* does not hold. The aim is to show that these structures do not emerge directly from properties such as the degrees of the nodes in the graph. Hence Milo, Shen-Orr, Itzkovitz, *et al.* [7] select a null hypothesis that the structures occur with equal frequency as the expected frequency in a *null model*, chosen to be a *randomized graph* with similar properties. The significance of a motif is demonstrated by showing that these isomorphic subgraphs occur with significantly higher frequency in the input graph than in the null model.

The randomized graph that makes up the null model must have the following three properties:

1. It has the same number of nodes and edges as the input graph.
2. Each node in the randomized graph has the same degree as the corresponding node in the input graph.
3. Each edge in the randomized graph is connected between two randomly selected nodes.

Especially for small input graphs, it may be that the only possible randomized graph is the original graph, and therefore network motifs can by definition not exist for this input.

To produce the null model graph for size $n = 3$ network motifs, the authors apply an algorithm based on iteratively selecting random pairs of edges from the graph, and swapping the destination nodes of the edges. To maintain the first two properties, this only occurs under certain conditions. Bi-directional and uni-directional edges are treated differently.

The null model graph for size $n > 3$ network motifs is more involved. The authors attempt to avoid assigning significance to a size n structure solely because it includes a significant size $n - 1$ sub-structure. For details, see Supplementary Web material of Milo, Shen-Orr, Itzkovitz, *et al.* [7].

2.5.1 Milo's algorithm

The original algorithm to find network motifs involves an isomorphic subgraph counting algorithm, performed once on the original input network. Then, the counting algorithm is applied to 1000 randomized input networks. Statistics such as mean frequency and variance are collected from the randomized networks. The frequencies from the original input network are then compared to the mean from the randomized networks plus two standard deviations. When the frequency exceeds this, the counted subgraph is reported as a network motif.

The isomorphic subgraph counting algorithm essentially loops over each edge of the graph, and from this edge builds subgraphs that the edge is part of, by following the edges in a connectivity matrix. Precise details on how isomorphic subgraphs are detected are not mentioned. However, one count per set of isomorphic subgraphs appears to be maintained.

2.5.2 Parallel and distributed network motif discovery

Our literature review uncovered multiple parallel and distributed approaches to computing network motifs. To illustrate the variety of these approaches, we will select two highly dissimilar approaches as examples. One example is the GPU-parallelized exact solution algorithm "NemoGPU" [30], which has a clear foundation in the original algorithm [7] with subsequent improvements [31]–[38]. In contrast, our other example, the estimation algorithm of Wang, Lui, Ribeiro, *et al.* [39] completely rephrases the problem, to arrive at an algorithm that may allow for large-scale distribution of computation.

2.5.3 Estimation algorithms

One method for sampling statistics about a graph is called *Random Walk* (RW). RW algorithms have high potential for distributed computation of estimates, as they generally don't require much synchronization.
According to Ribeiro and Towsley [40], "A RW samples a graph by moving a particle (walker) from a vertex to a neighboring vertex (over an edge). By this process, edges and vertices are sampled. The probability by which the random walker selects the next neighboring vertex determines the probability by which vertices and edges are sampled" [40].
This means that if the RW encounters nodes or edges with a non-uniform probability, this introduces a statistical bias into the sampling. Applying a RW to sampling motif statistics is possible, but a trivial implementation may introduce very large biases. Hence, such an algorithm may produce inaccurate results. For motif sampling, visiting edges with uniform probability would be ideal.
Ribeiro and Towsley [40] propose an algorithm "Frontier sampling," which attempts to produce more accurate results by visiting each node with a probability proportional to its degree. The authors apply Horvitz-Thompson estimators [41] to construct estimators for graph characteristics. These estimators converge toward the unbiased mean of these graph characteristics as the number of samples approaches infinity. For large graphs, the method tends to approximate visiting edges with uniform probability.
As for asymptotically unbiased estimators, Wang, Lui, Ribeiro, *et al.* [39] claim, "Biases introduced through sampling a bipartite graph using a lazy RW are easily removed. Biases introduced through sampling via a classical RW can only be removed if the graph is non-bipartite" [39].
Wang, Lui, Ribeiro, *et al.* [39] propose an algorithm to efficiently sample a large un-directed, connected, and non-bipartite graph for network motifs of size $k-1$ and size k simultaneously, "Pairwise Subgraph Random Walk (PSRW)." From the original graph, a CIS relation graph of size $k-1$ is constructed. This involves finding all CIS (see Section 2.2) of size $k-1$ in the original graph. These make up the nodes of the relationship graph. Edges in the relationship graph represent the existence of an edge between the subgraphs the nodes represent in the original graph.
The PSRW algorithm starts in a random node of the relationship graph. From there, it performs a RW in the relationship graph, iteratively selecting one edge at a time. The nodes of this edge each represent a size $k-1$ subgraph. Due to the properties of being linked in the relationship graph, these two subgraphs of size $k-1$ can be combined into a subgraph of size k. Further, the resulting size k subgraph is also a subgraph of the original graph.
Presumably, adding a subgraph to the counts of induced subgraphs involves computing a canonical label for each subgraph, as a key into an associative array, and incrementing the value at that position in the array by one. The authors do not explicitly mention the details of performing the count, neither at what stage any canonical labeling is performed, nor the data structures used.
To count subgraphs with PSRW, at each iterative step, i.e., for each relationship graph edge, both the newly selected size $k-1$ induced subgraph and the size k induced subgraph, resulting from combining with the previously visited size $k-1$

subgraph, are added to the counts towards the total number of such induced subgraphs. Presumably, in the base case, the size $k-1$ induced subgraph representing the starting node is added to the counts, but whether this instead occurs at each iteration step and is then accounted for in the final, statistical, step is not explicit. Finally, PSRW applies unbiased estimators to the counts to estimate the true frequency of the induced subgraphs, or potential motifs.

Wang, Lui, Ribeiro, *et al.* [39] further proposes the algorithm "Mix Subgraph Sampling (MSS)." MSS estimates motifs of the sizes $k-1$, k, and $k+1$ simultaneously. MSS resembles the PSRW algorithm, but performs the pairwise random walk over a CIS relationship graph of size k. Like PSRW, MSS samples the relationship graph's edges, combining pairs of size k subgraphs to count subgraphs of size $k+1$. In contrast to PSRW, MSS further decomposes each encountered size k subgraph. For each edge in the size k subgraph, if the edge were to be removed from the subgraph, it results in a size $k-1$ subgraph. This size $k-1$ subgraph is also a subgraph of the original graph. Hence, in addition to the counts performed by PSRW in the iterative step, MSS further adds all such decomposed subgraphs of size $k-1$ to the counts. Like PSRW, MSS applies asymptotically unbiased estimators to the final counts to estimate the true frequency of these different sizes of subgraphs in the original graph.

2.5.4 GPU acceleration

Lin, Xiao, Xie, *et al.* [30] compare the major steps of typical network motif discovery algorithms and propose the algorithm "NemoGPU" for motif discovery in a directed graph. In NemoGPU, the most computation-intensive steps of said algorithms have been replaced with less efficient, but highly parallelizable algorithms. In particular, work has been done to parallelize frequency estimation of motifs and subgraph enumeration. The temporary random graphs, to which subgraph frequency is compared, are generated in parallel on the GPU, allowing for a significant performance increase in the frequency estimation step. Further, the most computation-intensive steps of subgraph enumeration, enumerating the valid combinations of vertices, and computing the unique adjacency matrices have also been parallelized. NemoGPU attempts to avoid branching when possible, and carefully manages memory in such a way as to facilitate the GPU memory caching and avoid parallel threads of execution writing to the same data ranges, which could cause locking. Further, it attempts to divide work into similarly sized chunks to achieve load balancing across the GPU cores. The authors also test the applicability of certain optimizations from earlier studies, such as exploiting symmetry constraints on automorphism groups [42]. They further discuss an approach to computation on the GPU when memory for the problem set exceeds the total RAM. In summary, the authors achieve a performance increase, compared to similar algorithms using CPU computation, of one order of magnitude using a consumer-grade GPU, and two orders of magnitude using an NVidia Tesla GPU.

2.6 Temporal motifs

A *temporal network* extends the concept of a directed graph. An *event* corresponds to a directed edge but represents the notion that something occurred at a certain time and for a certain duration of time.

Definition 2.6.1 (Event)**.** $e = (t, d, v, v')$ is an *event* if (v, v') is a *directed edge* between two nodes, t is a starting point in time, and d is a duration in time.

Definition 2.6.2 (Temporal network)**.** $G = (V, E)$ is a *temporal network* if V is a set of nodes, and E is a set of events between the nodes of V.

Extending the concept of a network motif to temporal networks is not trivial. For a definition of a temporal motif that can convey the semantics of the temporal network, properties such as event order, inter-event time intervals, and duration need to be considered. To the best of our knowledge, no published definition considers all these aspects and how they inter-relate. In particular, no definition of a temporal motif considers the duration of the events.

Given the lack of a full definition of a temporal motif, it is not clear whether detecting such motifs in a temporal network of any meaningful size might be computationally feasible. Extending the network motif definition of Milo, Shen-Orr, Itzkovitz, *et al.* [7] with a further two degrees of freedom might imply that a naive algorithm would have a complexity with at least two more exponential factors than that of network motif discovery, and that these factors might be large.

Literature on temporal motifs hence focuses on finding limited definitions that facilitate computation while maintaining utility for analysis of the network. Liu, Guarrasi, and Sariyuce [8] performed a comparative study on four main definitions [9]–[12] of generally applicable temporal motifs, independent of topology while considering temporal adjacency.

Kovanen, Karsai, Kaski, *et al.* [9] introduced a definition of motif that reduces the required number of subgraph matching problems to linear in the size of the network. Song, Ge, Chen, *et al.* [10] introduced a definition allowing for streaming detection, where events can be partially ordered. Hulovatyy, Chen, and Milenković [11] modified constraints on the definition of Kovanen, Karsai, Kaski, *et al.*, enabling significantly lower complexity of motif detection, while maintaining equal or better utility for the analysis of certain types of network structures. Paranjape, Benson, and Leskovec [12] likewise modified constraints on the definition of Kovanen, Karsai, Kaski, *et al.*, for the purpose of capturing motifs occurring in a short burst.

To the best of our knowledge, the definition of Kovanen, Karsai, Kaski, *et al.* appears by far the most widely used general-purpose definition in the literature, and hence we will focus on it. The definition and the algorithm for motif discovery will be discussed in Chapter 3.

2.6.1 Optimization of Kovanen's algorithm: TM-Miner

Sun, Tan, Wu, *et al.* propose TM-Miner [43], [44], which is based on Kovanen's algorithm [9], and attempts to improve its performance through multiple strategies. Fundamentally, a particular evaluation order of the temporal network is applied, called time-first search. The properties of this evaluation order are then applied

to efficiently implement a *support threshold*, a heuristic measure for how frequent a motif must be for the algorithm to perform graph isomorphism computation and collect statistics regarding it. Further, optimizations that appear to depend on incremental updates of the global state are applied.
Sun, Tan, Wu, *et al.* report improvements in runtime compared to Kovanen's reference implementation by two to three orders of magnitude. However, they appear to contribute yet another measure of motif significance, which appears to override those of Kovanen.

2.6.2 Related thesis work at Chalmers University of Technology

This study is related in topic to another thesis in temporal motif discovery conducted at the Department of Computer Science and Engineering, Chalmers University of Technology. Bjurenlind and Hadi [45] studied modeling the movement of users within edge clouds using temporal networks and analyzing the results using temporal motif discovery. Further, the authors presented an application-specific algorithm for motif discovery in such networks and analyzed its performance. The authors' main research question was "how well temporal motifs can characterize the dynamics and functioning of [edge cloud systems]" [45].
In contrast, the focus of this study is on investigating the potential performance benefits of parallelization of published algorithms, not on implementing application-specific algorithms. The secondary focus is on investigating the applicability of application-specific optimization, again to published algorithms, in contrast to new algorithm development.
The reference implementation used by Bjurenlind and Hadi [45] was based on Kovanen, Karsai, Kaski, *et al.* [9], whereas the application-specific algorithm presented was primarily based on that of Sun, Tan, Wu, *et al.* [43], see Subsection 2.6.1. However, the algorithm design makes multiple assumptions on uncertain details of said algorithm to make the implementation of the algorithm implementable and reproducible. Bjurenlind and Hadi [45] fail to reproduce the results of the reference algorithm using said algorithm.

2.7 Open access temporal network data sets

Our literature review did not uncover a large number of temporal network data sets representing interactions that may exhibit community structure, openly available and appropriately curated for use with TMFinder or similar algorithms. We therefore also investigated opportunities to transform openly available data sets to the appropriate type of temporal network.

2.7.1 The email-Eu-core temporal network

Paranjape, Benson, and Leskovec [12] present a curated data set named *email-Eu-core-temporal* [46], using anonymized email data from a large European research

institution. The data set consists of 986 nodes, representing individuals, and 332334 events of the form (u, v, t), representing that u sent an e-mail to v at time t. These events correspond to 24929 edges in a static graph. A separate event is created for each recipient of the e-mail. The data set is ordered by time t and does not contains self-loops, that is, events of the form (u, u, t), indicating that one of the recipients of the e-mail was the one sending it.

2.7.2 Telecommunications in Milan

Barlacchi, Larcher, Casella, *et al.* [47] presented an anonymized data set [48] representing cellular network communication in the Milano metropolitan area in Italy. The data summarize interactions between, on one side, zones in this area, and, on the other side, country codes. Another data set [49] summarizes interactions between pairs of zones of this area. Further, the same types of data were also presented for the city of Trento. The data sets were collected by summarizing Call Data Records collected for the purpose of billing and network maintenance by Telecom Italia.

2.7.3 Other temporal network data sets

Rossi and Ahmed [50] present other potentially applicable data sets, many of which contain temporal aspects as well as social interactions, which might imply that some of them may be community-structured.

2.8 Discussion and implications

In this section, we first discuss some applicable findings from the literature review on parallel and distributed network motif discovery. Next, we discuss the implications of Kovanen's definition on the meaning of a temporal network motif. Following this, we discuss how this has implications on whether TM-Miner is actually an optimized algorithm based on Kovanen's, or a distinct definition of temporal network motif. Finally, we discuss the main implications of the literature review.

2.8.1 Parallelization and distribution of network motif discovery

As mentioned in Subsection 2.5.2, there are multiple approaches to parallelizing and distributing network motif discovery; however, a common theme between these approaches is to apply a different level of abstraction to the work to be performed.

2.8.2 Semantics of temporal network motifs

Unlike the traditional static motif [7], Kovanen's definition of a temporal motif (see Definition 3.1.6) does not carry with it any connotation of the motif being significant [9]. Kovanen argues that there are many statistical measures that can be meaningfully applied to evaluate the significance of a temporal motif. In the static motif, the null model to be disproven, for the motif to be significant, is that the

motif also occurs in a randomized version of the network with the same node ranks. Kovanen argues that the most obvious corresponding null model for the temporal network would be to compare occurrences in a time-shuffled network, but also argues that the statistical properties of the time-shuffled network do not at all map well to any datasets that come from real-world measurements. In particular, time-shuffling would give rise to Poisson distributions that do not occur in real datasets.

Kovanen proposes multiple statistical measurements that can give meaningful information about a temporal network and suggests that the best way to construct a null model for a temporal network is to compare it against itself. I.e., if the dataset contains information about more than one, statistically uncorrelated, type of events between the same nodes, then there are measures that can identify which motifs are particular to a chosen type of event.

In particular, Kovanen argues for using the symmetrized Kullback–Leibler divergence to measure the relative entropy between the distribution of motifs in the network under study and that of the reference, which facilitates finding commonly occurring motifs occurring more frequently, rather than emphasizing rare but relatively more frequent motifs.

Further, Kovanen argues for Kendall's τ, computed on the sets of motifs sorted by frequency, as a measurement to compare two temporal networks for relative similarity of motifs. $\tau = 1$ corresponds to the networks having identical distributions of motif occurrence, whereas $\tau = -1$ corresponds to the opposite situation, i.e. the most frequent motif in the first network is the least frequent motif in the second network, and so on.

Hence, the input to Kovanen's algorithm is a set of nodes and multiple types of events between these nodes. The output is not only a list of identified motifs but a variety of statistical measurements between them, allowing the user to select which motifs they find significant for their particular dataset.

2.8.3 TM-Miner as a distinct temporal motif definition

As stated in Subsection 2.8.2, Kovanen's algorithm allows for user selection of significance criteria. However, the TM-Miner approach of optimization, see Subsection 2.6.1, appears to contribute other significance criteria. It is therefore unclear if this approach is applicable to every situation Kovanen's algorithm is.

In fact, the differences between TM-Miner and Kovanen's algorithm may be large enough to be considered a distinct temporal motif definition, as quite minor modifications to the definition result in entirely different results; see Section 2.6.

That Bjurenlind and Hadi [45], see Subsection 2.6.2, failed to reproduce the results of the reference algorithm appears to support the view that TM-Miner implies a distinct definition of the temporal motif.

Further, the optimization approach of TM-Miner appears to depend strongly on the evaluation order and focus on incremental state updates. Therefore, Kovanen's reference implementation appears more promising than TM-Miner as a foundation for a parallelized or distributed algorithm.

2.8.4 Data sets

As mentioned in Subsection 2.7.3, there are many data sets that may or may not be applicable for temporal network motif discovery using TMFinder. However, the process of ensuring compatibility proved to be time-consuming, with many aspects that need to be considered, as will be discussed here. The literature review considered and rejected multiple network data sets, but only one alternative is presented here as an illustration.

The telecommunications data sets from Milan, see Subsection 2.7.2, initially appear promising, in representation of the interactions of both a local area and its surroundings, as well as presenting a clear timeline. However, the unit of analysis of the first network is ambiguous. More specifically, there exist two different types of entities that may represent nodes, namely *cellular network zone of Milan*, and *telephone network country code*. Hence, the static graph would have nodes of two colors, representing zones and countries respectively. Further, there is no information in this first data set about interactions between the countries, nor between the zones. Hence, the structure of the static network is a bipartite graph, where each edge has a zone on one side and a country code on the other. This implies that the data set is not representative of temporal networks in general. The second network has a clear unit of analysis, the zone, and therefore appears more suitable than the first for the application of temporal motif discovery. However, analyzing the results may require detailed knowledge of the specific areas of Milan, for the particular time studied.

Further, in the telecommunications data sets, there is no direct correspondence to *events* at discrete points in time. There exist discrete intervals of time, each with a particular amount of traffic, of a certain type, between certain nodes. Processing this data set for use with temporal network motif algorithms would require making multiple assumptions, and arbitrary decisions, about how to interpret the data. A straightforward method, to convert the amount of traffic data into events, could be to apply a hysteresis function to the amount, in increasing order of time. I.e. if the data suggest that the amount between nodes (u, v) first exceeds a certain threshold y_h for the time interval $[t, t + \Delta t)$, an event (t, d, u, v) is generated. When said amount for the edge no longer exceeds a second threshold y_l, for the time interval $[t', t' + \Delta t)$, said event's duration is computed as $d = t' - t$. For the time intervals between t and t', no additional events would be generated involving said edge. When $y_h > y_l$, this method reduces the rapid generation of multiple events in the case of the amount oscillating near an arbitrary threshold level.

However, the next problem for using the telecommunications data sets then becomes how to select the threshold levels. For the different zones of Milan, each zone does not have the same population, and therefore the same amount of traffic does not necessarily correspond to the same significance. Further complicating the situation, workplaces, commercial zones, and other sources of additional communication may cause varying levels of what constitutes significant traffic throughout the day. Also, both sides of the edge would need to be accounted for in computing an adjustment factor for the threshold. The complexity in processing this network for use, and proving the method correct, would in itself be worthy of a major section of the thesis, and therefore it was decided to be out of scope.

In contrast, the properties of the data set email-Eu-core-temporal, as presented in Subsection 2.7.1, facilitate conversion to a format for use by Kovanen's algorithm. Conversion mostly involves adding a duration column, which in practice is not used by the algorithm. The duration can hence be arbitrarily selected to a constant value, e.g., '0'. The unit of analysis is clear, namely individuals, and the meaning of the event is clear, namely that an e-mail from one individual is addressed to another individual at a particular time. This facilitates analysis of the resulting motifs. The properties of the network have also been previously studied in literature on temporal network motif discovery. The data set is also sufficiently large to lead to extensive computation times using TMFinder on a contemporary computer system. Hence, this network was selected as the focal data set for this thesis.

2.8.5 Implications

Our literature review revealed a common theme in many of the more advanced approaches to network motif discovery, in that a different level of abstraction of the division of work was applied. To the best of our knowledge, there are no studies that directly apply an approach based on community structure properties of graphs to network motif discovery.
Further, the literature lacks studies on parallelized and distributed approaches to the exact solution approach of temporal motif discovery.

3
Kovanen's algorithm

In this chapter, we first attempt to describe the definition of temporal network motifs as used in Kovanen's algorithm for temporal network motif discovery [9], as mentioned in Section 2.6. For concepts used, see Section 2.2 and Section 2.3. We then proceed to give a brief overview of said algorithm, with its major components, and how each part relates to said definition. For discussion on what makes a motif significant, see Subsection 2.8.2, and for a more in-depth view of the algorithm in the context of the research questions, see Chapter 4.

3.1 Kovanen's definition of temporal motifs

Consider a pair of events $e_1 = (t_1, d_1, v_1, v_1')$, $e_2 = (t_2, d_2, v_2, v_2')$, where t is the starting time of the event, d is the duration of the event, and (v, v') is the directed edge the event corresponds to. Let the events $\{e_1, e_2\}$ be sorted in ascending order on (t, d).

Definition (δt)**.** Let $\delta t = t_2 - (t_1 + d_1)$.

Definition (Common nodes)**.** The pair of events has *common nodes* if:

$$\{v_1, v_1'\} \cap \{v_2, v_2'\} \neq \emptyset$$

Definition 3.1.1 (Δt*-adjacent*)**.** The pair of events is Δt*-adjacent* if it has *common nodes* and $|\delta t| < \Delta t$.

In other words, the events of a Δt*-adjacent* pair of events have a common node, and further, the difference in time between the end of the first event, in order of starting time, and the start of the second is less than Δt.

Consider the directed graph G composed of all edges (v, v') of all the events. Consider a sequence p of events, in which the edges correspond to a valid path through G.

Definition 3.1.2 (Δt*-connected*)**.** The pair e_1, e_2 is Δt*-connected* if there exists a path p, in ascending order of t, d, where $p = [e_1, \ldots, e_2]$, and for each pair of consecutive events in $p = [\ldots, e_x, e_y, \ldots]$, the pair e_x, e_y is Δt*-adjacent*.

In other words, the pair of events is Δt*-connected* if there exists a sequence of events, in order of starting time, which starts in e_1 and ends in e_2, such that all pairs of consecutive events are Δt*-adjacent* [9].

Definition 3.1.3 (Connected temporal subgraph)**.** A subgraph of the temporal graph of events is a *connected temporal subgraph* if for any given pair of events in the subgraph, the pair is Δt*-connected*.

In other words, a connected temporal subgraph consists only of pairwise Δt-*connected* events. Such a subgraph is connected both topologically and temporally and could therefore be used as the basis of a temporal motif definition.
However, a star-shaped network of size n, where all events happen within Δt, contains $\binom{n}{k}$ different *connected temporal subgraphs* of k events [9]. This would cause problems both in terms of computational efficiency and the interpretation of motif statistics. Hence, Kovanen introduces further constraints on the type of subgraphs to consider.
Definition 3.1.4 (Valid temporal subgraph)**.** A connected temporal subgraph is a *valid temporal subgraph* if all Δt-*connected* events of each node are consecutive in starting time [9].
Remark. The unit set of a single event is chosen to be a valid subgraph. Hence, there always exists a valid subgraph that contains any chosen event.
The number of valid subgraphs of size k to consider, for a star-shaped network of size n, is thereby reduced to $n - k + 1$. However, this definition also leads to the restriction that each node can at most be involved in one event at a particular time. This means that there exist some patterns which may be interesting that the model cannot represent.
Let G_1, G_2 be temporal graphs, and G_1', G_2' the corresponding directed graphs of their edges.
Definition 3.1.5 (Isomorphic temporal graph)**.** $G_1 \simeq G_2$ if $G_1' \simeq G_2'$ and the temporal order of the corresponding events is identical.
Remark. For $G_1' \simeq G_2'$, see Section 2.3.
Definition 3.1.6 (Kovanen's temporal motif)**.** A *temporal motif* is a set of *valid temporal subgraphs* that are mutually *isomorphic*.

3.2 Kovanen's algorithm

Kovanen's approach to finding all temporal motifs in a temporal network is composed of the following major steps:
1. Find all *maximal connected subgraphs* (Definition 3.2.1).
2. Find all *valid temporal subgraphs* (Definition 3.1.4).
3. For each valid subgraph, identify the corresponding *temporal motif* (Definition 3.1.6).

3.2.1 Finding maximal subgraphs

Definition 3.2.1 (Maximal connected subgraph)**.** The *maximal connected subgraph* of an event is the largest *valid temporal subgraph* that contains the event.
Remark. For each event, there exists a unique maximal subgraph.
Because all valid subgraphs have a particular order, every possible valid subgraph is fully contained within at least one maximal subgraph.
Each maximal subgraph is formed by gradually extending each possible minimal valid subgraph as far as the properties of validity will hold; see Definition 3.1.4. For any event $e = (t, d, v, v')$ in the temporal graph, the subgraph $G' = (V, E) = (\{v, v'\}, \{e\})$ is by definition also a valid subgraph.
To find a maximal subgraph of event e:

1. Order the events by starting time.
2. Start with the subgraph $(\{v, v'\}, \{e\})$.
3. Scan the events in positive temporal order, attempting to extend the subgraph while validity conditions hold.
4. Do the same in the negative temporal order.
All events found in the process share the same maximal subgraph. Hence, all maximal subgraphs can be found in $O(|E|)$ time, where E is the set of events.

3.2.2 Finding valid subgraphs

To find all valid subgraphs, Kovanen considers an undirected graph in which the vertices correspond to events in the maximal subgraph. There are edges between the vertices if the events are Δt-*adjacent* and consecutive for either node. Kovanen iterates over all induced subgraphs of this graph; see Definition 2.2.2. A subgraph is identified as a valid subgraph if the events of each node are consecutive.

3.2.3 Identifying temporal motifs

In order to uniquely identify the temporal motif corresponding to a valid subgraph, Kovanen computes its *canonical labeling*; see Subsection 2.3.3. However, it is not enough to compute the canonical labeling of the topological structure of the temporal graph. Information about the order of events must also be included in the structure of the graph for which canonical labelings are computed.
Kovanen chooses to represent the temporal ordering in graph form by combining the topological structure of the temporal graph with the topological structure of the aforementioned graph used to identify valid subgraphs; see Subsection 3.2.2. The algorithm represents the temporal subgraph as a vertex-colored directed graph, in which the nodes of the temporal network are represented as vertices of one color, and the events as vertices of another color. Instead of the event being represented as a directed edge between two nodes, a directed edge is inserted from the origin node to the event, and another from the event to the destination node. Further, from each event, a directed edge is inserted from the event to its consecutive event, if one exists.
For each resulting graph, the canonical labeling is computed. Those with an identical canonical labeling are considered the same distinct temporal motif. Kovanen uses the tool Bliss to compute canonical labeling; see Subsection 2.3.3.

4

Parallelization

In this chapter, we mainly address RQ 3. To do so, we first introduce Amdahl's law and how it relates to the execution time of a partially parallel computer program. We proceed to show the method through which we parallelize parts of the execution of Kovanen's algorithm. We then show our results in the form of execution times on a particular dataset using different numbers of threads running in parallel. We relate this to Amdahl's law in order to show the inherent parallelism of the analyzed sections of the program.

4.1 Theory

Algorithms may be inherently sequential. In such algorithms, for each step of the computation, the result depends on the results from a previous step of computation. Other algorithms are inherently parallel, meaning that the result of each step of the computation is obviously and entirely independent from the result of every other step. Real-world algorithms and hence computer programs tend to consist of parts belonging to each of these categories.

Consider a fixed computation problem, computed sequentially on a single computer processor, where s and p are the proportions of execution time spent in inherently sequential and parallel parts, respectively. $s + p = 1$. When this problem is instead computed in parallel over N identical processors:

Definition 4.1.1 (Amdahl's law)**.** Amdahl's law states that the theoretical maximum speedup that could be achieved is: $\frac{1}{s+\frac{p}{N}}$

However, for many types of problems, the theoretical p grows relative to s as the size of the problem grows.

Applying Amdahl's law, we expect that, for a program with parallel proportion p, if the total sequential execution time, i.e. for $N = 1$, is T, then if the program is executing in parallel on N processors, $N > 1$, and the execution time is t, then the parallel execution time proportional to the sequential execution time, $\frac{t}{T}$, is:

Definition 4.1.2 (Proportional execution time)**.** $\frac{t}{T} = (1 - p) + \frac{p}{N}$

See Figure 4.1 for a visualization of how p affects the execution time.

Assuming our problem follows Amdahl's law, and given $T > t > 0, N > 1$, we seek to derive p, as a measure of the level of parallelism. Rearranging Definition 4.1.2,

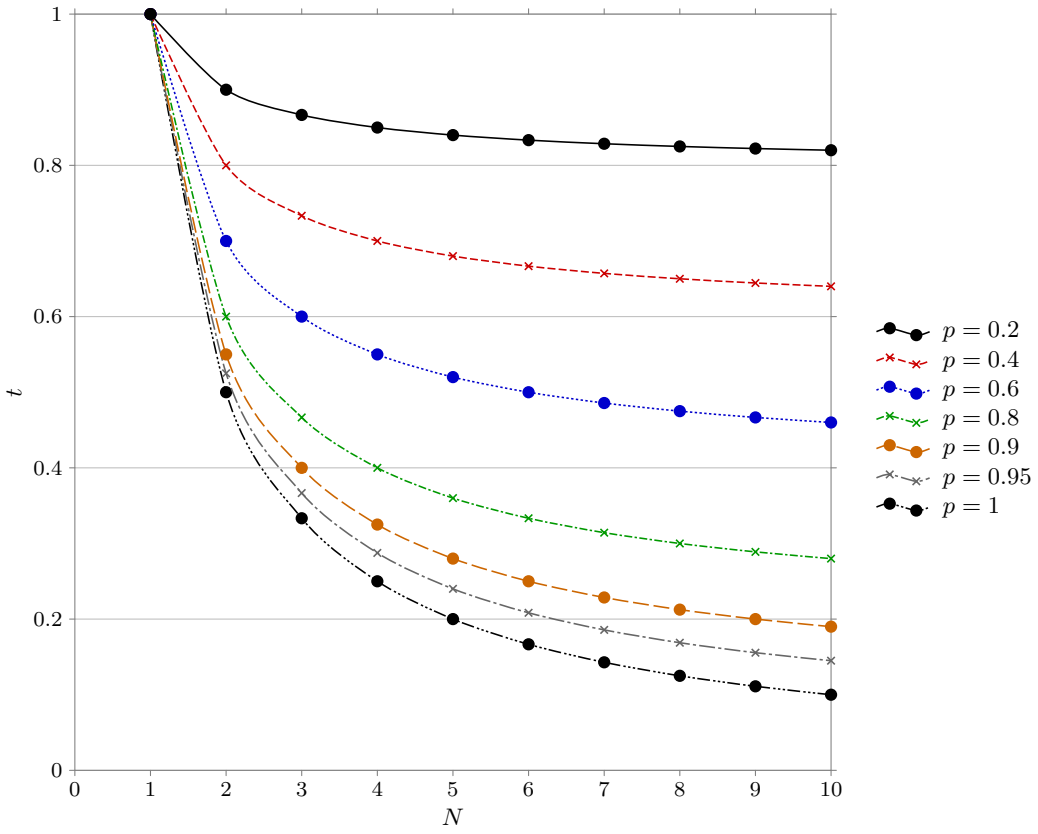


Figure 4.1: Ideal execution time of problems obeying Amdahl's law, with varying parallel proportion p, when executed in parallel over N identical processors

we get:

$$\begin{aligned}(1-p)+\frac{p}{N} &= \frac{t}{T}\\ p\left(\frac{1}{N}-1\right) &= \frac{t}{T}-1\\ p &= \frac{t-T}{T}/\frac{1-N}{N}\\ &= \frac{t-T}{T}\frac{N}{1-N}\\ &= -\frac{T-t}{T}\left(-\frac{N}{N-1}\right)\\ &= \frac{N\,(T-t)}{(N-1)\,T}\end{aligned}$$

Definition 4.1.3 (Approximation of p)**.** $p = \frac{N(T-t)}{(N-1)T}$

4.2 Method

We applied limited parallelization to Kovanen's algorithm by adjusting the reference implementation, TMFinder. For a high-level description of the algorithm from a mathematical point of view, see Section 3.1 and Section 3.2. We identified the main time-consuming section of the program and modified it to execute in parallel in different threads, using shared memory.

4.3 Implementation

Kovanen's reference implementation, TMFinder, consists of highly detailed and complicated C++ code, and referring to real program code here would obscure our discussion. From an abstract, high-level perspective, TMFinder has the structure given in Algorithm 4.1.

Algorithm 4.1: TMFinder, Kovanen's reference implementation (simplified)

```
1  input: int K, nodes N, events E, parameters P
2  output: map of motif × (int, motif_statistics)
3  begin
4    lookup ← compute_tables(K, N, E, P)
5    frequencies ← compute_frequencies(lookup)
6    statistics ← compute_statistics(lookup, frequencies)
7    return statistics
8  end
```

The input to TMFinder consists of the motif size K, the temporal graph $G = (N, E)$, and various parameters. The parameter *time window* refjers to the maximum time between two events for them to be considered Δt*-connected*, see Definition 3.1.2. The time window is required, as it makes up part of the definition of a motif, see Section 3.1.
The program begins by computing various data structures, here called 'lookup', for the use of the main computation functions. These are essentially constant after computation.

Among the most significant data structures computed at the first step are:

- All maximal subgraphs (see Definition 3.2.1)
- For each event, the count of all valid subgraphs starting with that event (see Subsection 3.2.2)

The 'statistics' in the final step of Algorithm 4.1 are discussed in Subsection 2.8.2. In short, Kovanen argues for entirely context-dependent definition of what is a significant motif. Depending on execution parameters to TMFinder, various different statistics are computed for each found motif, to allow the user to compute the significance of each motif.

We are primarily concerned with the computation of the 'frequencies' of the motifs, in the second step of Algorithm 4.1. The final step follows a highly similar structure but has more complications. We will therefore focus on 'compute_frequencies'; see Algorithm 4.2.

Algorithm 4.2: compute_frequencies (simplified)

```
1  input: tables lookup
2  output: map of motif × int
3  begin
4    frequencies ← ∅
5    for each node ∈ all_nodes(lookup)
6    begin
7      for each path ∈ all_paths_from(lookup, node)
8      begin
9        motif ← compute_motif(lookup, path)
10       if frequencies contains motif
11       then
12         frequencies[motif] ← frequencies[motif] + 1
13       else
14         frequencies[motif] ← 1
15       end if
16     end for
17   end for
18   return frequencies
19 end
```

In Algorithm 4.2, the two outer loops together iterate over all Δt-*connected* paths of length $\leq K$ in the temporal graph, by first iterating through each node that could start a path. More precisely, these loops are actually iterating over each *valid temporal subgraph*, see Definition 3.1.4 and Subsection 3.2.2, which is important to the algorithm but not for our discussion here.

For each 'path', 'compute_motif' performs the canonical labeling of its corresponding temporal subgraph. How this is done will not be important for our discussion here, but it follows the process described in Subsection 3.2.3. In short, another graph is computed to represent the temporal subgraph. The representative graph is a non-temporal, vertex-colored directed graph. The Bliss library is then used to perform canonical labeling on said representative graph. The computed motif, or more precisely the hash function value which Bliss assigns it, is then used as the key into an associative array, or 'map', in which the motif frequencies are stored. The corresponding map entry is incremented.

4.3.1 Parallelism in 'compute_frequencies'

For the purpose of parallel computation in shared memory, we have hereby reached the purpose of delving into this detail here. Theoretically, the loops up to computing 'motif' could be executed in parallel, with arbitrary selection of granularity of work, down to the individual 'path', as they do not write to shared data structures. However, the final step of incrementing the motif frequency in the associative array structure on lines $10 - 15$ will correspond to a *synchronization point*. No such operations can run in parallel, or it would risk creating a wrong result.

Canonical labeling of a graph is a very expensive operation with regard to execution time. Comparatively, a very small proportion of time is spent incrementing a value in an associative array. Adding synchronization to the incrementation operations on the associative array should not be an expensive operation. We will test this assumption. Our analysis of 'compute_frequencies' shows that we should expect $p \gg s$ for this part of the program.

4.3.2 Division of work for parallelism

In practice, the TMFinder code is not easily adaptable to arbitrary selection of work granularity due to its structure. Further, choosing a granularity level of the 'path' would lead to increased memory use, and be overly short for practical use. For the sake of our implementation we will choose to look at 'compute_frequencies' as in Algorithm 4.3. Here, 'compute_frequencies_from' corresponds to Algorithm 4.2 lines $7 - 16$.

Algorithm 4.3: compute_frequencies (chosen abstraction level)

```
1   input: tables lookup
2   output: map of motif × int
3   begin
4     frequencies ← ∅
5     for each node ∈ all_nodes(lookup)
6     begin
7       compute_frequencies_from(frequencies, lookup, node)
8     end for
9     return frequencies
10  end
```

To divide work between threads, we attempt a straightforward approach of parameterizing 'compute_frequencies' over ranges of the node indices, see Algorithm 4.4.

As a first step, we divide the work by dividing ranges of nodes to handle over the number of parallel threads. I.e., for n nodes and N threads, the first thread computes for node indices $\left[0, \left\lfloor \frac{n+\frac{N}{2}}{N} \right\rfloor - 1\right]$. This does not distribute the work equally over the threads, but for a large uniform network, the errors should tend to be small.

Algorithm 4.4: compute_frequencies (parallel version, N threads)

```
1  input: tables lookup
2  output: map of motif × int
3  begin
4    n ← |all_nodes(lookup)|
5    frequencies ← ∅
6    for each thread ∈ 0...(N − 1) in parallel
7    begin
8        l ← ⌊thread · (n + N/2)/N⌋
9        r ← ⌊(thread + 1) · (n + N/2)/N⌋ − 1
10       for each node ∈ all_nodes(lookup)[l ... r]
11       begin
12         compute_frequencies_from(frequencies, lookup, node)
13       end for
14   end for
15   return frequencies
16 end
```

4.3.3 Synchronization

Since the threads use a shared map data structure for the motif frequencies, there is a risk of data loss if two threads attempt to update the structure simultaneously, as in Algorithm 4.2 lines $10 - 15$.
To avoid this situation, we chose to implement synchronization using an *advisory mutual exclusion lock*, colloquially *mutex*, around all accesses to the map, see Algorithm 4.5. This approach forces all accesses to the frequencies map to be executed sequentially.

Algorithm 4.5: inc_motif

```
1  input: tables lookup, motif motif, map of motif × int frequency
2  output: void
3  begin
4    lock(frequencies_mutex(lookup))
5    if frequencies contains motif
6    then
7      frequencies[motif] ← frequencies[motif] + 1
8    else
9      frequencies[motif] ← 1
10   end if
11   unlock(frequencies_mutex(lookup))
12 end
```

We tested the performance impact of this synchronization by comparing running times of benchmarks with synchronization and with no synchronization, being allowed to produce incorrect motif statistics, on multiple data sets of limited size. The difference in execution time was within the margin of error, with only one motif count found to differ by 1 in the results of the non-synchronized benchmark. This was according to expectation, as the time for a lookup in a hash map is very short compared to the time for canonical labeling of a graph.

4.3.4 Implementation of 'compute_statistics'

'compute_statistics' is used for various purposes, selectively computing statistics towards the various significance criteria mentioned in Subsection 2.8.2, and is in many ways more complex than 'compute_frequencies'. However, the program follows

a very similar structure, with an outer loop essentially iterating over nodes, and inner loop with time-consuming operations leading up to reading and updating the shared data structure for a comparatively short time. Hence, we chose to implement the parallelization and synchronization as for 'compute_frequencies' detailed above, i.e., with exactly the same division of work strategy between threads, and by wrapping all accesses to the shared data structure in a mutex. Technically, this is the program we performed tests on in Subsection 4.3.3. For the sake of brevity and avoiding unnecessary complexity, we have chosen to omit the implementation details.

4.4 Results and implications

Benchmarks of the modified implementation of TMFinder were performed over the dataset `email-Eu-core-temporal` with the parameters of *max motif size* $k = 3$, *time window* 1200 (s), and *reference count* 0.
The benchmarks were executed on a computer equipped with 64 GB RAM, allowing for TMFinder's working memory to be contained entirely in RAM for the duration of the benchmark. The processor used was an Intel Core i7-9700 at 3.00 GHz clock frequency, with 8 cores, i.e., processors for the purpose of Amdahl's law.
Benchmarks were executed for $N = 1 \dots 8$ threads in sequence. Execution time was measured both for the entire program execution, as well as for individual sections of the program. Due to time constraints, each benchmark was performed only once.
The execution time of the fully sequential benchmark, i.e. for $N = 1$, was 52902.5 seconds, approximately 14.7 hours. For $N = 8$, it was 11619.2 seconds, approximately 3.2 hours. This implies a "speedup" of approximately 4.5 times.
For the sequential benchmark, the proportion of time spent in each section of the program was 0.54 in 'compute_frequencies' and 0.46 in 'compute_statistics', with less than 10 seconds spent in sections that we did not attempt to parallelize. The latter made up less than 0.0002 of execution time. For the $N = 8$ benchmark, the respective proportions were 0.56, 0.43 and 0.0009 respectively.
Using the execution times for $N = 1$ and $N = 8$, and our equation for p from Definition 4.1.3, we approximate the parallel proportion of execution time to be:

$$\begin{aligned} p &= \frac{N\,(T-t)}{(N-1)\,T} \\ &= \frac{8\,(52902.5-11619.2)}{(8-1)\,52902.5} \\ &\approx 0.8918 \end{aligned}$$

As a real-world computer system is significantly less parallel than ideal, the resulting division of work between the worker threads resulting from our algorithm is far from ideal, and there are no attempts made to reduce synchronization between threads, we can say with certainty that for the *computational problem*, $p > 0.8918$. See Figure 4.2 for a comparison between estimated execution time, based on this value of p, and actual benchmark execution times.
Performing the same kind of analysis on the execution times of the individual parallelized sections, we arrive at $p > 0.8814$ for 'compute_frequencies' and $p > 0.9048$

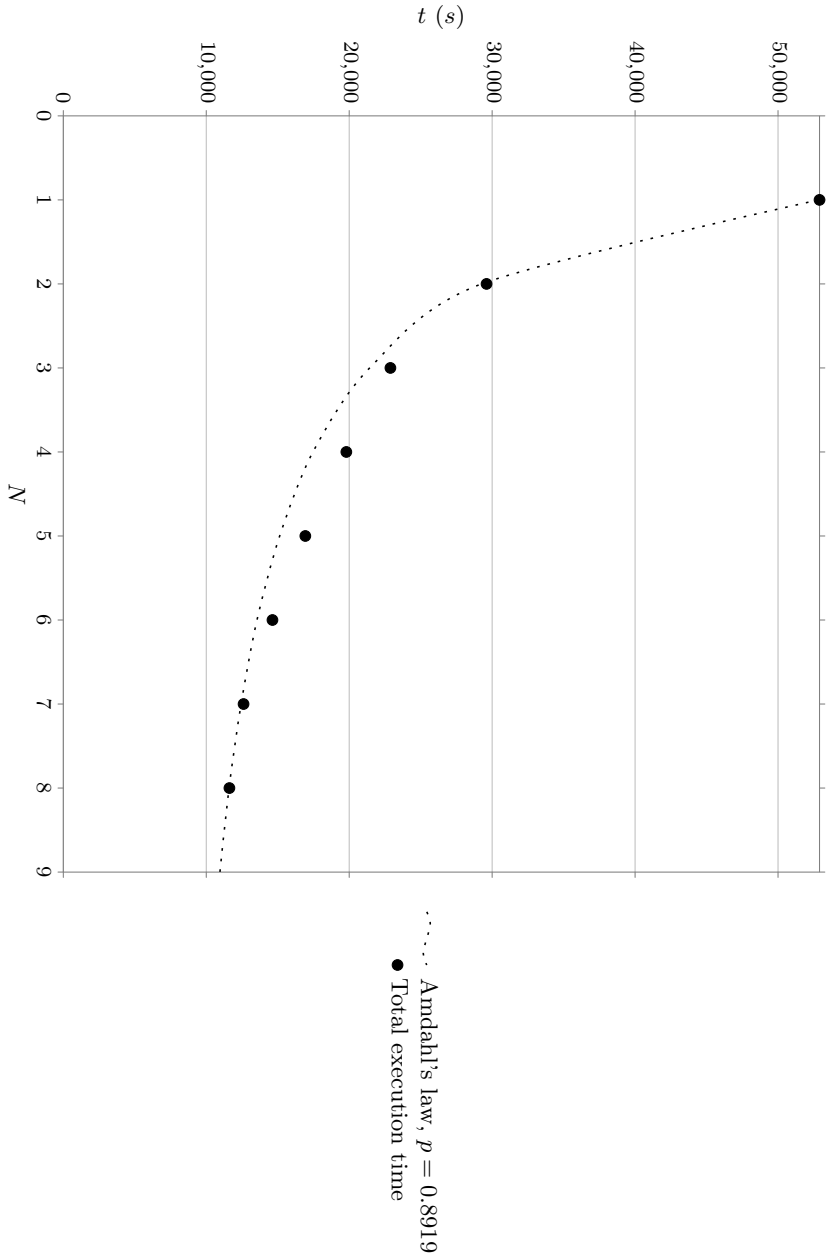


Figure 4.2: Total execution time in seconds of the entire benchmark on the dataset `email-Eu-core-temporal`, when executed in parallel over N processor cores, in comparison to predicted execution time for approximated value of p

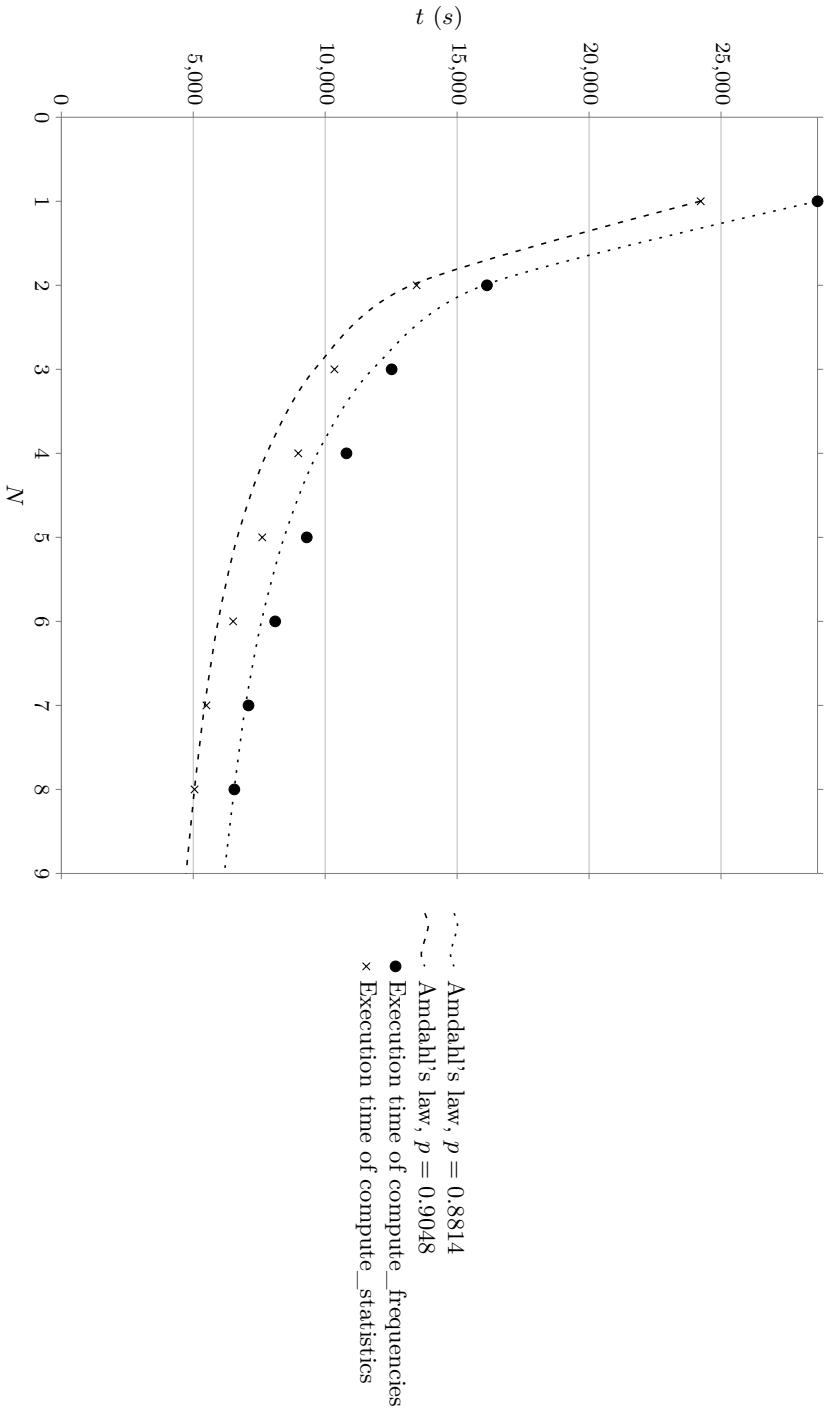


Figure 4.3: Total execution time in seconds, spent in each loop, of the entire benchmark on the dataset `email-Eu-core-temporal`, when executed in parallel over N processor cores, in comparison to predicted execution time for approximated value of p

for 'compute_statistics'. See Figure 4.3 for comparison between estimated execution time, based on these values of p, and actual benchmark execution times.
We can observe a similar trend between these two graphs. The measured execution times for lesser values of N are higher than predicted, based on the estimate of p based on $N = 8$. Conversely, we could argue that for greater values of N, the approximated p appears to converge towards a higher value.
Part of the reason for this is most likely the unequal division of work between the worker threads. As N grows larger, this does not ameliorate the inequality, but it does suggest that the estimated p may converge to a higher value for large values of N. Based on our sample, we are unable to draw further conclusions based on this.
Uncertainties notwithstanding, our results, based on a limited approach to parallelization of a program not designed for parallel computation, suggest that Temporal Motif Discovery as a computational problem has great potential for the application of parallelization techniques.

5
Distribution

5.1 Method

We followed an experimental approach to the development of our algorithm. Based on our understanding gained from literature review, we designed a limited number of small synthetic networks. These networks either clearly demonstrated, or did not demonstrate, the properties of community structure. We made use of these synthetic networks to test approaches to the division of work for a community-structured temporal network.
After the design and implementation of this approach, we then tested the implementation on a real-world community-structured temporal network. We then evaluated and compared the execution time of our approach against the original algorithm on said network through benchmarks. Work which is trivial to perform in parallel was accounted for separately to allow for estimation of the potential best-case parallel execution time.

5.2 Implementation

The main focus of our approach to the distribution of temporal motif discovery involves our approach to the division of work, such that it can be performed independently; see Subsection 2.8.5.

5.2.1 Assumptions

Going into the design of an algorithm to apply community structure to subdivide the work of Kovanen's algorithm, we founded our design on multiple assumptions. Some of the most critical assumptions were:
Assumption 5.1. Community-structured networks can be divided into communities by removing relatively few edges.
Assumption 5.2. The network reachable in $k-1$ steps from such an edge will be of limited size.

5.2.2 Algorithmic approach

Our approach is based on finding *lower* and *upper bounds* to the count of the temporal motifs of the entire network by computing exact motif counts over particular induced temporal subgraphs of said network.

In this section, we will refer to the exact count of a motif i in the full temporal network as c_i, and its lower bound and upper bound as ω_i and o_i respectively:

$$\omega_i \leq c_i \leq o_i$$

This notation is not asymptotic; the above condition is expected to hold for small networks as well.

5.2.2.1 Finding the lower bounds of motif counts

To find the lower bounds, we partition the temporal network into smaller temporal networks, each representing a community, see Section 2.4. These community subgraphs are mutually non-overlapping; any node or event cannot occur in more than one of these temporal networks. Temporal network motifs for each of these community networks are computed independently using Kovanen's algorithm, see Section 3.2. The total motif counts ω_i for these networks are aggregated per motif i.

These counts ω_i represent loose lower bounds of the motif counts for the entire network. $\omega_i \leq c_i$, following from the definition of temporal motif, and the non-overlapping temporal subgraphs being strict subgraphs of the entire network. As the proportion of events left out of the computation approaches zero, ω_i approaches c_i. Hence, the lower bound ω_i may estimate the total motif count c_i of the network when there are relatively few nodes not part of individual communities.

5.2.2.2 Finding the upper bounds of motif counts

The lower-bound counts ω_i, the total motif counts for the individual communities, do not account for motifs that cross the boundary between communities. Hence, to compute an upper bound o_i, we create a second type of work package to compute these boundary-crossing motifs. We compute temporal networks corresponding to these boundary areas. Membership in the boundary areas can be computed per edge across which communities were separated. Each boundary area, for a motif size of k, consists of the nodes of the edge, and all nodes that can be reached in $k-1$ steps. The temporal network for each work package then is the induced subgraph of these nodes in the entire temporal network.

Computing the exact motif count of these individual work packages, and aggregating the totals, we arrive at another count δ'_i, such that $o'_i = \omega_i + \delta'_i$. $c_i \leq o'_i$ and hence o'_i is a loose upper bound of c_i. However, o'_i is needlessly loose; for many types of network structures, it will count motifs in the boundary area multiple times.

To improve our upper-bound estimate, we therefore add a *condition* when computing the upper-bound work package. In Kovanen's algorithm, we only count those motifs that cross the boundary between the communities. More precisely, Kovanen's algorithm iterates over potential paths in the temporal graph. For this work package type, we *require* that a certain edge exists in the path; otherwise, computation of that path is skipped. The required edge corresponds to the edge for which we are creating the work package.

Applying this condition and proceeding with the aforementioned computation, we arrive at a smaller count δ_i, such that $o_i = \omega_i + \delta_i$. $c_i \leq o_i$, and o_i is a loose upper bound of c_i, however less loose than o'_i.

5.2.3 Partitioning into communities

To the best of our knowledge, there are no mature software packages for community detection directly operating on the type of temporal network we have described. The main reason for temporal community detection would be to detect communities that change over time, whereas our purpose is to provide a convenient place to section the temporal network by cutting low numbers of edges. Further, the boundary of the cut remaining static simplifies the later step of recombination.
Therefore, we first adapt the temporal network for analysis with conventional community detection algorithms, see Subsection 2.4.1. This entails creating a weighted, directed graph to represent the temporal network and computing communities of said graph. We can then use the resulting communities to partition the temporal network and create work packages.
Definition 5.2.1 (Flattened projection)**.** Let $G = (V, E)$ be a directed graph, and $G^* = (V^*, E^*)$ a temporal network. If $V = V^*$ and

$$E = \{(v, v') \mid e = (t, d, v, v'),\ e \in E^*\}$$

then we will refer to G as the *flattened projection* of G^*.
Remark. For more detail on *events*, see Section 3.1.
Definition 5.2.2 (Weighted flattened projection)**.** Let $G = (V, E)$ be the flattened projection of $G^* = (V^*, E^*)$. Let $G' = (V', E')$ be a weighted, directed graph. If $V = V'$ and

$$M(v, v') = \{e \mid e \in E^*,\ e = (t, d, v, v')\}$$
$$E' = \{(v, v', c) \mid (v, v') \in E,\ c = |M(v, v')|\}$$

then we will refer to G' as the *weighted flattened projection* of G^*.
First, we compute the weighted flattened projection G' of the temporal network G^*, resulting in a weighted, directed graph. Multiple events in G^* between the same pair of nodes (v, v'), i.e. $\{e \mid e \in E^*,\ e = (t, d, v, v')\}$, are projected into a single weighted, directed edge (v, v', c) in G'. The edge weight c represents the number of events the edge replaces. Said weight represents a stronger coupling between the nodes, which certain community detection algorithms can use to bias their choice of sectioning point between communities.
Second, we apply a community detection algorithm to the graph G', see Subsection 2.4.1 and Subsection 5.2.4. The output of said algorithm identifies clusters of nodes $[C_0, C_1, \cdots, C_n]$, which can be separated from the rest of the graph by removing a small number of, or a low total weight of, edges in the graph G'. Further, it indicates the sets of edges to remove to accomplish this, as a graph $G^C = (V^C, E^C)$ where $V^C = \{i \mid i \in \mathbb{N},\ 0 \leq i \leq n\}$. Since $V' = V^*$, the identified clusters have direct correspondence to the nodes of G^*, and each identified edge corresponds to a set of events in G^*.

Third, we can use each individual cluster C_i to produce a work package. For each cluster C_i, an induced temporal subgraph is produced by including only those events (t, d, v, v') of G^* in which both $v \in C_i$ and $v' \in C_i$. These work packages will be computed with unmodified parameters to Kovanen's algorithm.

5.2.4 Selection of community detection algorithm

A community detection algorithm needs to handle directed and weighted graphs to satisfy the requirements of our algorithm. A further requirement from a practical standpoint was that an implementation is freely available in a form that we could quickly integrate into our software. To the best of our knowledge, we found three algorithms that satisfy our requirements. These are described in Subsection 2.4.1.
Girvan-Newman has known theoretical worst-case bounds in time complexity, at least for certain types of networks. The Louvain method and Leiden method do not. However, in practice, the latter algorithms outperform the former by several orders of magnitude on many datasets.
We performed a minor benchmark to verify this behavior on the Eu-email-core dataset. The runtime of the Leiden method was 0.2 s, whereas the runtime of Girvan-Newman exceeded 20 minutes.
The Leiden method is designed to provide improved correctness of community detection over the Louvain method. Hence, we chose to continue with the Leiden method.

5.3 Results and discussion

Applying the Leiden-based distribution strategy to synthetic community-structured data gave promising results. The synthetic network structures were split fairly cleanly by Leiden into communities, with a relatively small proportion of nodes not being part of individual communities. Running times of the individual TMFinder computations on the work packages representing communities were quite short compared to the running time over the entire dataset. Summation of motif counts across work packages across communities, when adjusted with a factor for the smaller sizes of datasets, also approximated the counts for the entire dataset.
When applied to a real-world community-structured dataset, *email-Eu-core-temporal*, the division into community work packages for lower-bound motif discovery also resulted in significantly shorter cumulative execution times than that of the exact solution on the full dataset. Our approach for lower-bound discovery of motif counts was, assuming ideal parallelism, more than an order of magnitude faster than the exact solution of motif discovery on the same dataset, see Table 5.1.
However, a significantly smaller proportion of nodes formed part of the detected communities in the real-world dataset compared to the synthetic networks used for algorithm design. For this dataset, $\frac{1}{5}$ of the events of the dataset were excluded from the lower-bound work package computations using our approach. The properties of the network structure inside and outside communities are, by definition, dissimilar. Since the significant proportion of the network events left out of computation is both of a structurally dissimilar part of the network and involved in many potential

Computation	Time elapsed (seconds)
Work splitting	.906
WP 1	2954.245
WP 2	1263.374
WP 3	582.223
WP 4	520.837
WP 5	886.239
WP 6	417.579
WP 7	92.755
WP 8	71.695
WPs in sequence	6788.947
Ideal parallel run time	2955.151
Exact solution	53928.662

Table 5.1: Execution times of lower-bound motif discovery, on the dataset *email-Eu-core-temporal*, for one instance of work splitting resulting in 8 community-based work packages (WPs), in comparison to the exact solution using Kovanen's implementation.

network motifs, this leads to a situation where the results of the lower-bound computation may not be enough to give a representative approximation of the relative frequency of the motifs of the full network.
Attempting to compute the work packages to detect the difference between the lower bound and upper bound of motif counts in this situation revealed a fundamental flaw in our basic assumptions. Part of these unformulated assumptions were that the nodes not forming part of the communities were a relatively *small number*. Since we could further execute each upper-bound work package in a fraction of the execution time of exact computation of the same size work package, we had little concern for generating large work packages. However, even a small real-world dataset, such as *email-Eu-core-temporal*, proved to have a *large number* of such nodes, approximately $\frac{1}{5}$ of 25571 nodes, or around 5000. This proved to be a significant problem, further aggravated by the fact that we had also underestimated *how* large the resulting networks would be for a real-world dataset. The real-world datasets turned out to have a significantly lower *degree of separation* than our synthetic networks.
There exists a famous, but unproven, hypothesis called *six degrees of separation*. The hypothesis could be formulated that in an undirected graph where the nodes represent all people, and the edges represent which people have met, any two nodes are separated by at most 6 other nodes, or $6 + 1 = 7$ edges. Research done on the international social network Facebook shows that the degree of separation in such networks is lower, and, in Facebook's case, closer to 4 degrees [51].
In practice, the low degree of separation of real-world social networks meant that when selecting those nodes within a motif size's reach of any node, almost every node of the network was selected. This meant that on the real-world dataset, the approach for creating the second type of work package created thousands of work packages, each of almost the full size of the original work package for the entire network.

6
Conclusion

In this feasibility study, we tested the feasibility of parallelization of Kovanen's algorithm for temporal motif discovery by implementing a simple approach to thread parallelism in Kovanen's original implementation of his algorithm. We performed benchmarks of said implementation on the real-world dataset *email-Eu-core-temporal* and found that the parallelizable proportion of execution time $p > 0.89$. This shows that Kovanen's algorithm has great potential for parallelization.
Secondly, we designed and tested an approach to distributed temporal motif discovery based on Kovanen's original implementation of his algorithm. This approach was based on taking advantage of certain expected properties of community structured networks. Our methodology for developing the algorithm followed the experimental approach and made use of arbitrarily chosen synthetic networks of reduced size, which demonstrated or didn't demonstrate community structure, to test the design. Based on the success of the approach on these synthetic networks, we found a plausible design for computing upper and lower bounds of the motif counts of a community-structured network in a distributed fashion. This approach, if valid, could potentially be refined to computing exact results. Testing the design on the real-world dataset *email-Eu-core-temporal* showed that our approach for distributed computation of lower bounds of motif counts worked, running in significantly shorter time than the full computation even in sequence, but that the bounds were less tight than expected. However, this design appeared not to be feasible for computing the upper bounds of motif counts of real-world datasets.
Our results suggest that designing a functional algorithm for distributed computation of temporal motifs, especially larger motifs, based on Kovanen's algorithm and sectioning the temporal network, may require careful consideration in how to share computations involving any given path through the graph among multiple computational nodes.

References

[1] F. Harary and R. Z. Norman, *Graph theory as a mathematical model in social science*. University of Michigan, Institute for Social Research Ann Arbor, 1953. [Online]. Available: http://www.idiosophy.com/wp-content/uploads/2017/07/harary-norman.pdf.

[2] P. Holme and J. Saramäki, "Temporal networks," *Physics Reports*, vol. 519, pp. 97–125, 3 Oct. 2012, ISSN: 0370-1573. DOI: 10.1016/J.PHYSREP.2012.03.001.

[3] R. Pastor-Satorras, C. Castellano, P. V. Mieghem, and A. Vespignani, "Epidemic processes in complex networks," *Reviews of Modern Physics*, vol. 87, pp. 925–979, 3 Aug. 2015. DOI: 10.1103/RevModPhys.87.925. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.87.925.

[4] L. Zhao, G. J. Wang, M. Wang, W. Bao, W. Li, and H. E. Stanley, "Stock market as temporal network," *Physica A: Statistical Mechanics and its Applications*, vol. 506, pp. 1104–1112, Sep. 2018, ISSN: 0378-4371. DOI: 10.1016/J.PHYSA.2018.05.039.

[5] Y. Zhang, L. Wang, Y.-Q. Zhang, and X. Li, "Towards a temporal network analysis of interactive wifi users," *EPL (Europhysics Letters)*, vol. 98, p. 68 002, 6 Jun. 2012, ISSN: 0295-5075. DOI: 10.1209/0295-5075/98/68002. [Online]. Available: https://iopscience.iop.org/article/10.1209/0295-5075/98/68002.

[6] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. 10 008, Oct. 2008. DOI: 10.1088/1742-5468/2008/10/P10008. arXiv: 0803.0476 [physics.soc-ph].

[7] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks," *Science (New York, N.Y.)*, vol. 298, pp. 824–827, Nov. 2002. DOI: 10.1126/science.298.5594.824.

[8] P. Liu, V. Guarrasi, and A. E. Sariyuce, "Temporal network motifs: Models, limitations, evaluation," *IEEE Transactions on Knowledge and Data Engineering*, p. 1, 2021, ISSN: 1558-2191. DOI: 10.1109/TKDE.2021.3077495.

[9] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki, "Temporal motifs in time-dependent networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2011, P11005, 11 Nov. 2011, ISSN: 1742-5468. DOI: 10.1088/1742-5468/2011/11/P11005. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-5468/2011/11/P11005.

[10] C. Song, T. Ge, C. Chen, and J. Wang, "Event pattern matching over graph streams," *Proc. VLDB Endow.*, vol. 8, pp. 413–424, 4 Dec. 2014, ISSN: 2150-8097. DOI: 10.14778/2735496.2735504. [Online]. Available: https://doi.org/10.14778/2735496.2735504.

[11] Y. Hulovatyy, H. Chen, and T. Milenković, "Exploring the structure and function of temporal networks with dynamic graphlets," *Bioinformatics*, vol. 31, pp. i171–i180, 12 Jun. 2015, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btv227. [Online]. Available: https://doi.org/10.1093/bioinformatics/btv227.

[12] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," Association for Computing Machinery, 2017, pp. 601–610, ISBN: 9781450346757. DOI: 10.1145/3018661.3018731. [Online]. Available: https://doi.org/10.1145/3018661.3018731.

[13] S. Yu, Y. Feng, D. Zhang, H. D. Bedru, B. Xu, and F. Xia, "Motif discovery in networks: A survey," *Computer Science Review*, vol. 37, p. 100 267, Aug. 2020, ISSN: 1574-0137. DOI: 10.1016/J.COSREV.2020.100267.

[14] P. E. Ross, "Why cpu frequency stalled," *IEEE Spectrum*, vol. 45, p. 72, 4 2008, ISSN: 1939-9340. DOI: 10.1109/MSPEC.2008.4476447.

[15] C. A. Navarro, N. Hitschfeld-Kahler, and L. Mateu, "A survey on parallel computing and its applications in data-parallel problems using gpu architectures," *Communications in Computational Physics*, vol. 15, pp. 285–329, 2 Feb. 2014, ISSN: 1815-2406. DOI: 10.4208/cicp.110113.010813a. [Online]. Available: https://www.cambridge.org/core/product/identifier/S181524060000493X/type/journal_article.

[16] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring User Influence in Twitter: The Million Follower Fallacy," in *In Proceedings of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM)*, Washington DC, USA, May 2010.

[17] L. M. Haas, "Two approaches to deadlock in distributed systems," AAI8128634, Ph.D. dissertation, 1981.

[18] M. Grohe and P. Schweitzer, "The graph isomorphism problem," *Commun. ACM*, vol. 63, no. 11, pp. 128–134, Oct. 2020, ISSN: 0001-0782. DOI: 10.1145/3372123. [Online]. Available: https://doi.org/10.1145/3372123.

[19] L. Babai, "Graph isomorphism in quasipolynomial time [extended abstract]," in *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '16, Cambridge, MA, USA: Association for Computing Machinery, 2016, pp. 684–697, ISBN: 9781450341325. DOI: 10.1145/2897518.2897542. [Online]. Available: https://doi.org/10.1145/2897518.2897542.

[20] B. D. McKay, "Practical graph isomorphism," *Congressus Numerantium*, vol. 30, pp. 45–87, 1981. [Online]. Available: http://users.cecs.anu.edu.au/~bdm/nauty/pgi.pdf.

[21] B. D. McKay and A. Piperno, "Practical graph isomorphism, ii," *Journal of Symbolic Computation*, vol. 60, pp. 94–112, 2014, ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2013.09.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747717113001193.

[22] T. Miyazaki, "The complexity of McKay's canonical labeling algorithm," in *Groups and Computation II*, ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 28, Providence, RI, USA: American Mathematical Society, pp. 239–256, ISBN: 978-1-4704-3986-6. DOI: https://doi.org/10.1090/dimacs/028.

[23] T. Junttila and P. Kaski, "Engineering an efficient canonical labeling tool for large and sparse graphs," in *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, D. Applegate, G. S. Brodal, D. Panario, and R. Sedgewick, Eds., SIAM, 2007, pp. 135–149. DOI: 10.1137/1.9781611972870.13.

[24] T. Junttila and P. Kaski, "Conflict propagation and component recursion for canonical labeling," in *Theory and Practice of Algorithms in (Computer) Systems – First International ICST Conference, TAPAS 2011, Rome, Italy, April 18–20, 2011. Proceedings*, A. Marchetti-Spaccamela and M. Segal, Eds., ser. Lecture Notes in Computer Science, vol. 6595, Springer, 2011, pp. 151–162. DOI: 10.1007/978-3-642-19754-3_16.

[25] S. Fortunato and C. Castellano, *Community structure in graphs*, 2007. DOI: 10.48550/ARXIV.0712.2716. [Online]. Available: https://arxiv.org/abs/0712.2716.

[26] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Phys. Rev. E*, vol. 69, p. 066 133, 6 Jun. 2004. DOI: 10.1103/PhysRevE.69.066133. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.69.066133.

[27] Y. Xu, T. Ren, and S. Sun, "Community detection based on node influence and similarity of nodes," *Mathematics*, vol. 10, no. 6, 2022, ISSN: 2227-7390. DOI: 10.3390/math10060970. [Online]. Available: https://www.mdpi.com/2227-7390/10/6/970.

[28] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: Guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 1, p. 5233, 2019. DOI: 10.1038/s41598-019-41695-z. arXiv: 1810.08473.

[29] J. Zhang, J. Fei, X. Song, and J. Feng, "An improved louvain algorithm for community detection," *Mathematical Problems in Engineering*, vol. 2021, pp. 1–14, Nov. 2021. DOI: 10.1155/2021/1485592.

[30] W. Lin, X. Xiao, X. Xie, and X. Li, "Network motif discovery: A gpu approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, pp. 513–528, 3 2017, ISSN: 1558-2191. DOI: 10.1109/TKDE.2016.2566618.

[31] S. Wernicke and F. Rasche, "Fanmod: A tool for fast network motif detection," *Bioinformatics*, vol. 22 9, pp. 1152–3, 2006.

[32] Z. R. M. Kashani, H. Ahrabian, E. Elahi, *et al.*, "Kavosh: A new algorithm for finding network motifs," *BMC Bioinformatics*, vol. 10, pp. 318–318, 2009.

[33] P. Ribeiro and F. Silva, "G-tries: An efficient data structure for discovering network motifs," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10, Sierre, Switzerland: Association for Computing Machinery, 2010, pp. 1559–1566, ISBN: 9781605586397. DOI: 10.1145/1774088.1774422. [Online]. Available: https://doi.org/10.1145/1774088.1774422.

[34] S. Khakabimamaghani, I. Sharafuddin, N. Dichter, I. Koch, and A. Masoudi-Nejad, "Quatexelero: An accelerated exact network motif detection algorithm," *PLOS ONE*, vol. 8, no. 7, pp. 1–15, Jul. 2013. DOI: 10.1371/journal.pone.0068073. [Online]. Available: https://doi.org/10.1371/journal.pone.0068073.

[35] X. Li, R. J. Stones, H. Wang, H. Deng, X. Liu, and G. Wang, "Netmode: Network motif detection without nauty," *PLOS ONE*, vol. 7, no. 12, pp. 1–9, Dec. 2012. DOI: 10.1371/journal.pone.0050093. [Online]. Available: https://doi.org/10.1371/journal.pone.0050093.

[36] P. Ribeiro, F. Silva, and L. Lopes, "Parallel discovery of network motifs," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 144–154, 2012, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2011.08.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731511001729.

[37] G. Xue, E. B. Suh, J. W. Touchman, W. Zhang, and T. Wang, "A parallel algorithm for extracting transcription regulatory network motifs," in *BIBE 2005. 5th IEEE Symposium on Bioinformatics and Bioengineering*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2005, pp. 193–200. DOI: 10.1109/BIBE.2005.8. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/BIBE.2005.8.

[38] J. Chen, W. Hsu, M. L. Lee, and S.-K. Ng, "Nemofinder: Dissecting genome-wide protein-protein interactions with meso-scale network motifs," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '06, Philadelphia, PA, USA: Association for Computing Machinery, 2006, pp. 106–115, ISBN: 1595933395. DOI: 10.1145/1150402.1150418. [Online]. Available: https://doi.org/10.1145/1150402.1150418.

[39] P. Wang, J. C. S. Lui, B. Ribeiro, D. Towsley, J. Zhao, and X. Guan, "Efficiently estimating motif statistics of large networks," *ACM Trans. Knowl. Discov. Data*, vol. 9, 2 Sep. 2014, ISSN: 1556-4681. DOI: 10.1145/2629564. [Online]. Available: https://doi.org/10.1145/2629564.

[40] B. Ribeiro and D. Towsley, "Estimating and sampling graphs with multidimensional random walks," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10, Melbourne, Australia: Association for Computing Machinery, 2010, pp. 390–403, ISBN: 9781450304832. DOI: 10.1145/1879141.1879192. [Online]. Available: https://doi.org/10.1145/1879141.1879192.

[41] D. G. Horvitz and D. J. Thompson, "A generalization of sampling without replacement from a finite universe," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 663–685, 1952, ISSN: 01621459. [Online]. Available: http://www.jstor.org/stable/2280784 (visited on 06/10/2022).

[42] J. A. Grochow and M. Kellis, "Network motif discovery using subgraph enumeration and symmetry-breaking," in *Research in Computational Molecular Biology*, T. Speed and H. Huang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 92–106, ISBN: 978-3-540-71681-5.

[43] X. Sun, Y. Tan, Q. Wu, B. Chen, and C. Shen, "Tm-miner: Tfs-based algorithm for mining temporal motifs in large temporal network," *IEEE Access*, vol. 7, pp. 49 778–49 789, 2019. DOI: 10.1109/ACCESS.2019.2911181.

[44] X. Sun, Y. Tan, Q. Wu, J. Wang, and C. Shen, "New algorithms for counting temporal graph pattern," *Symmetry*, vol. 11, no. 10, 2019, ISSN: 2073-8994. DOI: 10.3390/sym11101188. [Online]. Available: https://www.mdpi.com/2073-8994/11/10/1188.

[45] T. Bjurenlind and H. Hadi, "Modeling user mobility for edge clouds," M.S. thesis, Department of Computer Science and Engineering, Chalmers University of Technology, 2021.

[46] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*, http://snap.stanford.edu/data, Jun. 2014.

[47] G. Barlacchi, R. Larcher, A. Casella, *et al.*, "A multi-source dataset of urban life in the city of milan and the province of trentino.," *Scientific Data*, vol. 2, 2015, ISSN: 20524463. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-84959024723&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[48] Telecom Italia, *Telecommunications - sms, call, internet - mi*, 2015. DOI: 10.7910/DVN/EGZHFV. [Online]. Available: https://dataverse.harvard.edu/citation?persistentId=doi:10.7910/DVN/EGZHFV.

[49] Telecom Italia, *Telecommunications - mi to mi*, 2015. DOI: doi:10.7910/DVN/JZMTBJ. [Online]. Available: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/JZMTBJ.

[50] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: https://networkrepository.com.

[51] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna, "Four degrees of separation," in *Proceedings of the 4th Annual ACM Web Science Conference*, ser. WebSci '12, Evanston, Illinois: Association for Computing Machinery, 2012, pp. 33–42, ISBN: 9781450312288. DOI: 10.1145/2380718.2380723. [Online]. Available: https://doi.org/10.1145/2380718.2380723.