UNIVERSITÀ DEGLI STUDI DI MILANO

CORSO DI DOTTORATO IN INFORMATICA
XXXVII° CICLO

DIPARTIMENTO DI INFORMATICA

A Framework for Network Evolution

INF/01

Dottorando:
Alessia GALDEMAN

Relatore:
Prof.ssa Sabrina GAITO

Co-relatore:
Prof. Matteo ZIGNANI

Coordinatore del dottorato:
Prof. Roberto SASSI

A. A. 2023/2024

Contents

Chapter 1

Introduction

With the advent of the digital transformation that has led to any organization being flooded with data, the research field of data science is in high demand. According to the International Data Corporation (IDC), the global digital transformation market projected to reach 3.4 trillion by 2026 with a compound annual growth rate (CAGR) of 16.3% [1], and as a consequence the research field of data science is in high demand. This exponential growth reflects the urgent need for organizations to leverage data-driven insights to remain competitive in an increasingly digital landscape. In fact, according to Gartner, 91% of companies are currently engaged in some form of digital initiative and 87% of senior business leaders say digitalization is a priority, highlighting the widespread recognition of the importance of integrating technology into business operations [2, 3].

Network science In the life-cycle of a data science project, the core is represented by the knowledge discovery step where data scientists leverage any kind of theories, techniques, and algorithms from many different fields to extract knowledge from data. Among the many approaches, from statistics to data mining and machine learning, *network science* or complex network analysis has gained attention in the last years both from the theoretical and the application perspective. Network science is a multidisciplinary field examining complex structures and dynamic interactions within complex systems. The main focus of network science is to investigate the relationships between interconnected entities, typically represented as nodes (or vertices), connected by links (or edges) that represent interactions or dependencies. According to Barabasi [4], one of the pioneers of the field, network science is an attempt to understand networks emerging in nature, technology, and society using a unified set of tools and principles. The reason lies behind the great potential of modeling the vast amount of data we can obtain today with a graph-based data structure, that can capture patterns and structures unrevealed by other techniques. Modeling through networks — graphs — has been essential for improving our understanding of the fundamental principles governing interconnected systems. Network Science has shown its effectiveness across various fields, with relevant insights and achievements.

Network science's application The application of network science to *social science* is one of the earliest and most influential uses of network analysis, dating back to the pioneering work of sociologist Moreno [5, 6], who introduced in 1932 the concept of "sociograms" to map human interpersonal relationships. Then, social network analysis has flourished, producing a wealth of insightful studies across various domains of human interaction. Researchers have employed network science techniques to analyze polarization in climate change debates [7] and investigate the drivers of social influence in user migration from Twitter to Mastodon [8]. Other studies have delved into the digital behavior of Instagram users [9] and leveraged the network science concept of homophily to predict friendships [10]. The versatility of network analysis is further demonstrated by work characterizing gamers on Steam through their interactions [11]. Additionally, researchers have proposed novel centrality measures to rank lurkers—users who primarily consume content without actively participating—on platforms like Twitter and FriendFeed [12]. Other example of successful application of network science methods is *finance*, where network science has been applied to analyze financial stability, particularly in understanding interbank relationships and systemic risk. Studies have revealed how contagion channels can lead to financial crises, thereby emphasizing the interlinked nature of banks and markets [13]. Moreover, network science has proven to be a powerful tool in the fight against financial fraud. By representing transactions, user interactions, and product reviews as complex networks, researchers have developed sophisticated methods to detect suspicious patterns and identify fraudulent activities. These approaches leverage graph analysis techniques, such as identifying dense subgraphs and bicliques, combined with machine learning algorithms to uncover coordinated fraudulent behavior that may not be apparent when examining individual transactions or accounts in isolation [14, 15, 16]. In *biology* and *medicine*, network science has been instrumental in mapping protein-protein interactions, which are crucial for understanding cellular processes and disease mechanisms. This approach helps identify potential drug targets or combinations and understand disease pathways [17, 18]. Network models have also been used to study ecological networks, analyzing interactions among species and their environments, which helps in understanding biodiversity and the impacts of environmental changes on ecosystems.

Temporal networks Up to the very last years, research was mainly focused on static networks - where connections between entities remain unchanged over time - as they are much more manageable than dynamic networks in terms of complexity, scalability, and data availability. However, all large systems, whether financial, biological, social, or technological, are not only complex but also time-varying. Researchers began to focus their attention on a more complex type of network, that includes temporal information to catch when connections form, persist or dissolve, thus introducing the concept of *temporal networks.* Temporal networks represent the optimal modeling in domains like urban mobility, where timing data is essential for understanding traffic patterns and commuter behavior [19]. Brain networks rely heavily on temporal information to map neural impulses and their propagation [20]. Social networks leverage time data to model the spread of information and trends [21, 22, 23]. In financial studies, temporal context is vital for correlating transactions with external events such as exchange rate fluctuations or economic crises [24]. Similarly, disease-spreading networks depend significantly on infection timelines to track and predict outbreak patterns [25].
Network Evolution In this context, the literature is more focused on approaches that track the change of properties over time, describing how a network behaves snapshot by snapshot from a macroscopic point of view. This allows to highlight properties such as densification and shrinking diameters [26]. Observing different real graphs with regularly spaced snapshots, Leskovec *et. al.* [26] notice that most of the observed graphs densify over time, with the diameter (average distance between nodes) decreasing. While existing research on temporal networks has largely focused on describing how networks change over time, there has been less emphasis on understanding the underlying mechanisms that drive these evolutionary patterns. This gap in our understanding is critical, as unraveling the mechanisms behind network evolution is crucial for comprehending the current structure of temporal networks and predicting how they will grow. The literature on network mechanisms suggests that a comprehensive understanding of network evolution requires considering the mesoscopic level, rather than solely focusing on macroscopic properties. As Granovetter [27] noted, "*A fundamental weakness of current sociological theory is that it does not relate micro-level interactions to macro-level patterns in any convincing way. Large-scale statistical, as well as qualitative, studies offer a good deal of insight into such macro phenomena as social mobility, community organization, and political structure. At the micro level, a large and increasing body of data and theory offers useful and illuminating ideas about*

what transpires within the confines of the small group. But how interaction in small groups aggregates to form large-scale patterns eludes us in most cases." He argued that the most effective *micro-macro bridge* is offered by the analysis of processes within interpersonal networks. It is through these networks that small-scale interactions are translated into large-scale patterns, which then, in turn, feed back into small group dynamics [27]. Thus, a more efficient approach to understanding the evolution of a network lies in the mesoscopic layer, where the interplay between individual nodes and the larger network structure can be explored.

By extracting the mechanisms, named rules, of evolution from a temporal graph, we can reveal the general mesoscopic mechanisms governing network dynamics. This means understanding how microscopical and mesoscopic network structures evolve over time. The **main goal** of this thesis is to develop a comprehensive and versatile framework for modeling, mining, analyzing, and represent the evolutionary rules governing network dynamics. Our aim is to provide a robust methodology that can generate an evolutionary profile for a wide range of network types, offering insights into their growth patterns and structural changes over time. While some research has been conducted on specific network evolution mechanisms, there remains a need for a more general approach, without any a-priori assumption. For instance, triadic closure is a well-known evolutionary process assuming that individuals sharing a common friend are more likely to become friends themselves in the future [28]. However, our proposed framework seeks to go beyond individual mechanisms, integrating multiple factors and processes to create a more complete picture of network evolution.

Rules of evolution The evolutions in which each starting structure can change are called graph evolution rules (GER). Graph evolution rules mining is a frequency-based pattern discovery method used to examine the dynamics of networks as they evolve over time. GERs are designed to identify recurring local changes throughout the network's evolution [29]. Drawing inspiration from the notion of *association rules* in the context of data mining, GERs consist of two essential components: a precondition and a set of postconditions. These rules indicate that a subgraph matching (being isomorphic to) the precondition is likely to evolve into one of the configuration represented by the postconditions, with the corresponding probability. This approach yields results that are not only quantifiable but also readily understandable and interpretable to human observers. Figure 1.1 shows an example of the possible evolutions of an open triangle (i.e. two nodes connected to the same third node –triangle– that are not linked with each other –open). The evolution at the center represents the typical triadic closure process, where the non-connected nodes in the triangle get a link. However, as mentioned before, the open triangle can evolve in more complex structures, for instance, it can close with a reciprocal connection like in the first evolution, or it can be connected with another external node at the same time. Graph evolution rules can capture the complexity of network

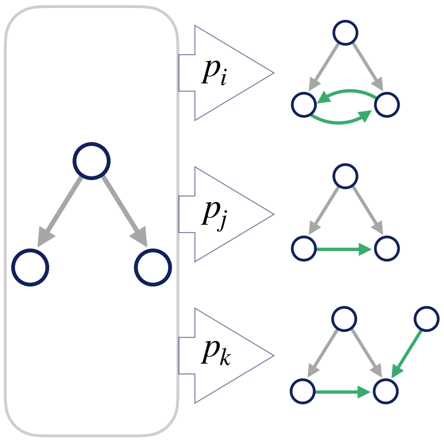


Figure 1.1: Evolutionary profile generated by the TULIP algorithm. *The profile shows a starting subgraph (on the left) and its three most frequent evolutions, each associated with probabilities p_i, p_j, and p_k respectively.*

evolution by counting all the possible evolutions started from a configuration (like the open triangle in the example) and building a probability distribution over the evolutions. In this way, it is possible to have a general overview of mesoscopic structure evolutions.

Despite its potential, the study of network evolution through its rule of evolution is just at the beginning. This emerging field faces significant challenges that have limited its progress so far. At the core of these difficulties lies the complexity of identifying and analyzing Graph Evolution Rules (GERs). This process essentially constitutes a subgraph mining problem, demanding precise enumeration and counting of occurrences within the network. The most challenging aspect of this task is the *isomorphism check*. An isomorphism in mathematics is a structure-preserving mapping that enables the reversal of two structures of the same type. So, two graphs are considered isomorphic if there exists a bijective function that maps the nodes of the first graph into the nodes of the second one. This isomorphism check is crucial because to accurately count occurrences of the same pattern within a network, we must determine whether seemingly different subgraphs are structurally identical. This is not a straightforward comparison, as two isomorphic graphs may have entirely different representations in terms of node labeling or edge ordering, yet still maintain the same underlying structure. The complexity of this check becomes a significant bottleneck in the analysis process. Furthermore, to ensure the comparability of results across different studies and implementations, a *standardized canonical form* for these subgraphs is necessary. This canonical form serves as a common representation that is unique for each isomorphic graph. By establishing such a form, we can compare results over different runs of the algorithm without the need for additional isomorphism checks later in the process. This standardization not only improves efficiency but also facilitates reproducibility and cross-study comparisons, which are essential for the advancement of network evolution research.

Only a few works exist in the domain of Graph Evolution Rules (GER) mining, with most state-of-the-art methods following a two-step process. Initially, rules are derived through a frequent subgraph mining (FSM) algorithm, followed by filtering results using metrics such as support and confidence. Usually, the FSM algorithm is a modified version of gSpan [30], a leading method renowned for its efficiency. This algorithm processes a collection of undirected static graphs and outputs connected frequent subgraphs. The pioneering algorithm in this field is GERM, proposed by Berlingerio *et. al.* [29]. It focuses on undirected edge insertion events while considering relative temporal order. Leung *et. al.* [31] introduced the concept of Link Formation Rules (LFR) for directed networks, which was later adopted and extended by Ozaki *et. al.* [32] to include undirected networks and rule relationships. Both GERM and LFR utilize minimum image-based support [33] and gSpan for frequent subgraph mining. Further advancements came with Vakulík's DGR miner [34], which incorporated edge deletion and relabeling into its evolution rules. Finally, EvoMine, developed by Scharwächter *et. al.* [35], represents a more comprehensive approach, dealing with complex evolution patterns beyond simple edge insertions and introducing a novel support measure. While the aforementioned algorithms have laid the groundwork for GER mining, they predominantly focus on what we will refer to as *stand-alone rules*. These rules describe isolated evolutionary processes by enumerating instances of one subgraph transforming into another. Unlike more general graph evolution rules (depicted in Figure 1.1) that show multiple possible evolutions, a stand-alone rule depicts a single evolution from a pre-condition to a post-condition, with an associated frequency. Figure 1.2 shows the example of a stand-alone rule, that corresponds to one of the possible evolutions captured by the rule of Figure 1.1. Drawing from association rule terminology, the pre-condition (or body) represents the initial subgraph, while the post-condition (or head) represents the evolved state. The rule's semantic implication is that a subgraph matching the pre-condition is likely to evolve into one matching the post-condition, with the probability conditioned on the occurrence of the pre-condition.

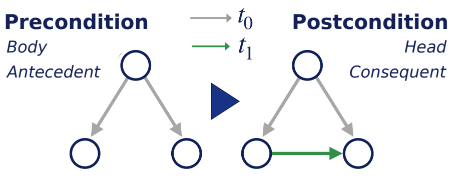


Figure 1.2: *Stand alone rules example*

While this approach offers valuable insights into specific evolutionary occurrences, it has some limitations. Firstly, these algorithms often lack a comprehensive analytical framework for extracting meaningful insights from the results. This limitation makes it difficult to interpret and use the mined rules effectively. Secondly, the absence of a standardized canonical form for rules requires an additional isomorphism check to compare results across different studies or implementations. This limitation is a problem for reproducibility and cross-study analyses. Lastly, these methods typically do not incorporate a tailored null model, which is crucial for identifying statistically significant rules and distinguishing them from random occurrences.

Goals and contributions This thesis presents a comprehensive and versatile framework for modeling, mining, analyzing, and representing the evolution rules that shape network dynamics. Our approach covers both standalone rules and more general ones, offering an unprecedented, complete toolkit to analyze network evolution. This framework, named GERANIO (GEneral fRAmework for Network evolutIOn), is available online here. The main contributions of this thesis, provided by GERANIO and depicted in Figure 1.3, are the following:

- **Framework for evolving networks**: We propose a universal taxonomy for modeling evolving networks, applicable across various domains. This taxonomy addresses the diverse modeling approaches found in existing stand-alone algorithms and extends to related topics (such as user migration);

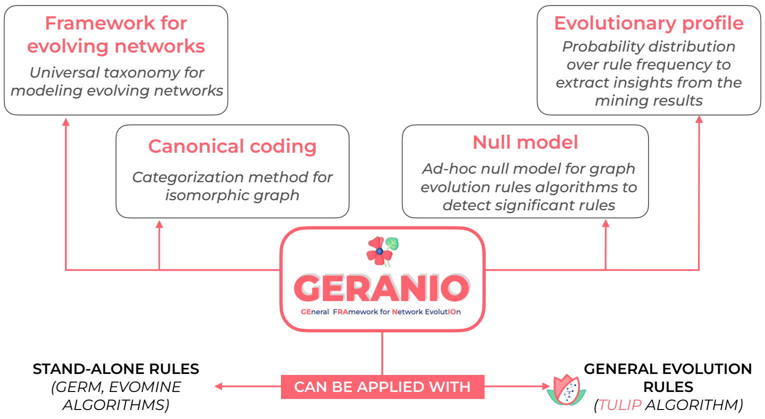

Framework for evolving networks
Universal taxonomy for modeling evolving networks
Evolutionary profile
Probability distribution over rule frequency to extract insights from the mining results
Canonical coding
Categorization method for isomorphic graph
Null model
Ad-hoc null model for graph evolution rules algorithms to detect significant rules
GERANIO
GEneral FRAmework for Network EvolutIOn
STAND-ALONE RULES
(GERM, EVOMINE ALGORITHMS)
CAN BE APPLIED WITH
GENERAL EVOLUTION RULES
(TULIP ALGORITHM)

Figure 1.3: *The main contributions of the geranio framework*

- **Canonical coding**: To facilitate comparison of graph evolution across different networks, we introduce an efficient categorization method for isomorphic subgraphs, ensuring universal applicability of results;
- **Null model**: We propose ad-hoc null models, designed specifically for graph evolution rules algorithms, to extract significant rules by evaluating their statistical significance beyond mere frequency, thus allowing us to distinguish meaningful patterns from random noise in the network's evolution;
- **Tulip algorithm**: The novel TULIP (Temporal subgraphs for evolutionary profiling) algorithm mines graph evolution rules from a broader perspective than stand-alone approaches. It captures all possible subgraph evolutions along with their probabilities, offering a comprehensive evolutionary footprint of a network;
- **Evolutionary profile**: To extract insights from the mining results, we propose the evolutionary profile—a probability distribution of rule frequencies. For general graph evolution rules, this translates into a multidimensional profile describing the probability of subgraph transformations. These profiles can be stacked across different networks, providing immediate visual feedback on the evolving behavior of graphs (but also nodes, or groups).

These contributions collectively aim to enhance our understanding of network dynamics and provide powerful tools for comparing different networks or analyzing changes over time. The following part will elaborate on these contributions, distinguishing between the two types of graph evolution rules: stand-alone and general.

Stand-alone rules Despite the limitations, stand-alone rules fits perfectly when the goal of the research is to find the frequency of some known evolutionary processes. For instance to look for cycles in a financial network. Recognizing both the potential and limitations of stand-alone rules, our research began by addressing the gaps in this area, providing to the GERANIO framework tools to mine stand-alone rules more effectively and also leverages them to provide a fuller understanding of graph evolution dynamics. First, this includes a canonical form, i.e. a unique representation of the graph, allowing for easy comparison of graphs for isomorphism. In this form, each graph is converted to a standardized, normalized format that remains consistent regardless of the initial labeling or representation of the graph. We leveraged the bliss algorithm [36] applied on a multilayer version of each temporal subgraph. The idea is depicted in Figure 1.4: each of the three nodes of the original temporal subgraph is duplicated as many times as the maximum timestamp (plus one because it starts from zero). Gray edges (called pillar edges) connect nodes that represent the same entity on different layers (for instance $(1, 4), (4, 7)$ while blue edges represent the actual edges of the original graph, positioned between nodes of the layer corresponding to their timestamp. For instance the link from node 1 to node 2 is translated in the edge $(4, 5)$ because its timestamp is 1 (second layer). In this way, we have a static, non labeled representation of the temporal graph that can be processed by the bliss algorithm. This returns a permutation that generates the canonical form for the given graph. Finally, as unique identifier of each isomorphism class, we use an integer representation of the canonical form's adjacency matrix. This unique code for each isomorphic graph could make results over different networks comparable.

Second, the stand-alone part of the GERANIO framework introduces an ad-hoc null model to extract statistically significant graph evolution rules from graph, to distinguish relevant rules from frequent ones. Finally, GERANIO offers an analytical tool to extract insights from algorithms outputs. For instance, we proposed a profile for the network based on graph evolution rules, called GER profile. It is a probability distribution over rules frequency that allows fast comparison of different networks' evolutionary behavior.

General graph evolution rules While stand-alone rules provide valuable insights in certain contexts, they often fall short of capturing the complete evolutionary dynamics of a network. To address this limitation, the GERANIO framework incorporates tools designed for the general graph evolution rules. Central to this approach is TULIP, an innovative algorithm that captures all possible subgraph evolutions along with their associated probabilities, offering

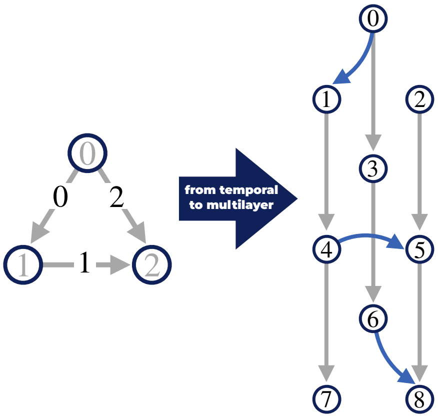


Figure 1.4: From temporal to multilayer. *Example of a temporal subgraph represented as a multilayer graph. The number of layers corresponds to the range of timestamps (three layers in this case, with timestamps from 0 to 2), and nodes are replicated in each layer. The graph includes pillar edges (gray), which connect the replicated nodes, and the actual edges of the temporal subgraph are placed in the layer corresponding to their timestamp.*

a comprehensive evolutionary footprint of a network. Beyond the TULIP algorithm, GERANIO provides the complete tool for categorizing, analyzing, and visualizing network evolution rules, enabling deeper insights into the underlying dynamics of temporal networks' evolution. This includes for instance an extension of the canonical coding method designed for stand-alone rules, to accommodate general evolution rules, ensuring consistent representation across different network types and sizes. A key feature of this comprehensive approach is the evolutionary profile, which aggregates probability distributions across all rules. This profile serves as a powerful comparative tool: networks with similar profiles likely evolve in similar ways. The flexibility of the evolutionary profile allows for diverse applications. For instance, it can be used to examine how a single network evolves differently across various time periods, or to compare evolution patterns across multiple networks.

Thesis structure This thesis is structured into four main parts, each addressing different aspects of network evolution analysis. The **first part** provides a comprehensive background, describing the unified framework for temporal networks modeling and the theoretical foundation for the subsequent parts of the thesis. This background section also includes an overview of Graph Evolution Rules (GER) terminology and introduces the concept of microcanonical randomized models (null models). The **second part** delves deeper into the GERANIO framework with its part dedicated to stand-alone rules, exploring its enhancements and applications. The **third part** introduces a broader perspective on graph evolution rules. This section completes the GERANIO framework with the TULIP algorithm, which aims to capture a more comprehensive view of network dynamics. The **fourth part** addresses an evolutionary aspect external to the networks that cannot be fully captured by graph evolution rules alone, but has become increasingly significant in the wake of the Twitter-X transition: user migration. This phenomenon involves the mass movement of users from one platform to another, often in response to changes in policy, ownership, or functionality. It is particularly relevant in the Web3 context, where hard forks in blockchain networks can trigger structured and observable user migrations between competing platforms. In this thesis, this final part explores the unique challenges and implications of user movement within decentralized networks, complementing the graph-based analyses of the previous chapters.

1.1 Unified modeling framework and terminology

The first part of this thesis establishes a unified framework with the theoretical foundation that will be mentioned in the subsequent parts. It begins by introducing a comprehensive and unified framework for temporal networks modeling, providing a common terminology and conceptual grounding that will be used consistently throughout the work. This is a crucial step, as the existing literature on network evolution often employs different models and terminology, even when referring to the same concept. After laying this foundational framework, the background section delves into the theoretical concepts of graph evolution rules. These rules are the network equivalent of association rules in the data mining field, so we start by detailing the key concepts of association rules. This is followed by an overview of subgraph counting methods: we cover techniques such as frequent subgraph mining, motifs, and graphlets, in both their static and temporal variants, highlighting how they differ from graph evolution rules in their subgraph counting approach. Next, we examine the common concepts that form the basis of the various works on graph evolution rules. This includes how a rule is composed, the primary support measure used (minimum image based), and the property of anti-monotonicity. We also provide a detailed explanation of the main mining algorithm employed in existing graph evolution rule studies: gSpan [30]. The background section then presents a comprehensive review of the key existing works in this domain, including GERM[29], LFR[31, 32], DGR[34], TPminer, and Evomine[35]. This ensures that the reader has a thorough understanding of the state-of-the-art approaches and their respective contributions. Then, a dedicated section is included to explain the concept and categorization of null models (microcanonical randomized reference models) in the context of temporal networks, following the work by Gauvin *et al.*[37]. The final chapter presents case studies applying our theoretical framework, with several of our research works sharing common datasets and modeling approaches. We focus particularly on the Steemit platform as a representative of the Web3 ecosystem, demonstrating how graph evolution rules can uncover insights in complex digital environments that integrate social and financial interactions.

1.2 Stand-alone evolution rules

The literature on graph evolution rules is relatively limited, with existing works often presenting algorithms without comprehensive analytical tools. The algorithms that have been proposed typically suffer from various limitations in their scope and applicability. The second part of this thesis seeks to address these shortcomings by enhancing the part of the GERANIO framework dedicated to stand-alone rules, addressing key limitations, and introducing novel analytical tools. Our work begins with the development of a tailored null model for the GERM algorithm. Drawing from the categorization proposed by Gauvin *et al.* [37], we introduce a timeline-shuffled null model. This model prioritizes the preservation of network topology over temporal distribution and is applied to timeline representations of temporal graphs. This approach aligns with GERM's graph representation, but also with the scope of the work to analyzing the characteristics of the dynamics of the network, rather than its specific topology. We apply the null model on three distinct networks with varying time granularities, two social networks and one citation network. The results demonstrate the significant impact of this null model on the evaluation and interpretation of identified rules, revealing the prevalence of under-represented rules and suggesting that temporal factors and other mechanisms may influence evolutionary paths in complex networks. Building upon this foundation, we introduce the GER profile, an analytical tool designed to facilitate easy comparison of evolutionary behaviors across networks. This probability distribution over stand-alone rule frequencies allows for the characterization of network growth in diverse contexts. In the NFT category, we analyze two networks representing transactions on popular yet contrasting markets. The first

is CryptoKitties, a game-based NFT platform, while the second is OpenSea, a marketplace with a broader, more general scope. For the BOSN category, we focus on two networks derived from the Steemit platform. One network encodes financial transactions between users, while the other captures follow operations, thus classified as a social network. Our analysis reveals that while some rules are consistently frequent across all networks, others are closely tied to the specific nature of each platform. For instance, we observed that rules exhibiting strongly reciprocal traits were particularly frequent in the network with a predominantly social nature (follow relationships in Steemit). We then extend the application of the GER profile to multiple scales, exploring its utility not only at the graph level (like in the previously described Web3 context) but also at community and node levels. At the community level, we construct GER profiles for individual communities and compare them across four diverse networks, including communication, social, citation, and online discussion networks. Our findings indicate that communities evolving in similar ways tend to be close in the network. Finally, we leverage the GER approach to develop a temporal behavioral node representation, using the GER profile of ego networks. As a case study, we analyze the Sarafu network, a complementary currency platform with rich temporal data. These Node Evolutionary Profiles (NEPs) enable the identification of groups of nodes characterized by similar evolution rules, revealing common interaction patterns and providing insights into user behavior within the network. Through these enhancements and applications, the first part of this thesis advances the stand-alone part of the GERANIO framework, offering new tools and perspectives for understanding network evolution at multiple scales.
Part II is partially based on the following publications:

- Alessia Galdeman, Matteo Zignani, Sabrina Gaito. *Unfolding temporal networks through statistically significant graph evolution rules. 2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA),* (IEEE, 2023).
- Alessia Galdeman, Matteo Zignani, Sabrina Gaito. *Disentangling the growth of blockchain-based networks by graph evolution rule mining. 2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA),* (IEEE, 2022).
- Alessia Galdeman, Matteo Zignani, Sabrina Gaito. *Graph evolution rules meet communities: assessing global and local patterns in the evolution of dynamic networks. Big Data Mining and Analytics* (2024).

- Alessia Galdeman, Matteo Zignani, Christian Quadri, Sabrina Gaito. *Graph evolution rules for node temporal behavior representation. Accepted at the International Conference on Discovery Science* (2024).

1.3 General graph evolution rules

The third part of this thesis focuses on the graph evolution rules from a broader perspective, completing the **GERANIO** framework with the **TULIP** algorithm. We dedicate a different part of the thesis to this topic alone to stress the difference between stand alone rules and general graph evolution rules offered by the TULIP algorithm. While stand-alone rules are valuable in certain contexts, they often fall short in capturing the full complexity of network evolution. The TULIP algorithm addresses this limitation that builds upon the terminology and concepts introduced in the background section. To the best of our knowledge, this approach is unprecedented in the field. Unlike algorithms for stand-alone rules, our method can capture all possible ways a subgraph evolves, along with their corresponding probabilities. The pipeline of the algorithm is shown in Figure 1.5. TULIP begins by examining a set of small, static

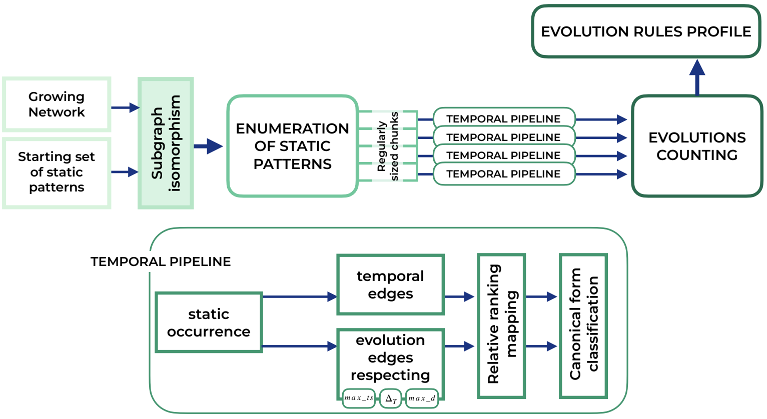


Figure 1.5: Pipeline of the Tulip algorithm*. The algorithm begins by enumerating all occurrences of a given set of static patterns. The results are then divided into chunks, and through the temporal pipeline, Tulip extracts pairs of evolutions in the form* $(pre, post)$*. These evolutions are subsequently counted, and the evolutionary profile is constructed*

subgraphs. This approach is grounded in the well-established principle that network evolution can be effectively studied through mesoscopic structures, with even three-node configurations providing significant insights [38, 39, 40]. We then employ the VF2 algorithm [41] to enumerate all instances of these static subgraphs within the larger network. For example, starting with a two-node structure with reciprocal edges, VF2 identifies all node pairs in the initial graph that match (are isomorphic) to this pattern. Once we have the enumeration of all the static patterns, we divide the occurrences in regularly sized chunks and apply the temporal part in parallel. The temporal pipeline takes each occurrence of a static pattern (so for instance the couple of edges $((x, y), (y, z))$ respecting the 2-edges path pattern) and first extracts the timestamps on the edges obtaining the temporal occurrence (so $((x, y, t_1), (y, z, t_2)))$, this will represent the precondition of our evolution (the starting point). After that, we need to build the post condition, collecting all the edges of the evolution: edges starting or ending in one of the node of the precondition, happened in a timestamp between $max\_ts + 1$ and $max\_ts + 1 + \Delta_t$ where max_ts is the maximum timestamp of the precondition (following the previous examples it would be $max(t_1, t_2)$) and Δ_t is an interval of time. To mitigate the noise of hub nodes, we limit the number of edges starting and ending at each node at max_d. Once the temporal edges and the evolutions are obtained, the couple of edgelists is processed.We first apply a relative-ranking mapping to the timestamps, normalizing them to start from 0 (relative) and having the maximum value to correspond with the number of distinct timestamps (ranking). Finally, we extract the canonical code identifier, as explained in the previous section, to generate comparable codes for both the temporal edges and the evolutions. Finally, the algorithm yields pairs of starting temporal subgraphs and their evolutions canonical codes with the corresponding counting. From this, we construct evolution profiles for each starting temporal subgraph, encompassing all observed evolutions and their probability distributions. We validate our method by applying it to five diverse graphs, varying in nature, size, and temporal coverage. This application demonstrates the method's capacity to extract meaningful insights into network evolution across different contexts. The final section of this part is focused on the added value of graph evolution rules in evolutionary analysis. We provide a comparative analysis with existing evolutionary analysis techniques, highlighting the unique contributions and insights offered by our approach.

1.4 User Migration

The fourth part of this thesis explores an evolutionary aspect typical of Web3 networks that cannot be fully captured by graph evolution rules alone: user migration. This phenomenon is particularly relevant in the context of blockchain-based online social networks (BOSNs) and presents unique challenges for network analysis.

We use as a case study the hard fork happened on the Blockchain-Online social network (BOSN) Steemit, that caused the birth of Hive. Our first study focuses on two key aspects: the propensity of hubs to migrate to a new social platform following a shock event, and the influence these hubs exert on their neighbors' migration decisions. Our findings reveal that different types of hubs employ varied strategies when choosing to migrate. For instance, financial hubs (nodes with highest number of financial operations) tend to diversify their approach by maintaining a presence on both the original and new platforms. Regarding hub influence, we observe that users directly interacting with hubs are more likely to migrate. These results underscore the critical role of understanding hub activity and influence in monitoring and managing user migration processes.

Building on this, we delve into the intricate interplay between groups, discussions, and migration patterns. Through a network-based analysis centered on community identification in multilayer networks, combined with text mining techniques, we uncover several key insights:

- The position of a group within the network of social and economic interactions correlates with its likelihood to migrate, with marginal groups showing a higher propensity to leave.
- The network structure of groups plays a significant role, as users in densely connected groups with strong financial interactions are more likely to remain on the original platform.
- Users who choose to leave exhibit distinct discussion topics compared to those who stay.
- User groups engaged in monetary transactions show increased interest in migration-related content if they are considering leaving.

These findings highlight the crucial role of social and economic relationships during user migration triggered by fork events. In the broader context of online social media, this motivates the need for a network-inspired approach to investigating user migration, focusing on groups and specific subgraphs while leveraging user-generated content.

The final section of this part introduces a novel machine learning pipeline utilizing graph neural networks (GNNs) to predict user migration in BOSNs. We model the data as a directed temporal multilayer graph, capturing both social and monetary interactions among users. To address the challenge of class imbalance in node classification, we propose a data-level balancing technique based on undersampling. Our evaluation demonstrates that GNNs are a suitable machine learning approach for user migration prediction. Moreover, our proposed undersampling approach significantly enhances predictive power on severely imbalanced data.

Part IV is based on the following publications:

- Alessia Galdeman, Matteo Zignani, Sabrina Gaito. *User migration across web3 online social networks: behaviors and influence of hubs. ICC 2023-IEEE International Conference on Communications* (IEEE, 2023).
- Cheick Tidiane Ba, Manuel Dileo, Alessia Galdeman, Matteo Zignani, Sabrina Gaito. *Analyzing user migration in blockchain online social networks through network structure and discussion topics of communities on multilayer networks. Distributed Ledger Technologies: Research and Practice* (2024).
- Cheick Tidiane Ba, Alessia Galdeman, Manuel Dileo, Matteo Zignani, Sabrina Gaito. *User migration prediction in blockchain socioeconomic networks using graph neural networks.* Proceedings of the 2023 ACM Conference on Information Technology for Social Good, 2023, Lisbon, Portugal.

1.5 Conclusion

In this thesis, we propose a comprehensive and versatile framework for modeling, mining, analyzing, and representing the evolutionary rules governing network dynamics, named GERANIO. Our primary aim was to provide a robust methodology capable of generating an evolutionary profile for a wide range of network types, offering unprecedented insights into their growth patterns and structural changes over time. As we conclude this research, we can confidently assert that the framework we have developed represents a significant advancement in the field of network science. Unlike previous approaches that often focused on specific aspects or mechanisms of network evolution, our methodology offers a holistic and adaptable solution that can be applied across diverse domains. Our work has yielded several key contributions that collectively advance the field of network science and provide valuable tools for researchers and practitioners alike. We begin with the GERANIO framework part dedicated to stand-alone rules, introducing a canonical form, a null model, and an analytical tool in the form of the GER profile. This advancement provides a more robust foundation for analyzing individual graph evolution rules. A major contribution of this thesis is the TULIP algorithm, which enables the counting of all possible evolutions a subgraph can undergo, along with their corresponding probability distributions. This approach offers a more nuanced and complete view of network dynamics. The thesis is structured to provide a thorough exploration of these concepts. It begins with a rich background section that deepens our understanding of network evolution and presents a unified GER framework, including models and terminology used throughout the work. Subsequent parts focus on the GERANIO framework for stand-alone rules and general graph evolution rules respectively. The final part of the thesis shifts focus to a dynamic process of particular importance in Web3 environments: user migration. This section analyzes the influence of hubs and groups on migration decisions and presents a machine learning pipeline for predicting user migration at the node level. This thesis opens up several promising paths for future research, both in practical application of the graph evolution rule approach and in enhancing the rule mining process. One key direction is the use of evolutionary profiles for detecting shock events or change points in network evolution, as well as for generating synthetic networks with realistic evolutionary characteristics. This represents a practical application of graph evolution rules approach, given thanks to the GERANIO framework. The integration of artificial intelligence also presents promising possibilities, both in employing AI to assist the mining of graph evolution rules (for instance introducing an approximate counting) and in using evolutionary profiles as features in machine learning tasks. Further exploration of network evolution from a user-centric perspective, analyzing individual behaviors and their impact on overall network dynamics, could yield valuable insights. Additionally, investigating the interplay between network structure, user behavior, and external events in shaping the evolution of complex networks, particularly in the context of emerging Web3 technologies, promises to deepen our understanding of these systems. By addressing these research directions, we can continue to deepen our understanding of network evolution, enhancing our ability to model, predict, and interpret the dynamics of complex systems across various domains.

Part I

Background and Related works

The first part of this thesis establishes a unified framework and theoretical foundation that will be leveraged in the subsequent chapters.

It begins with a comprehensive literature overview of network evolution studies, providing context for the work that follows.

The chapter then introduces a comprehensive and unified framework for modeling evolving networks, offering common terminology and conceptual grounding to be used consistently throughout the work. This is a crucial step, as existing literature on network evolution often employs varied models and terminology, even when referring to the same concepts. By establishing this shared language, we aim to facilitate clearer communication and comparison of ideas within the field.

Following this foundational framework, the background section delves into the theoretical concepts of graph evolution rules. These rules are presented as the network equivalent of association rules in data mining. We begin by detailing key concepts of association rules, providing readers with a familiar starting point before transitioning to the more specialized domain of graph evolution. The chapter then provides an overview of subgraph counting methods, covering techniques such as frequent subgraph mining, motifs, and graphlets in both their static and temporal versions. We highlight how these methods differ from graph evolution rules in their approach to subgraph counting, emphasizing the unique characteristics and advantages of graph evolution rules. Next, we examine the common concepts that form the basis of various works on graph evolution rules. This includes an exploration of rule composition, the primary support measure used (minimum image based), and the property of anti-monotonicity. We provide a detailed explanation of the main mining algorithm employed in existing graph evolution rule studies: gSpan. This thorough examination of the underlying principles and methodologies ensures that readers have a solid grasp of the technical foundations of the field. The graph evolution chapter then presents a comprehensive review of key existing works in this domain. This includes detailed discussions of GERM [29], LFR [31], DGR [34], TPminer [42], and Evomine [35]. By examining these state-of-the-art approaches and their respective contributions, we provide readers with a thorough understanding of the current landscape of graph evolution rule research and set the stage for our own contributions.

Then, we include a dedicated section explaining the concept and categorization of null models, specifically microcanonical randomized reference models, in the context of temporal networks. This discussion follows the work by Gauvin et al.[37], providing crucial context for understanding the significance and interpretation of network patterns and evolution rules. By providing this comprehensive theoretical foundation, literature review, and explanation of key concepts, the first part of the thesis equips readers with the necessary background to fully engage with and appreciate the novel contributions presented in subsequent chapters. This thorough grounding ensures that the more advanced concepts and methodologies introduced later in the thesis are built upon a solid understanding of the field's current state and fundamental principles.

The final chapter in this part focuses on case studies, providing practical applications of the theoretical concepts discussed earlier. Several of our research works presented in this thesis share common datasets and modeling approaches, allowing us to present them cohesively. We offer additional details about the semantic aspects of the graph representations and provide relevant background information for each case study. A particular emphasis is placed on the Steemit platform, which serves as a representative example of the Web3 ecosystem. Steemit's unique features, including its blockchain-based infrastructure and integration of social and financial interactions, make it an especially rich case study for applying graph evolution rules.

Chapter 2

Temporal and evolving networks

The study of network evolution is a dynamic and multifaceted field that seeks to understand how complex systems of interconnected entities change over time. While traditional network analysis often focuses on static snapshots, the temporal aspect of network growth is crucial for capturing the true nature of evolving systems. Network evolution can be approached through various lenses, from examining individual processes like triadic closure and homophily to considering broader, system-wide patterns of growth. A comprehensive understanding of network evolution requires us to move beyond static analyses and embrace the temporal dimension of network dynamics. This temporal perspective allows us to observe how local interactions and processes aggregate over time to shape the global structure of the network. For instance, the principle of triadic closure—where two nodes connected to a common third node are likely to form a connection—can be studied not just as a static property, but as a time-dependent process that unfolds as the network grows. Similarly, homophily, the tendency of similar nodes to connect, can be examined as a force that shapes network structure over time, influencing both the formation of new connections and the strengthening or weakening of existing ones.

By focusing on these temporal aspects, researchers can uncover the underlying mechanisms that drive network growth and evolution. This approach not only provides insights into how networks reach their current state but also offers predictive power for understanding future network configurations. As we delve deeper into the literature on network evolution, we will explore how various studies have tackled these temporal dynamics, the methodologies they've employed, and the insights they've gained into the fundamental processes shaping network structures across diverse domains.

2.1 Triadic closure

Numerous models, mechanisms, and measures have been proposed to describe network growth from a link formation perspective. Triadic closure, in particular, has emerged as a significant mechanism in this context [43]. The fundamental assumption underlying triadic closure is that individuals sharing a common friend are more likely to become friends themselves in the future [28]. Despite being recognized as a fundamental mechanism in the formation of dense groups and communities in social networks [44], the properties and laws governing triadic closure remain underexplored at a large scale, mainly due to the limited availability of temporal-annotated datasets capturing the growth of large social networks.

In analyzing triadic closure from a network structure perspective, a common approach involves investigating subgraph-level features, particularly by identifying and examining the network at the level of 3-node subgraphs. In directed networks, the triadic structure becomes more intricate, as more 3-node directed subgraphs are possible. They are commonly referred to as *triangles* [45] in undirected networks and *triads* [45] in directed networks. Existing methods have the objective of performing a *categorization* [46], i.e., separating the triads based on their structure. Bagatelj *et al.* [47] introduced a subquadratic algorithm suitable for large and sparse networks, classifying triads into different isomorphism classes based on network structure with complexity of $O(m)$, where m represents the number of links. This algorithm is the de facto standard, as successive works worked on its optimization. For example, Chin *et al.* [48] worked on distributed versions of this algorithm, focusing on designing data structures for distributed computing. These improvements were leveraged by Parimalarangan *et al.* [49] that combined them with more recent efficient algorithms designed for undirected networks. Santoso *et al.* [50] focused on how to use these previous advancements in a single machine, focusing on methodologies for data compression, preprocessing, and parallelization. Finally, other works, such as Buriol *et al.* [51] and Pavan *et al.* [52], have focused on a slightly different setting, where the exact and the approximate counting of triangles is conducted in a streaming fashion.

The structure of a network can be characterized by the distribution of these subgraphs, as demonstrated by Milo *et al.* [53], a useful approach even in the context of online social networks [54]. Alrhmoun *et al.* [55], for instance, used the distribution of 3 nodes motifs to analyze the strategies of bots in a social network. While static structure provides insights into triadic closure effects, leveraging temporal information is crucial for a more comprehensive analysis. From a triadic closure perspective, this involves focusing on *temporal* *triads*, or 3-node temporal subgraphs. Identifying these temporal subgraphs is less straightforward than in the static case. For example, Kovanen *et al.* [56] consider a subset of temporal subgraphs where the time difference between consecutive events is less than an input interval Δc, and all events are consecutive. Paranjape *et al.* [57] extended this by removing the constraint on consecutive events, allowing the study of subgraphs occurring in short bursts and introducing a time window Δw to bound the time difference between the last and first events in a subgraph. Overall, different models exist in the literature [58], emphasizing that the distribution of temporal subgraphs can be employed for network comparison and generation, similar to the static scenario [59, 60, 61]. For instance, Hulovatyy *at el.* [58] proposed a vectorial representation of nodes using dynamic graphlets. This method adopts a common approach, i.e. decomposing networks into small temporal subgraphs (network motifs or graphlets) [53, 56, 57] to characterize network evolution. In this context, Faisal and Milenkovic [62] adopt a graphlet-based approach for representing the different snapshots of a temporal network. The idea of using graphlets and their orbits has been further improved in Aparicio *at el.* [63], where the graphlet-based representation of each snapshot is used to identify the orbit transitions across two consecutive snapshots to highlight inter-snapshot correlation. When limited to 3-node subgraphs, the idea of graphlet-orbit transitions is close to the intuition of computing the probability of moving from an open temporal triad to a closed one, which is at the basis of our triadic closure rules and the seminal work in Doroud *et al.* [64]. However, it is worth pointing out a fundamental difference between our work and [64, 63]: the latter are strongly dependent on the window size of the snapshots since transitions are computed among consecutive snapshots. On the contrary, we adopt a different perspective oriented towards a continuous-time approach, so we do not impose discretization of the growing network into snapshots and avoid the issue of selecting a proper window size.

To enrich the information provided by these approaches, previous work introduced some temporal metrics to quantify triadic closure. These metrics can capture and quantify the presence and dynamics of the triadic closure mechanism in a network, and they can also be used to compare different networks. Among them, we have the concept of *triadic closure delay*, a temporal metric to measure the speed of undirected closed triads [65]. In the undirected case, a subgraph transitions from an open triad (connected with two links) to a closed triad (triangle) when the last pair connects. The triadic closure delay measures the difference in time between the opening time (i.e., the time at which the second link was formed, leading to an open triad) and the closing time (when the third link is formed). In the directed case, bidirectional links complicate the definition of opening and closing times, as the insertion of a link, does not imply a triadic closure. therefore, in a successive work [66], different definitions of opening and closing times were proposed, allowing the definition of a directed triadic closure delay.

2.2 Homophily

The study of homophily in network evolution has provided valuable insights into how similarity influences the formation and dynamics of connections over time. For instance, the work by Nguyen *et al.* [67] examined contact networks using WiFi log data from a university campus, revealing that gender homophily increases over the course of a semester, while homophily based on income and grades remains relatively stable. This temporal perspective on homophily highlights the dynamic nature of social preferences and their impact on network structure. Weber *et al.* [68] advanced our understanding by investigating the effects of choice and induced homophily in the formation of learning groups. Their findings demonstrated that gender homophily is particularly pronounced among male students, shedding light on the complex interplay between individual choices and structural factors in shaping network evolution. Expanding on the temporal aspect, Kovanen *et al.* [69] introduced the concept of "temporal homophily" through their analysis of mobile phone record networks. This novel approach considers not just the existence of homophily at a given moment, but its persistence over time, providing a more nuanced view of how similarity-based connections endure in evolving networks. The practical implications of homophily in network evolution were underscored by Kister *et al.* [70], who examined its impact on node2vec embeddings for disease spread prediction. Their research revealed that homophily serves as a valuable indicator of infection status, suggesting that incorporating homophily-based features could enhance the accuracy of epidemic forecasts, particularly in scenarios where only partial information about contacts is available.

These studies collectively demonstrate the multifaceted role of homophily in network evolution, from its varying influence across different attributes to its temporal dynamics and potential applications in predictive modeling. By examining homophily through a temporal lens, researchers have uncovered its nuanced effects on network growth and structure, offering valuable insights for understanding and predicting the behavior of complex social systems.

2.3 Network evolution through community

Network evolution can also be studied by a mesoscopic perspective, through the lens of community dynamics. This approach focuses on the formation, growth, merging, splitting, and dissolution of communities within networks, providing insights into the collective behavior of nodes and the emergence of higher-order structures. A seminal work in this area is by Palla *et al.* [71], who introduced a method to track the evolution of overlapping communities in social networks. Their study of co-authorship and mobile phone call networks revealed that larger communities tend to be more stable over time, while smaller ones are more dynamic, often fragmenting or merging. Building on this, Mucha *et al.* [72] developed a framework for studying community structure in multilayer networks, allowing for the analysis of community evolution across different timescales. This approach enabled researchers to identify persistent communities and track how they evolve, merge, or split over time. Greene *et al.* [73] proposed a method for detecting "evolutionary events" in dynamic networks, such as the birth, death, merging, and splitting of communities. Their work on a large-scale mobile phone network demonstrated how these events can be used to characterize the overall evolution of the network structure. Tantipathananandh *et al.* [74] introduced the concept of "community matching" to study the persistence and change of communities over time. Their framework allowed for the identification of stable cores within evolving communities, providing insights into the long-term dynamics of social networks. More recently, Rossetti *et al.* [75] developed a framework called TILES for tracking the evolution of overlapping communities in streaming scenarios. This approach enables real-time analysis of community dynamics, offering potential applications in areas such as online social network analysis and real-time event detection. These studies collectively demonstrate that examining network evolution through community dynamics provides a rich understanding of how local interactions aggregate to form larger structures, and how these structures evolve over time. This perspective bridges the gap between micro-level node interactions and macro-level network properties, offering valuable insights into the complex processes driving network evolution.

Chapter 3

Framework for temporal networks' modeling

Here, we propose a general framework to model temporal graphs depending on the data used. Note that we are going to describe the framework considering directed temporal interactions without additional labels, but it can also be adapted to the labeled or undirected case.

3.1 List of timestamped interactions

Any temporal data that can be modeled into a graph usually comes as a list TL of timestamped transactions/interactions, as shown in the example of Figure 3.1. The timestamps can have different granularities, starting from milliseconds to weeks, months, and so on. However, the specificity of the timestamp should depend on the case study we are working on. For instance, when we are dealing with a citation network, a second granularity is too precise because citations occur slowly in time. On the other hand, if we consider a bitcoin transaction network, a daily granularity could be too general, because a lot of transactions happen on the same day (for instance on Ethereum in the year 2022, the blockchain recorded more than 800k transactions per day). Since we need different levels of time aggregation, we first define the interval time windows set $I =< [s,e)_i, i = 1, 2, \dots T >$. So, I is a collection of time intervals that offers the freedom to aggregate time in different ways. For instance, in Figure 3.1 we choose to aggregate with an hourly granularity, so the first time window is from 8:00:00 to 9:00:00, the second time window is from 9:00:00 to 10:00:00, and so on. Using a time window of the form $[k, k + \delta)$ as in the example, we can obtain a regular time aggregation.

3.2 Interval model

Once we have defined the time windows in I, we are able to build the first step of temporal graph modeling. The *interval model* shapes the transactions/interactions in TL into a sequence of static graphs $\mathcal{G} = \{G_1, G_2, \dots G_T\}$. Each G_k in $\mathcal{G}$ is described by the tuple (V_k, E_k) that corresponds to the nodes and edges sets respectively. Specifically, $E_k = \{(u,v)|(u,v,t) \in TL, t \in [s,e)_k\}$, and $V_i = \{u|(u,v) \in E_i \vee (v,u) \in E_i\}$. So, each G_k includes all edges whose timestamp is included in the k-th time window. The nodes are the ones that are incident to the edges in E_k. Figure 3.1 shows the graphic example of the transactions listed in TL modeled as a sequence of static graphs $\mathcal{G}$ (the numbers on the edges do not imply that edges are labeled with the aggregated timestamp, they are present to highlight the single time window). The interval model alone can be enough for modeling the temporal information in cases where we want to isolate the interactions that happened in a single time window. For instance, in financial networks, we can use the interval model to find anomalous time windows. However, there are cases where we need an aggregated view of the temporal evolution.

3.3 Projections

From the sequence of graphs in the initial interval model, we can derive different projections, i.e. single graphs that can describe the evolution in a period. We identify three types of projections -growing, snapshot and evolving- that can offer a more compact or more informative representation of the temporal data. For every projection, the set of nodes V^P always corresponds to the nodes adjacent to the edges considered. The distinctive elements of each projection are the definition of edges set E^P and the labeling function f^P that assigns a temporal label to each edge.

Growing Projection

The *growing projection* (GP) is the most compact representation, because it flattens all the edges in a specific period of time, eventually selecting just one edge for each pair of nodes. Formally, the growing projection of $\mathcal{G}$, from timestamp l to timestamp m, is defined as $E^{GP}_{[l,m)} = \{(u,v)|(u,v) \in \bigcup_{k=l}^{m-1} E_k\}$. It is clear that taking the union of all edges (u,v) in the specific period, we keep just one edge from a node u to a node v. The labeling function $f^{GP}_{[l,m)}$ assigns a timestamp to each edge, basically selecting one of the multiple edges between that couple. There are different ways to select which one of the edges to keep, the most common ones use the minimum or maximum timestamp, so to keep the most recent or most old link. The choice of the minimum or maximum timestamp depends on the case study. For instance, in a graph where a link (u, v) in G_k means that user u followed v in time window k, it could be better to keep the oldest link. So we keep the first time they made a connection, and not the time of the eventual re-follow. If we use the first-interaction option, the labeling function is formally defined as follows:

$$f^{GP}_{[l,m)} : E^{GP}_{[l,m)} \to [1, 2, \ldots, T]$$

$$(u, v) \to min\{k|(u, v) \in E_k\}$$

Figure 3.1 shows the graphic representation of $G^{GP}_{[1,4)}$: an example of multiple edges is from node b to node c. In the growing projection the timestamp of the edge (b, c) is 1 because is $min(\{1, 2\})$. This projection is suitable for cases where the number of multi-edges is low or it does not influence how the system evolves. For instance, it is a good choice for modeling social or trust networks. On the other hand, it could be a lossy representation if we want to study the transactions between users, since we would lose information about the strength of exchanges.

Snapshot Projection

Another representation is the *snapshot projection* (SP), which is the only multi-edge graph. It maintains the concept of flattening from the growing projection, but instead of choosing one edge for each couple (u, v), it keeps all of them with the relative timestamp. So, in this case the edge set $E^{SP}_{[l,m)}$ is a multiset that include all edges recorded from E_l to E_{m-1} (as formally defined in the footnote [1]). The labeling function $f^{SP}_{[l,m)}$ here simply assigns to each edge (u, v) in G_k the timestamp k. The snapshot projection is clearly less compact than the growing projection, but it could be the right choice in cases where it is important to keep track of all multiple interactions between the same couple of nodes, for instance, to model economical transactions.

Evolving Projection

The last representation of our framework is the *evolving projection* (EP). This option combines the previous ones because it is not a multi-edge graph but it contains the most complete information about the entire evolution of each edge. Formally, the edge set is defined as the one of the growing projection, so $E^{EP}_{[l,m)} = \{(u, v)|(u, v) \in \bigcup_{k=l}^{m-1} E_k\}$. The labeling function assigns a string of characters of length equal to the duration of the period $[l, m)$ -so the length is $m - l$. The number (0 or 1) in position i tells if the edge was present in G_i.

[1] $E^{SP}_{[l,m)} = \langle \chi, \mu_\chi \rangle$ with $\chi = \bigcup_{k=l}^{m-1} E_k$ and $\mu_\chi : \chi \to N^+, \mu_\chi(u, v) = |\{k|(u, v) \in E_k\}|$

Formally

$$f^{GP}_{[l,m)}(u,v) = \{s_n\}$$
$$\text{where } s : [0, m-l) \to \{0,1\} \text{ with } s(n) = \mathbb{I}[(u,v) \in E_n]$$

Let us consider the example of edge (b,c) in Figure 3.1: the label is 110 because the edge was present in G_1 and G_2 but not in G_3 (there is a 0 in the $3rd$ position) of the interval model sequence $\mathcal{G}$. The evolving projection is more informative with respect to the snapshot projection because it can also include relabeling (if instead of 1s or 0s the character reflects the labels - eventually encoded) or edge deletion. This representation (with its adaptations) is used in several graph evolution rules algorithms because it allows the representation of the entire evolution in a single labeled graph.

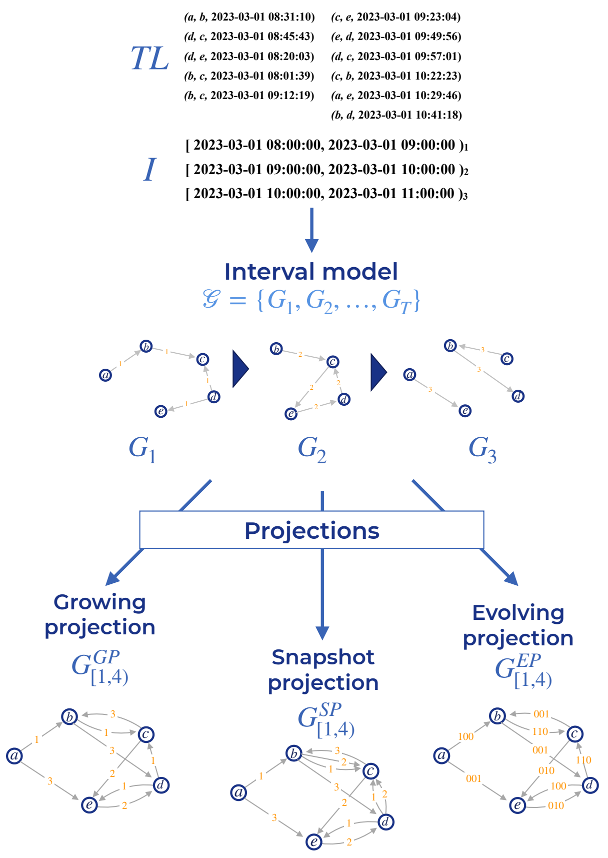

TL
(a, b, 2023-03-01 08:31:10)
(d, c, 2023-03-01 08:45:43)
(d, e, 2023-03-01 08:20:03)
(b, c, 2023-03-01 08:01:39)
(b, c, 2023-03-01 09:12:19)
(c, e, 2023-03-01 09:23:04)
(e, d, 2023-03-01 09:49:56)
(d, c, 2023-03-01 09:57:01)
(c, b, 2023-03-01 10:22:23)
(a, e, 2023-03-01 10:29:46)
(b, d, 2023-03-01 10:41:18)
I
[2023-03-01 08:00:00, 2023-03-01 09:00:00)1
[2023-03-01 09:00:00, 2023-03-01 10:00:00)2
[2023-03-01 10:00:00, 2023-03-01 11:00:00)3
Interval model
$\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$
G_1
G_2
G_3
Projections
Growing projection
$G^{GP}_{[1,4)}$
Snapshot projection
$G^{SP}_{[1,4)}$
Evolving projection
$G^{EP}_{[1,4)}$

Figure 3.1: Framework for temporal networks. *The schema illustrates the process of modeling temporal networks to study their evolution. It begins with two fundamental inputs: a list of interactions* (T) *and a list of intervals* (I). *These inputs are used to construct the interval model, represented as a sequence of static graphs. The interval model then serves as the starting point for three distinct temporal network projections, each offering a unique perspective on the network's evolution over time.*

Chapter 4

Graph evolution rules

4.1 Introduction

This chapter introduces the concept of graph evolution rules (GERs) as a powerful tool for understanding network evolution. Graph evolution rules are inspired by association rules in the data mining field, so we start by describing the latter, providing a familiar context for readers. The chapter then explores various subgraph counting methods, including frequent subgraph mining, motifs, and graphlets, contrasting them with GERs to highlight the unique advantages of the latter. We delve into the fundamental concepts underlying GERs, such as rule composition, support measures, and anti-monotonicity, and explain the key mining algorithm: gSpan. The chapter concludes with a comprehensive review of seminal works in the field, setting the stage for our novel contributions to the domain of graph evolution analysis.

4.2 Preliminaries

Graph evolution rules are related to two main research topics: association rules and subgraph counting respectively in data mining and graph mining fields. Let us first explain the association rules concept and its temporal extension. The next subsection will provide details on the different approaches to subgraph counting, and their extension for temporal networks.

4.2.1 Association rules

Association rules are a fundamental concept in data mining that aim to discover interesting relationships and associations between items in large datasets.

They were introduced in 1993 by [76] as an organization aid in the supermarkets industry field and then quickly extended to other domains. These rules are typically represented as "if-then" statements, where certain items or itemsets are found to co-occur together with a certain frequency, also called *support*. Initially, they were used in market basket analysis, i.e. a technique that study customer purchase patterns (through their shopping baskets data) to identify associations and relationships between items frequently bought together. An example of an association rule could be

$$\{Bread, Eggs\} \rightarrow \{Milk\}$$

The rule suggests that if someone buys *bread* and *eggs*, he/she is probably buying also *milk*. Note that the association rules express co-occurrence and not causality. Such associations can be used to drive business strategies like product placement, cross-selling, targeted promotions, and inventory management. From a set of items, one can obtain an exponentially increasing number of rules due to all the elements' combinations. However, association rules algorithms return just the most important ones, typically selected through a support and confidence measure. Given a rule $r : X \rightarrow Y$ with X being a set of items (or even a single item), and $Y \notin X$, and given a set of transactions D the support of r id defined as

$$s(r) = P(X \cap Y) = \frac{\sum_{t_i \in D}[X \in t_i \& Y \in t_i]}{|D|}$$

So, it represents the ratio of the ratio between the number of transactions containing both X and Y over the total number of transactions. On the other hand, the confidence of a rule indicates its strength, and it is computed as the transactions containing both X and Y over the transaction containing X, formally defined as

$$c(r) = P(Y|X) = \frac{\sum_{t_i \in D}[X \in t_i \& Y \in t_i]}{\sum_{t_i \in D}[X \in t_i]}$$

Temporal association rules. A more advanced model in this context concerns temporal association rules, introduced by Ale *et. al.* [77], where timestamps are assigned to transactions. Specifically, given a set of items $R = \{A_1, \dots A_n\}$ and a transaction database d composed by subsets of R called transactions s, each of them having assigned a timestamp t_s, the authors defined as $V(X, d)$ the cardinality of the set of transactions $s \in d$ where the elements of X (with $X \subseteq R$) are contained. The lifespan $[t, t']$ of the set X is defined as the minimum interval of time shared by all items in X, formally

$$t = max_{t_1}\{[t_1, t_2] = lifespan(A_i), \forall A_i \in X\}$$

$$t' = min_{t_2}\{[t_1, t_2] = lifespan(A_i), \forall A_i \in X\}$$

Consequently, $d_{[t,t']}$ corresponds to the subset of transactions $s \in d$ whose $ts \in [t, t']$. Finally, the support is defined as

$$s(X)_{[t,t']} = \frac{V(X, d_{[t,t']})}{|d_{[t,t']}|}$$

So, a temporal association rule $X \to Y[t_1, t_2]$, holds in d if:

- its support $s(X \cap Y)_{[t,t']} \geq \sigma$;
- its confidence $c(X \cap Y)_{[t,t']} = \frac{s(X \cap Y)_{[t,t']}}{s(X)_{[t,t']}} \geq \lambda$;
- it temporal support $t_2 - t_1 \geq \tau$.

In other words, the rule holds with support s, temporal support *tau* and confidence c if, in the time window $[t_1, t_2]$ with a duration equal or greater than τ , both X and Y are contained in the s% of transactions of d, while c% of transactions containing X also contain Y.

4.2.2 Subgraph counting approaches

The subgraph counting problem, i.e. the ability to compute subgraph frequencies [46], is the common factor of several approaches that study a network from a topological point of view. The most popular ones are network motifs, graphlets, and frequent subgraphs. The differences between them concern the mining approach and the criteria to consider them worthy of attention. Let us consider them separately.

Frequent subgraph mining. Frequent subgraph mining (FSM) is the main principle of graph mining. Like data mining in general, graph mining aims to extract statistically significant and useful knowledge from data. The frequent subgraph mining goal is to extract all frequent subgraphs that occur more than a specific threshold, called *support threshold.* According to the survey by Jiang *et al.* [78], there are two formulations of the FSM problem based on input graph modeling. The first one is *graph transaction based FSM* where the input graph is a database of graphs, here the support count of a subgraph corresponds to the number of graphs in the database where the subgraph occurs.

In the second case, where the input is a single graph (*single graph based FSM*) the support computation requires a more specific definition because the exact occurrence counting would require too long runtimes. The algorithms of both formulations are divided into two categories. The first one comprehends all methods with an *Apriori-based approach*, where a Breadth First Search (BFS) strategy is used to explore the subgraph lattice of a database. The second category concerns the *pattern growth based methods*, which consists in adopting a DFS strategy to recursively extend each subgraph. ***Frequent subgraph mining in temporal graphs*** was introduced in 2006 by Borgwardt *et al.* [79]. The idea is to model the temporal network in a union graph, that involves all nodes and edges appearing along time. The label on the edge encodes the evolution by specifying the timestamps when it was present in the graph. For instance, an edge with the label 0110 indicates the edge appeared in the graph in the second timestamp (position 2 of the string), it stayed in the graph for the next two timestamps and then was removed. Another approach [80] to apply frequent temporal subgraph mining is to apply a static version of FSM on each snapshot and then combine the results whether in an aggregated or incremental way.

Motifs. Motifs are mesoscopic structures of a network, larger than single nodes or links, but smaller than the entire network. One of the earliest works introducing the concept is Milo *et al.* [53], which defined motifs as classes of isomorphic induced subgraphs with cardinality higher in the data than in a reference null-model. Since most systems are better described using also temporal information, studying their mesoscopic structure requires analyzing both topology and time. There are several studies concerning ***temporal motifs***, and each one proposes its own temporal motif model with its own constraints that make a subgraph valid in some models but not in others. The main difference concerns the concept of temporal isomorphism: relaxing the constraint that a pattern should respect to be temporally isomorphic to another makes the computation easier. For instance, Kovanen et al. [81] consider as valid only subgraphs that include all consecutive events (temporal edges) that occur one maximum Δ_c timestamps after the previous one. While Paranjape *et al.* [82] consider isomorphic two subgraphs with the same topology and whose edges respect the same temporal ordering while remaining in a time window of Δ_t.

Graphlets. Graphlets are equivalence classes of induced connected subgraphs [46]. With respect to the FSM or the motifs scenarios, here the subgraph definition is more precise because it requires the subgraphs to be induced by any set of a predefined number of nodes. A set of k nodes can only form one class of graphlet because the fact that the subgraph is induced by the set of nodes implies considering all edges between them. The count of graphlets is made by the graphlet-degree distribution, which leverages the concept of orbit. An orbit is an equivalence class over nodes, i.e. a set of nodes that are topologically equivalent. In other words, two nodes are in the same orbit if when switched the subgraph still belongs to the same graphlet. For instance, in a star-shaped subgraph, all leaves belong to the same orbit. The graphlet-degree distribution GDD_G [83] is a feature matrix that in position (i, j) specifies the number of nodes that belong j times to orbit i. In the dynamic context, ***temporal graphlets*** were first studied by Faisal *et al.* [84], which studied static graphlets in each snapshot and then analyze the time series of the resulting GDD_G. A more proper temporal graphlet was defined by Hulovatyy *et al.* [58] as an equivalence class of isomorphic Δ_t-connected temporal subgraphs. The Δ_t parameter controls the maximum time that could pass from a temporal event (edge) to the next one.

In summary, graphlets, motifs, and frequent subgraph counting are all techniques used in graph analysis to identify patterns within graphs. Figure 4.1 offers a visual explanation of the differences. While frequent subgraphs and motifs search for all types of patterns (often limited by a maximum number of edges/nodes) that are actually present in the graph (hence the input graph $\mathcal{G}$ is the only input), graphlet counting starts from a predefined set of patterns to search for. The prefined set is composed by all k-graphlets, i.e. (connected) subgraphs induced by a given number of nodes k. The difference between motif and frequent subgraph is that in the latter case the frequency (or support) s of the subgraph is the only way to evaluate the importance fo a subgraph. In the motif case, instead, the algorithms evaluate the statistical significance of subgraphs with the introduction of a null model. Consequently, the importance of a subgraph is evaluated through a z-score z.

4.2.3 Common concepts and terminology

In this section, concepts and terminology about graph evolution rules mining are introduced and they will be used throughout the rest of the thesis.

Graph evolution rule composition A graph evolution rule is a frequent temporal pattern that helps analyze the evolution of a dynamic network. Following the association rules concept of the data mining context, they are composed of a precondition and a post-condition. Figure 4.2 shows an example of an open triad that evolves into a closed triangle. Note that the precondition and post-condition are also called body and head respectively(in [29, 42]) or antecedent and consequent (in [34]). The rules in the different algorithms

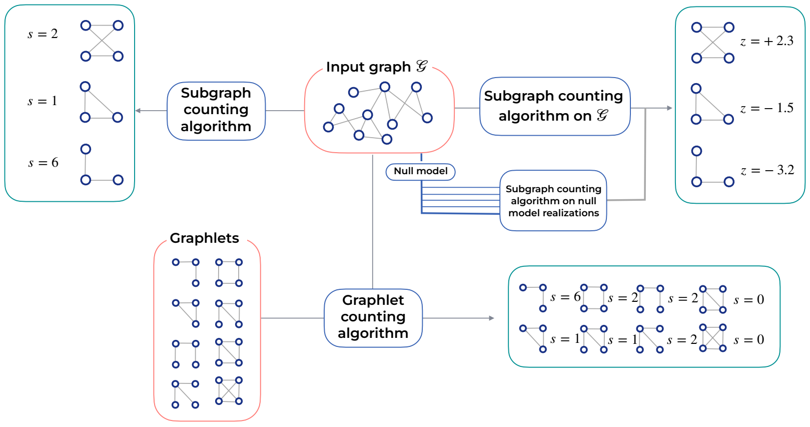

Input graph 𝒢
Subgraph counting algorithm
s = 2
s = 1
s = 6
Subgraph counting algorithm on 𝒢
z = + 2.3
z = − 1.5
z = − 3.2
Null model
Subgraph counting algorithm on null model realizations
Graphlets
Graphlet counting algorithm
s = 6
s = 2
s = 2
s = 0
s = 1
s = 1
s = 2
s = 0

Figure 4.1: *Differences between frequent subgraphs, motifs and graphlets.*

change shape because of the different constraints they have. However, the common concept is their interpretation: a subgraph matching the precondition will probably evolve matching the postcondition.

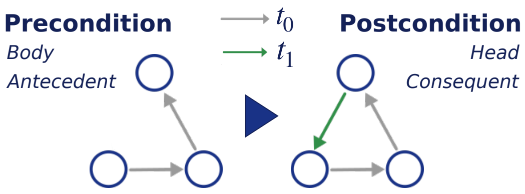


Figure 4.2: *Example of graph evolution rule. On the left graph with two links - grey arrows - created at time t_0, while on the right the head of the rule where the graph on the left evolves by adding the new link - green arrow - at the successive timestamp t_1.*

Minimum image based support The graph evolution rules mining algorithms return the set of frequent rules, so it is necessary to define a support measure that can establish a rule's frequency. Methods often choose the min-

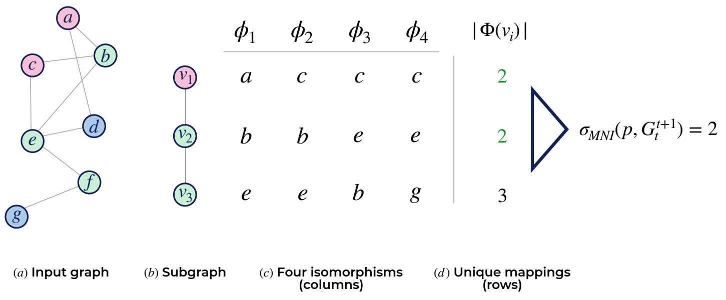


Figure 4.3: MIB support. *Example of how the Minimum-Image Based Support of subgraph in (b) is counted in in the input graph in (a).*

imum image based support (MIB). Given a graph G, a subgraph p and the set $\Sigma(S) = \{\phi_1, \phi_2, \ldots, \phi_m\}$ of all isomorphisms (or embeddings) of p in G, the MIB support [85] corresponds to the number of unique nodes of the input graph G that each node of the subgraph $p = (V_p, E_p)$ is mapped to. So, formally the MIB support is defined as follows:

$$\sigma_{MIB} = min_{v \in V_p}\{|\Phi(v)|\},$$

where $\Phi(v)$ represent the set of unique mappings for each $v \in V_p$, defined as:

$$\Phi(v) = \bigcup_{i=1}^{|\Sigma(S)|} \phi_i(v)$$

Let us consider the toy example of Figure 4.3. Given the input graph in 4.3a, in order to obtain the MIB support of the subgraph in 4.3b it is necessary to detect its graph isomorphisms (or embeddings), shown in each column of the table in Figure 4.3c. The next step is the computation of the cardinality of $\Phi(v_i)$ for each v_i in p, shown in Figure 4.3d. So, the MIB support of the pattern p in G is the minimum value of $\Phi(v_i)$, that is equal to 2. The minimum image based support is a frequent choice because it respects the anti-monotonicity property while being not as computationally expensive as other support measures.

Anti-monotonicity property A support metric σ for a graph G is defined as *anti-monotonic* if, for all subgraph S, P, with S being a P's subgraph, it must hold that $\sigma(S) \leq \sigma(P)$. Basically, the support of a pattern's subgraph must be higher than the support of the pattern itself.

gSpan algorithm Every graph evolution rule mining algorithm is based on a (modified) version of a state-of-the-art frequent pattern mining method, called gSpan [30]. It is an efficient algorithm to mine frequent subgraphs (patterns), that takes as input a set (or a database) of undirected static graphs and that returns connected frequent subgraphs. The idea is to leverage the minimum DFS code concept to make the graph isomorphism computation easier. Let us use an example to explain the DFS code's idea. Given the graph G of Figure 4.4a, its DFS code is the series of 5-tuple listed in Figure 4.4b. Basically, it consists in encoding the edges visited with a Depth First Search on the graph with 5-tuple of the following format $(i, j, l_i, l_{(i,j)}, l_j)$ where i, j are the index of the nodes in visiting order, and $l_i, l_j, l_{(i,j)}$ are the labels of the two nodes and the edges respectively. There exist multiple DFS codes for each graph/pattern, however, the codes can be lexicographically ordered so that the first one in the sorted list will be the minimum DFS code. The essential property of min DFS codes is that two graphs with the same min DFS code are isomorphic, as stated in a Theorem in [30]. So, the lexicographical order allows sorting all DFS codes of the same graph/pattern to find the minimum one, but it also enables sorting all min DFS codes. If codes of different patterns can be sorted, it is possible to build a *DFS tree*. In the DFS tree, every subgraph is a node, and the children of a node are all subgraphs obtained by a single-edge extension on the rightmost path. This extension must consist in adding an edge from the rightmost vertex and it can be a *backward edge* if the edge point to another vertex in the subgraph, or a *forward edge* if it links the rightmost vertex to a new vertex. Essentially, gSpan algorithm starts with single-edge patterns, ordered lexicographically, then for each pattern it performs an extension of the rightmost path recursively, building the DFS tree. At every extension, gSpan checks if the pattern that is expanding the DFS tree is minimum and if it is frequent. Specifically, a pattern is not minimum if there exists another isomorphic pattern whose DFS code is the minimum, and it is not frequent if its support is not greater than the fixed threshold . When the pattern is either not frequent or minimum, the tree is pruned on that node, meaning that the recursive extension stops and the algorithm returns the previous levels. Graph evolution rules methods typically set a parameter max_edge that indicates the maximum number of edges that the post-condition of every GER must have. This constraint reduces the gSpan running time because it does not have to find all frequent patterns but every recursive extension can stop once the level of the DFS tree corresponding to the max_edge has been reached.

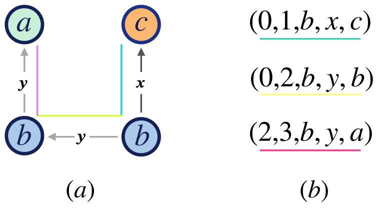


Figure 4.4: DFS Code Representation of a Graph*. The graph in (a) can be represented with the DFS code in (b). The DFS code is a series of 5-tuples, each of one representing an edge (visited in DFS). Each 5-tuple indicates the nodes indexes, their labels and the label of the edge. Edges in (a) are highlighted with a colored line that matches the color of the underline of each edge in (b)*

4.3 GERM

Graph evolution rule mining (GERM) is the method developed by Berlingerio *et. al.* [29] in 2009. The idea is to extract frequent patterns, filter them using both a support and confidence measure, and finally extract graph evolution rules from the remaining frequent patterns, eventually rejecting subgraphs that cannot be transformed. In the following, we are going to provide details about the input graph format, the support and confidence measure used, and the rules extraction method from a temporal pattern.

4.3.1 Graph representation

GERM algorithm uses a modified version of gSpan method to extract frequent temporal patterns, that are filtered and subsequently transformed into graph evolution rules. GERM algorithm presents both a conceptual and actual graph representation. Following the taxonomy described in Chapter 3, the conceptual representation consists in a sequence of growing projections

$$\mathcal{G}^{GP} = \{G_{[}1,2)^{GP}, G_{[}1,3)^{GP}, \ldots, G_{[}1,T)^{GP}\}$$

On the other hand, the compact representation is just the last graph of the previous sequence, which includes all temporal snapshots. Formally, a time-evolving graph (conceptual representation) is represented by a sequence of undirected graphs $G_1, \ldots G_T$, where $G_t = (V_t, E_t)$ and $V_1 \subseteq V_2 \subseteq \ldots V_T$ and $E_1 \subseteq E_2 \subseteq \ldots E_T$. The compact graph of $G_1, \ldots G_T$, is a single graph

$G = (V, E, t, \lambda)$ with $V = \bigcup_{t=1}^{T} V_t$, $E = \bigcup_{t=1}^{T} E_t$, λ a labeling function and t a function that assigns a timestamp to each edge e in the following way:

$$t(e) = argmin_j\{E_j | e \in E_j\}$$

The FSM algorithm can find all temporal frequent patterns on the compact graph, however, a key idea is to reduce the number of redundant patterns to consider. Let us consider the example in Figure 4.5, patterns in (a) and

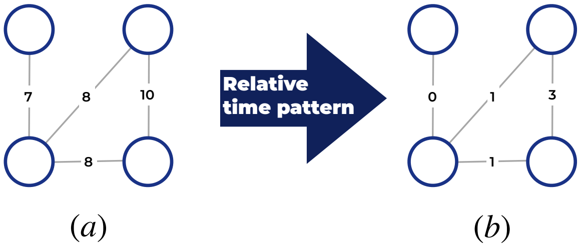


Figure 4.5: Relative-time pattern. *Example of the conversion of a temporal subgraph to a relative-time pattern. Timestamps are adjusted by subtracting the minimum timestamp value, effectively shifting the first timestamp to zero. For example, an edge originally timestamped at* 10 *is now labeled with* 3*, as the new reference point is* 0 *instead of the original minimum of* 7*. This transformation preserves the temporal sequence and intervals while providing a standardized representation*

(b) will be considered different, even if the only difference between them is a time constant $\Delta = 7$. The idea of GERM is to consider *relative-time patterns*, aggregating together in a single equivalence class all subgraphs that share the same topological structure and whose timestamp differ by the same Δ over all the edges. It translates into considering just one pattern per equivalence class, i.e. the one where the minimum timestamp is zero, so in the example, GERM would consider just the one in Figure 4.5b. So, GERM method forces the FSM algorithm (gSpan) to consider just relative-time patterns.

4.3.2 Support and confidence measure

Since GERM is based on a frequent subgraph mining method, it is necessary to specify how they define a pattern to be *frequent*. Berlingerio *et. al.* use the *minimum image based* (MIB) support, because it is a good approximation with respect to the real occurrence number. Moreover, it respects the *anti-monotonicity property*, and it is computationally easier to compute with respect to considering the occurrence number. While the support measure can give insights into how frequent is for a pattern to happen, the probability that the evolutionary steps occur in a specific order is given by the confidence measure. The confidence is computed as the ratio of the support of the head over the support of the body. In this way, it returns the portion of times that the head evolves into a subgraph that matches the body. Using the MIB support in the confidence computation guarantees that the result is a value between 0 and 1, thanks to the anti-monotonicity property.

4.3.3 Rules extraction

A key property of GERM rules is that for each head there exists a single body, this is essential to facilitate the computation of support, considering it equal to the support of its head. The idea behind the transformation from temporal pattern to graph evolution rules is that the body is equal to the head without the last evolution. Formally, given a pattern head $H = (V_h, E_h)$, the body is the subgraph $B = (V_b, E_b)$, where $E_b = \{e \in E_h | t(e) < max_{e^* in E_h}(t(e^*))\}$ and $V_b = \{v \in V_h | deg(v, E_b) > 0\}$. $deg(v, E_b)$ stands for the degree of node v when considering just the edges in the body (E_b). Essentially, given a head, the body is obtained by removing the edges with the maximum timestamp and the nodes that with the latter removal would remain isolated. If the resulting body is connected, then the temporal pattern can be transformed into a graph evolution rule. Otherwise, i.e. if the body is not connected, the pattern cannot be considered as a GER, because the MIB support cannot be computed for a disconnected subgraph. Figure 4.6 shows an example of a temporal pattern (shown in 4.6a) that is transformed into a GER (shown in 4.6b). While Figure 4.7 proposes an example of a temporal pattern (4.7a) that cannot be transformed into a rule, because it would produce a disconnected body (4.7b).

4.4 LFR

Leung *et. al.* [31] proposed a specific type of graph evolution rule, named Link Formation Rule (LFR). The focus is on the process that drives single links formation, for this reason, LF rules are more restrictive with respect to the others, but at the same time, the mining time is significantly lower. In this

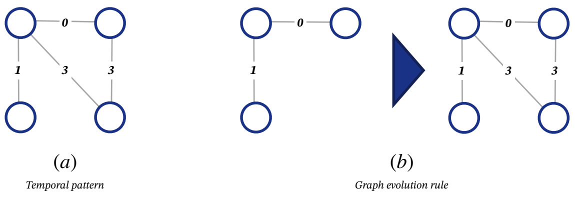


Figure 4.6: Temporal graph into GER. *The temporal pattern in (a) is transformed into a Graph Evolution rule in (b). The post-condition is the entire temporal graph, while the pre-condition consists in the temporal graph without the edge(s) with maximum timestamp (3).*

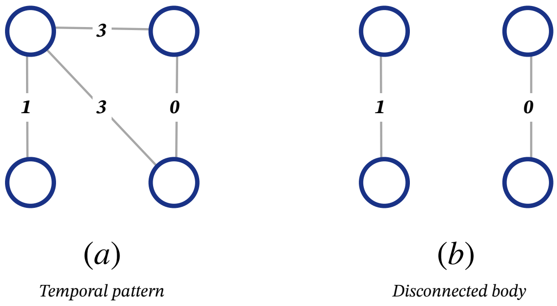


Figure 4.7: *Example of a temporal pattern (a) that cannot be transformed into a GER. The reason is that the body (pre-condition) would be the disconnected graph in (b).*

section, we start by describing the temporal graph representation they used. Then, we are going to explain in detail the format of Link Formation Rules, as well as the pipeline of the method. A detailed description of the support measure utilized by the LFR mining algorithm, and the random-graph part of the pipeline will follow.

4.4.1 Graph representation

LFR method starts from a directed, labeled, time-stamped graph. According to the taxonomy defined in Chapter 3, they modeled the temporal graph as a growing projection $G^{GP}_{[1,T+1)}$, that includes all edges and nodes created through the whole time span.

4.4.2 Characteristics of LF-Rules

Link Formation Patterns. In order to explain LF-rules, it is necessary to first describe the frequent temporal patterns they come from, called LF-patterns. An LF-pattern is 4-tuple $p = (V_p, E_p, L, l_p)$, where V_p, E_p, L and l_p are respectively the set of vertices, the set of edges, the finite set of labels and the function assigning labels to nodes and edges. Specifically, $V_p = \{S_p, I_p\}$ where S_p is a special subset containing only the start node s and the end node e. While I_p contains the so-called intermediate nodes, i.e. the nodes in the pattern that are not s or e (I_p may be empty). Moreover, E_p contains the direct edge (s, e) and all the other edges that eventually involve intermediates nodes. Given $\rho(u, v)$ being the geodesic distance between u and v (length of shortest path), edges in E_p must respect one of the following conditions:

1. If $\mathcal{I}_p = \emptyset$, then $\rho(s, e) = \rho(e, s) = 1$
2. If $\mathcal{I}_p \neq \emptyset$, then $\forall u \in S_p, \forall v \in I_p, min(\rho(u, v), \rho(v, u)) = 1$

Basically, if the pattern contains only the nodes s and e, then there must be also an edge between (e, s), as shown in Figure 4.8a. Otherwise, all intermediate nodes must have a distance equal to one to or from both s and e. Figure 4.8b depicts an example of a pattern that cannot be considered an LF-pattern, because intermediate nodes do not respect condition 2. For instance, node u does not have a minimum distance from or to both s and e of 1.

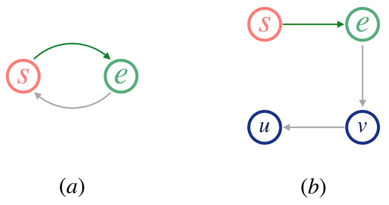


Figure 4.8: Validity of LF-patterns. *The pattern in (a) is a valid LF-pattern because it contains only s and e nodes and there is the link (e, s). On the contrary, the graph in (b) is not valid because node u has distance equal to 2 and 3 with respect to nodes e and s respectively.*

Isomorphism of LF-patterns Two LF-patterns p and q are isomorphic if there exist a bijective function $f : V_p \rightarrow V_q$, such that:

1. $\forall u \in V_p, l_p(u) = l_q(f(u))$

2. $\forall (u,v) \in E_p, (f(u), f(v)) \in E_q with l_p(u,v) = l_q(f(u), f(v))$
3. $f(s_p) = f(s_q)$ and $f(e_p) = f(e_q)$

with l being a labeling function, and s_p, e_p being respectively the source and end node of pattern p. On the other hand, a subgraph isomorphism is a function $sf : V_p \rightarrow W \subset V_q$ with the same conditions of f. If sf exists, p is a sub-pattern of q and q is called super-pattern of p.

Temporal Constraint of LF-patterns. An LF-pattern does not enforce a lot of constraints on the timestamps of the edges included, the only requirement is that the edge between the start node s and the end node e must present the maximum timestamp. The reason lies in the concept of the LF-pattern itself, because the focus is on the formation of the link from a start node to an end node, considering previous connections.

From LF-patterns to LF-rules Given that the only important temporal information is the timestamp on the edge between s and e, the extraction of LF-rules from LF-patterns consists simply in considering the cited link as a post-condition. Formally, a Link Formation Rule $r(p)$ is defined as follow:

$$r(p) = p^a \rightarrow p$$

where p is the LF-pattern, and p^a is a subgraph that maintains the same vertex set of p, while $E_{p^a} = E_p \backslash \{(s,e)\}$. Figure 4.9a shows an example of LF-pattern and Figure 4.9b depicts its translation into LF-rule. It is evident how the edge (s,e) acts as a post-condition of the rule.

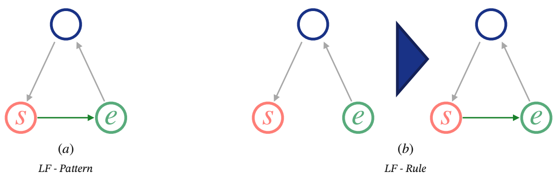


Figure 4.9: From LF-pattern to LF-rule*. The LF-Pattern in (a) is transformed into the LF-Rule in (b) using the LF-pattern ad post-condition, and considering the same pattern without the edge (s,e) as pre-condition.*

4.4.3 Pipeline of the method

The LFR-method follows the pipeline depicted in Figure 4.10. First, the authors adapt the gSpan algorithm to LF-rules requirements. Applying the modified version of gSpan, they extract LF-patterns that are transformed into LF-rules and filtered using both a support and confidence threshold. The lower branch of the schema in Figure 4.10 is needed for supplementary filtering of the resulting rules, taking into consideration a random graph. In fact, this part of the method starts with a specific randomization of the input graph, then they leverage the random graph to compute the expected support of each LF-rule obtained from the actual graph (upper branch of the method). Using the actual support of the LF-rules and the expected support, they were able to compute the surprise value for each rule. This last metric allows obtaining all meaningful rules, that are the object of a rule evaluation. In the next sections

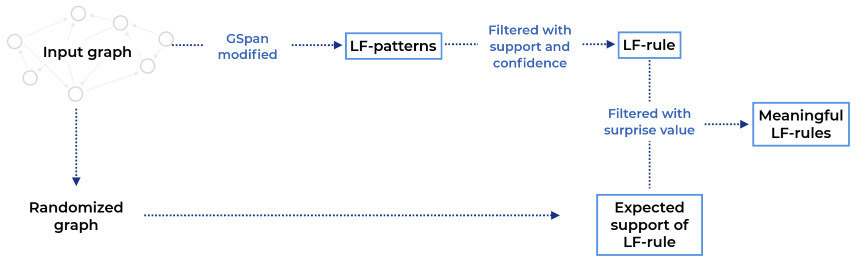

Input graph
GSpan modified
LF-patterns
Filtered with support and confidence
LF-rule
Filtered with surprise value
Meaningful LF-rules
Randomized graph
Expected support of LF-rule

Figure 4.10: *Pipeline of LFR method.*

we are going to provide details about the modifications they performed on the original gSpan algorithm, the support and confidence measures they defined, the randomization technique they relied on, and finally the expected support and the surprise value metric they used to obtain the final output.

4.4.4 gSpan adapted for LF-rules

The LF-rule mining algorithm, called LFR-Miner, is developed by extending the state-of-the-art algorithm for subgraph mining *gSpan*. The extension of the algorithm unfolds in two directions based on the difference between the setting of gSpan and the ones of LFR-Miner: *(i)* with LFR-Miner the graph is directed, and *(ii)* LF-rules are a subset of all the frequent patterns. The original gSpan deals with a database of undirected graphs, while LF-Miner works with a single directed graph. So, the first extension allows considering edge directions. The introduction of ego-based frequency is useful for enumerating the occurrences of patterns. Another important extension concerns the fact that LFR-Miner output is a subset of what gSpan would return because not all frequent patterns (from gSpan) are also LF-rules. Therefore, they leverage gSpan's pattern growth method to prune patterns that are not compatible with LF-rules' constraints.

4.4.5 Ego-based Support and confidence

LFR method proposed a novel support measure for subgraphs that is easier to compute from a computational point of view, with respect to the number of occurrences, while respecting the anti-monotonic property. Before providing a thorough explanation of the support measure, called *ego-based support*, it is necessary to give preliminary definitions.

Ego-based Occurrence This definition is needed to quantify the occurrence of a pattern with respect to a specified focal node. Given a focal node w and a pattern p in a graph G, an ego-based occurrence of p is a subgraph q that respects the following conditions:

1. q is isomorphic to p, with the classical definition of graph isomorphism
2. w corresponds to the start node of q, so $s_q = w$
3. the timestamp of the edge from the start to end node in q denoted as $t(s_q, e_q)$, satisfies:

$$t(s_q, e_q) > max_{(u,v)\in E_q \setminus (s_q,e_q)} t(u,v)$$

The set of ego-based occurrences of p with $s_p = w$ is denoted as Σ^p_w, and if there exists at least one ego-based occurrence ($|\Sigma^p_w| > 0$), then p occurred with respect to w, and w supports p. Figure 4.11 shows a toy example: the LF-pattern in 4.11b occurred with respect to node G, but not with respect to F because the timestamp of the edge (s, e) (that in this case would correspond to (F, E)) is not the highest.

Ego-based Frequency Given a pattern p, and the set of ego-based occurrences Σ^p_w for all nodes $w \in V$, the ego-based frequency of p in G is defined as follows:

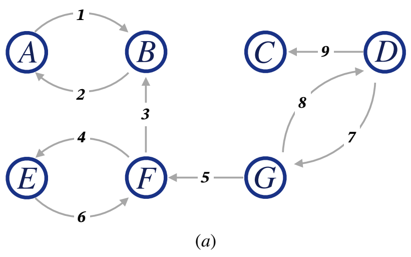
$$freq(p, G) = \sum_{w\in V} \delta(p, w)$$

with

$$\delta(p,w) = \begin{cases} 1 & \text{if } |\Sigma_w^p| > 0 \\ 0 & \text{otherwise} \end{cases}$$

Basically, $freq(p,G)$ counts the number of nodes that serve as focal nodes of p in G. Considering the example of Figure 4.11, the pattern in 4.11b has a frequency equal to 1, because the only node supporting it is node G.

Ego-based Support and Confidence After the definition of ego-based occurrence and ego-based frequency, the concepts of ego-based support and confidence are straightforward. The support of an LF-rule $r(p)$ is defined as $supp(r(p)) = \frac{freq(p,G)}{|V|}$, while the confidence of the same rule is defined as $conf(r(p)) = \frac{freq(p,G)}{freq(p^a,G)}$, with p^a being the post-condition of the LF-rule. If we consider the LF-rule in Figure 4.11b, its support is equal to $\frac{1}{7}$ while the confidence is equal to $\frac{1}{3}$, because p^a occurs with respect to nodes F, G, D.



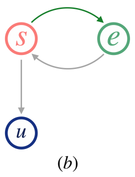


Figure 4.11: *A Graph in (a) and one of its LFR-Rules in (b). The support of (b) is $\frac{1}{7}$ while the confidence is $\frac{1}{3}$.*

4.4.6 Finding meaningful rules

A feature that distinguishes the LFR method from the other graph evolution rules algorithms is the introduction of a null model. This allows assessing whether the relevance of a rule is due to the particular structure of the graph or is a typical evolutionary mechanism in dynamic networks.

Random graph Since LF-patterns focus on the ego-based frequency, they designed a randomization method focused on start and end nodes. Given an input graph G, the random graph G' is obtained by swapping end nodes. Specifically, for each edge (u, v), the method picks another edge randomly (w, z) and switches the end nodes, obtaining the edge $(u, z), (w, v)$. The swapping will be confirmed if the following conditions take place:

- $u \neq z$ and $w \neq v$
- before the swapping, $(u, z) \notin E'$ and $(w, v) \notin E'$

In other words, the swapping will be discarded and tried again with a different randomly chosen edge if it would produce self-loops or duplicated edges. The degree, label, and timestamp distributions of G are preserved in the randomized G'.

Expected support and surprise value On the randomized graph G', they compute the support of each rule extracted from G, and call the measure *expected support*, denoted as $supp(r(p), G')$. The surprise value $sur(r(p))$ is defined as

$$sur(r(p)) = \frac{supp(r(p), G)}{supp(r(p), G')}$$

When the surprise value is higher than one, it reveals that the rule has higher-than-expected support given G. The higher the value, the more relevant the rule is.

4.5 Evomine

This section will outline the details of the algorithm proposed by [35] in 2016, called *EvoMine*. It shares the same two-phase structure of other algorithms such as GERM [29] or LFR [31]: first, it performs a frequent subgraph search, then it processes and filters the resulting frequent pattern according to specific evolutionary constraints and graph evolution rule format. However, the trademark of the EvoMine algorithm lies in the evolutionary constraints and its ability to detect also edge deletion and relabeling in addition to edge insertion. So, in this section, we are going to provide an exhaustive explanation of evolutionary constraints and graph representation, giving details about the EvoMine GER format. Then, we are going to give details about the actual rule mining algorithm, and the support measure proposed.

4.5.1 Graph representation

EvoMine is based on a special temporal graph representation, that the authors call *union graph.* Given a sequence of directed graphs $G_1^T = (G_1, \ldots, G_T)$ with $G_t = (V, E_t, l_t)$, a union graph $\mathcal{G}_U(G_1^T)$ is a flattened representation that encodes all the temporal information in the node and edge labels, that formally corresponds to the evolving projection $G^{EP}_{[1,T+1)}$ as defined in Chapter 3. Let us consider the example depicted in Figure 4.12 to explain the label encoding. Given the graph sequence of Figure 4.12a, its union graph is illustrated in 4.12a. The number of characters of labels corresponds to the length of the graph sequence. Each element in position i of the edge label is either ϵ if the edge in timestamp i is missing, or the value of the edge attribute. For instance, the edge between the two left nodes is $1\epsilon1$, stating that the edge belongs to the edge set of all timestamps except the second one (because the character in the second position of the encoded label is ϵ). In this case, edges have no attributes, so the possible values of each character of the edge encoded label are ϵ or 1. Meanwhile, in the node labels, each element in position i represents the attribute of the node in timestamp i. For example, the top right node's label is 221, indicating that this node changed its attribute in the last timestamp (from yellow to blue in the toy example).

4.5.2 Evolutionary constraints

One of the most distinctive features of EvoMine rules is that they consider evolutions that happen in two consecutive timestamps. This implies that every change highlighted in the postcondition took place in the immediate next timestamp with respect to the timestamp of the precondition's edges. Moreover, Scharwächter *et al.* state three topological constraints that characterize a graph evolution rule detected by the EvoMine method. For a rule $r : (G_{pre}, G_{post})$, where $G_{pre} = (V_{pre}, E_{pre})$ and $G_{post} = (V_{post}, E_{post})$, it is true that $V_{pre} = V_{post}$. In other words, the post-condition and pre-condition must share the node set. The second topological constraints concern the presence of actual evolution: an EvoMine rule is valid if $E_{pre} \neq E_{post}$ or $\ell_{pre} \neq \ell_{post}$, with ℓ_{pre}, ℓ_{post} corresponding to the set of labels of the pre-condition and post-condition respectively. The last constraint involves the concept of *union graph.* Following the definition of union graph in the previous paragraph, the third topological constraint is easy to define: to ensure the locality of the process captured by the evolution rule, the union graph of (G_{pre}, G_{post}) must be connected.

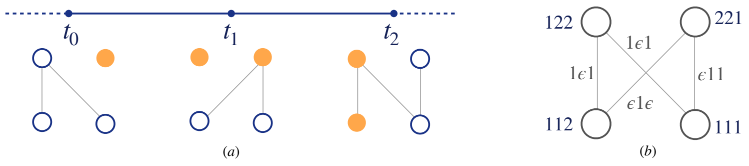


Figure 4.12: Union graph - toy example. *(a) shows a three-timestamps graph sequence, while (b) represents its union graph, where the evolution of edges and nodes is encoded in edges/nodes labels. In the edge labels, the number of characters indicates the length of the graph sequence. As for node label, each element in position i indicates the attribute of the node in timestamp i (in this case 1 or 2, indicating the node being blue or yellow respectively). Meanwhile, in the edge labels, ϵ or 1 indicate whether the edge is missing or not in the timestamp corresponding to the position of the character in the label.*

4.5.3 The method and support measures

As previously stated, EvoMine is developed with a two-phase approach: in the first step they gSpan [30], to extract frequent temporal patterns. The constraints about union graph connectivity and equal node sets are guaranteed by the use of gSpan itself. Then, they filter the temporal pattern, considering just the patterns where a change in edge or nodes is detected (second topological constraint). Note that gSpan returns only *frequent* temporal patterns, so it is necessary to define a support (frequency) measure that is able to select which patterns have to be considered. EvoMine method adapts the gSpan algorithm, using two different support measures: *(i)* an embedding-based support (the well-known minimum image-based support), and *(ii)* an event-based support (novel definition introduced for the first time in Scharwachter's EvoMine paper).

Embedding-based support. In the embedding-based scenario, the idea is to use a support measure that can count the number of embeddings of a rule in the entire graph sequence. Among the alternatives that respect the anti-monotonic property, a prominent example is the minimum image based support σ_{MIB} described in section 4.2.3. In the EvoMine case, the MIB support of a rule $\sigma_{MIB}(p|G_i^{i+1})$ corresponds to the support of the rule's union graph on each intra-snapshot graph G_i^{i+1}. To obtain the final support of a rule in the entire graph sequence $\mathcal{D} = \{G_0^1, G_1^2, \dots G_{T-1}^T\}$, they defined the aggregated MIB support as $\sigma_{agg}(p|\mathcal{D}) = \Sigma_{G_i^{i+1} \in \mathcal{D}} \sigma_{MIB}(p|G_i)$

Event-based support. This novel definition is based on the idea that the support of a rule coincides with the number of change events containing the rule. Let us provide some preliminary concepts, before delineating accurately the event-based support. They consider *event* every change that the rules can capture, so the insertion or deletion of nodes and edges and the relabeling of nodes and edges. When considering the event-based support, the FSM algorithm accepts as input a set of event graphs. Basically, an event graph is a subgraph of the union graph $\mathcal{G}_U(G_t^{t+1})$ induced by the event neighborhood. The event neighborhood has a slightly different definition when considering edge or node events: *(i)* the *node event neighborhood* is a set of nodes that includes the node that causes the event, and its direct neighbors at timestamp t and/or timestamp $t+1$; *(ii)* the *edge event neighborhood* is defined as the union of the node event neighborhoods of the nodes adjacent the edge. Figure 4.13 shows an example: given the union graph (4.13b) obtained from the two-timestamp graph sequence in 4.13a, and the event of adding the edge between nodes u and v, Figure 4.13c represents the corresponding event graph. It is clear that the graph in 4.13c is the subgraph of 4.13b induced by u, v and its neighbors, i.e. the edge event neighborhood of (u, v). In conclusion, the event-based support of a rule corresponds to the number of event graphs from the event graph database where the rule union graph is a subgraph.

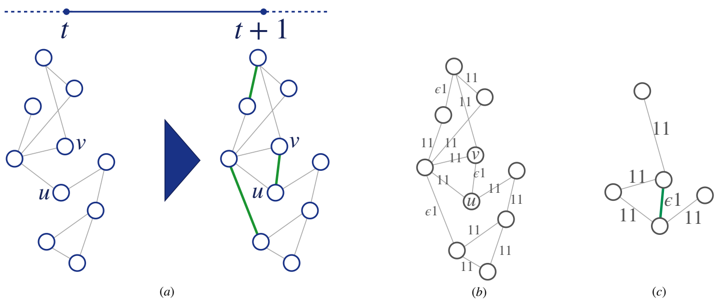


Figure 4.13: Event graph - toy example. *(a) shows two timestamps of a graph sequence, where the creation of the edge between nodes u and v is highlighted, and (b) represents its union graph. (c) is the event graph associated with the creation of (u, v), that is the subgraph of (b) induced by the neighboring nodes of u and v.*

4.6 TP Miner

This section will describe the algorithm published by Miyoshu *et. al.* in 2011, called TP Miner [42]. This method shares some ideas with other graph evolution rules algorithms while having its own characteristics. So, in this section, we first are going to outline the algorithm ideas and graph representation. Afterward, we are going to explain the concepts of frequent time and representative patterns, with all the definitions that come along. Then, we are going to describe how to go from patterns to rules and how the authors analyzed the results using DAGs.

4.6.1 Algortithm ideas

The method is divided into four different steps. The first one consists in applying gSpan, a frequent pattern mining algorithm, like in all the others graph evolution rules methods. In contrast with other algorithms, TP Miner applies a consequent filter on resulting frequent time patterns. In fact, it keeps only patterns that are considered as *representative*, in addition to having a confidence value over a fixed threshold. Graph evolution rules are then extracted from the set of representative time patterns following the same idea of GERM or LFR: the body of a rule is created by discarding all edges with the maximum timestamp, while the head is the representative temporal pattern as it is. The final step consists in building a graph evolution DAG and an abstract graph evolution DAG and analyzing results directly on the directed acyclic graphs. The next sections will provide details about each phase.

4.6.2 Graph representation

TP Miner models the temporal graph as a snapshot projection $G^{SP}_{[1,T+1)]}$, including all edges and nodes and allowing multi edges.

4.6.3 Frequent time patterns

TP Miner's first phase involves the mining of frequent time patterns, using a modified version of the state-of-the-art algorithm gSpan. The algorithm is the same one proposed by Berlingerio *et. al.* in GERM, obtained by adapting the support measure and the canonical representation to a single dynamic graph instead of a sequence of static graphs. As in GERM, the algorithm introduces a confidence measure. However, in TP Miner the definition is more specific with respect to the one in GERM (defined as $\frac{\sigma(head)}{\sigma(body)}$). Formally it is defined as: $conf_G(P_b \rightarrow Ph) = \frac{|O_{P_b}(P_h)|}{|\Phi_G^{P_b}|}$ where $\Phi_G^{P_b}$ denotes the set of occurrences of P_b in G, while $O_{P_b}(P_h)$ stands for the occurrences of P_b that grows into P_h. So the confidence in TP Miner measures the percentage of occurrences of the body pattern that evolves into the head pattern. Finally, a frequent time pattern discovered by gSpan is considered valid if $conf_G(P_b \rightarrow Ph) \geq \tau$.

4.6.4 Representative patterns

The next algorithm step is to filter the time patterns that resulted to be frequent and valid, selecting only the representative ones. In order to explain how the algorithm selects representative patterns, it is necessary to provide some preliminary definitions.
Structure representability The main concept behind the definition of structure representability is that a pattern is *structure-representable* by another pattern if the nodes of one pattern can be mapped into the other ones, and they have similar edges. Formally, given two patterns P and P', P' is *structure representable* by P if $R_s(P', P) \leq \delta$, where:

- $R_s(P', P) = min_{|V_{P'}|=|V_{P''}|, P'' \subseteq^induced P} diff(P', P'')$
- $diff(P', P) = \begin{cases} min_{f \in \mathcal{F}(\mathcal{P},\mathcal{P}')} d(P', P, f) & \text{if } \mathcal{F}(P', P) \neq \emptyset \\ \infty & \text{otherwise} \end{cases}$
- $d(P', P, f) = \sum_{x,y \in V_{P'}} |I^{P'}(x, y) - I^P(f(x), f(y))|$
- $I^P(x, y) = \begin{cases} 1 & \text{if } (x, y) \in E_P \\ 0 & \text{otherwise} \end{cases}$
- $\mathcal{F}(P', P) = \{f | \forall x \in V_{P'}[l_{P'}(x) = l_P(f(x))] \textit{ and } d(P', P, f) \leq \delta\}$ So, one pattern is structure representable by another pattern if there exists a mapping function for a minimum number (δ) of edges.

Smoothing support After the definition of structural representability, it is possible to explain the concept of smoothing support, which essentially extends the concept of occurrence to structurally similar patterns. Formally, given a set of frequent patterns TP, the smoothing support of P_i is defined as follows:

$$ssup_G(P_i) = min_{1 \leq ileq|V_{P_i}}|\{o(i)|o \in S_{P_i}\}$$

where:

- $S_{P_i} = \bigcup_{P_j \in TP, diff(P_i, P_j) \leq \delta} \Phi_G^{P_j}$

- $o(i)$ represents an occurrence of pattern P_i
- $Phi_G^{P_j}$ is the set that contains all occurrences of P_j in G

Preservation of support The concept of smoothing support is used in the definition of support preservation. The support of a pattern P' is said to be preserved if the following inequality holds:

$$P_s(P', P) = \frac{|ssup_g(P) - ssup_G(P')|}{max\{ssup_g(P), ssup_g(')\}} \leq \epsilon$$

Representative patterns Finally, a pattern P is defined as *representative* if it structurally represents and preserves the supports of enough patterns. Formally, a pattern P is representative if the set $C_{TP}^{\delta,\epsilon}(P)$ is large enough, where:

$$C_{TP}^{\delta,\epsilon}(P) = \{P_j \in TP | Rs(P_j, P) \leq \delta, Ps(P_j, P) \leq \epsilon\}$$

4.6.5 From patterns to rules

TP Miners follows the same idea as other algorithms such as GERM for extracting graph evolution rules from frequent (and in this case representative) patterns. As shown in Figure 4.6, the head of the rule (post-condition) is the temporal pattern itself, while the body is obtained by removing all edges with $max_{ts_i \in P} ts_i$.

4.6.6 Graph evolutions DAGs

The real novelty of this algorithm with respect to the others stands in the way they analyze the results. In fact, they do not stop at returning the graph evolution rules and the associated statistics such as the number of GER obtained and the computing time requested. They propose to analyze the GERs from a global point of view, building graph evolutions DAGs that combine the results. The DAG will have the evolutions of rules as edges and the bodies or heads as nodes. Specifically, they propose $DAG_{GER} = (V, E)$ where:

- $E = \{(p_b, p_h) | p_b \to p_h \in GER\}$
- $V = \{p | p \to p_h \in GER \vee p_b \to p \in GER\}$

Afterward, they also propose a more global representation of the DAG_{GER} that merges together branches that differ only for the timestamps in the precondition.
Figure 4.14a shows an example of two temporal patterns that differ only for

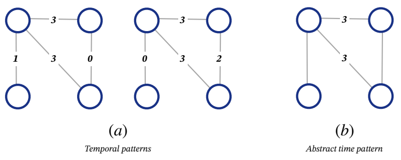


Figure 4.14: Abstract time patterns*. The two patterns in (a) are mapped in the same abstract time pattern in (b), because they differ only for non-maximum timestamps.*

the non-maximum timestamps. Their abstract time version is shown in Figure 4.14b, where the only labels shown are the ones on edges that would appear only in the head of the rule.

Analysis of results The authors analyze the resulting patterns with a network-based approach on the two different DAG_{GER}. Specifically, they rank the patterns (i.e. nodes of $DAGs$) by degree and closeness centrality. Then, they provide observations about the differences between the two orders.

4.7 DGR Miner

This section will describe the details of the algorithm proposed by Vaculik in 2015, called DGR Miner [34]. It is designed for labeled multigraph, both directed and undirected. The common point with other GER algorithms is the use of the gSpan algorithm (even if it is adapted to fit the evolution rules specifics). On the other hand, DGR Miner distinguishes itself from others because it proposes its own graph representations and support measures. In this section, we are going to first describe the dynamic graphs representation proposed by Vaculik, then we will outline the representation and constraint that characterize graph evolution rules in DGR. Next, we are going to illustrate the union graph representation, which is an essential concept that will be used to explain afterward the DGR method.

4.7.1 Dynamic graphs representation

In their study, the authors considered dynamic graphs allowing labeled multi-edges but not taking into account self-loops and identical edges (same vertices, timestamps, and labels). They represent the temporal graph as a sequence of static graphs where $G_i \neq G_{i+1}$. Each static graph G_i is defined as $G_i = (V_{G_i}, E_{G_i}, f_{G_i}, l_{G_i,V}, l_{G_i,E}, t_{G_i,V}, t_{G_i,E})$ where:

- V_{G_i}, E_{G_i} are respectively the set of vertices and edges;
- f_{G_i} is a function that maps an edge e to a pair of vertices (u, v), where $u \neq v$;
- $l_{G_i,V}, l_{G_i,E}$ are respectively the labeling function for vertices and edges;
- $t_{G_i,V}, t_{G_i,E}$ corresponds to the mapping function to assign timestamps to vertices and edges respectively.

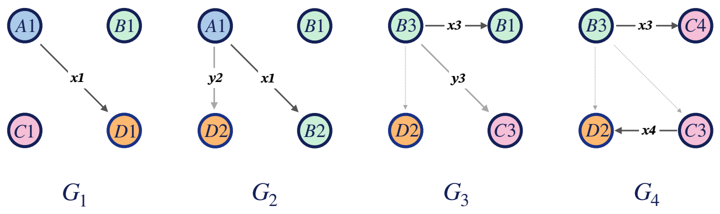


Figure 4.15: DGR representation of a dynamic graph. *Each node and edge is annotated with its label, followed by the timestamp of its most recent change. The representation maintains information about removed edges until they are replaced.*

The representation of the dynamic graph is shown in Figure 4.15. The key element is the explicit annotation of edges and nodes. Specifically, each node or edge is annotated with a label followed by the timestamp of the last snapshot where it changed. For instance, the bottom-right node in G_1 is annotated with $D1$, in G_2 it changed the label into B so the new annotation is $B2$. However, in G_4 the annotation remains $C3$ because in the current timestamp it is not changed. Moreover, the representation keeps the information about the removed edges until a new edge replaces them. According to the taxonomy described in Chapter 3, the dynamic graph is modeled as a sequence of growing projections $\mathcal{G}^{GP} = \{G^{GP}_{[1,2)]}, G^{GP}_{[1,3)]}, , \dots, G^{GP}_{[1,T+1)]}\}$, with a labeling function that combines the labels with the timestamp, assigning the maximum timestamp instead of the more typical configuration that considers the minimum (like in GERM).

4.7.2 How to be a DGR rule

First of all, rules in DGR are represented in a similar way with respect to dynamic graphs, with the exception of timestamps. In order to get general pattern growth rules, it is necessary to use relative timestamps. Specifically, they use timestamp 0 to denote the last change that occurred, while $-t$ indicates an evolution developed t snapshots earlier. In addition to the relative timestamp, a DGR rule must respect some constraints. Specifically, the DGR method discovers rules that represent either an addition or deletion of a subgraph or some specific changes in vertices or edges. Let us describe the three options separately.

Addition of a subgraph Figure 4.16a shows an example of a rule where

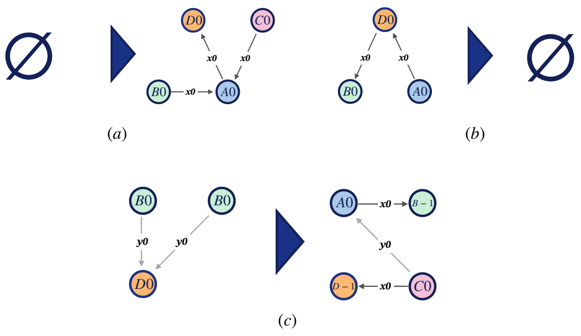


Figure 4.16: Examples of valid DGR rules. *The rule in (a) represents the addition of a subgraph case; while the one in (b) shows the deletetion of a subgraph case. Finally, in (c) the rule corresponds to a subgraph changes.*

the addition of a subgraph occurs. Specifically, $G_a \to G_c$ is a rule because $V_a = \emptyset \bigwedge V_c \neq \emptyset$. Following the representation indications, every edge and vertice in G_c must have a timestamp equal to 0.

Deletion of a subgraph Opposite to the previous case, the deletion of a subgraph occurs when $V_a \neq \emptyset \bigwedge V_c = \emptyset$, as shown in the example of Figure 4.16b.

Subgraph changes The third case covers situations when changes in nodes or edges of the antecedent subgraph occur. Let us consider the graph evolution rule in Figure 4.16c as an example: the first condition the rule must respect is that the antecedent and the consequent subgraphs must share some nodes, unlike the two previous cases. Moreover, all the following conditions must hold:

- *Either all nodes or all edges in G_c must have timestamp equal to zero*, meaning that either all the nodes or all the edges changed. This can happen because an edge or a node just appeared (like the edge between the two

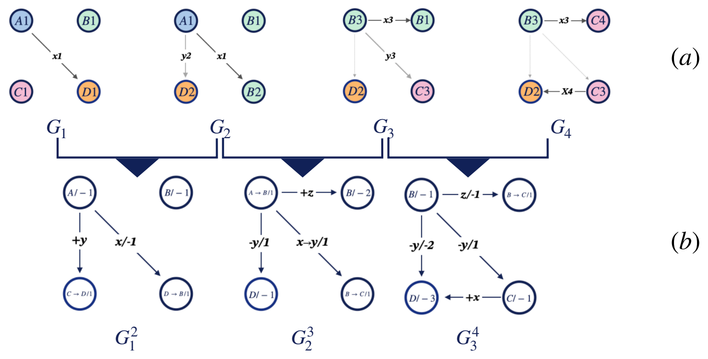


Figure 4.17: Union graph example. *The sequence of four graphs in (a) is transformed into the sequence of union graphs in (b).*

upper nodes in Figure 4.16c) or because it changed its label (like the blue upper-right node in Figure 4.16c);

- *The nodes and the edges that appeared in G_c* (the nodes belonging to $V_c \backslash V_a$ or the edges in $E_c \backslash E_a$) *must have timestamp equal to zero*, respecting the representation definition. In the example of Figure 4.16c, this condition is reflected in the pink bottom-left node and in all the edges;
- For all edges and nodes that both G_a and G_c present, *if the timestamp in G_c is the exact precedent of the one in G_a, then the label must be the same.* Otherwise, *if the timestamp in G_c is greater or equal the one in G_a, then the labels must change.* The latter situation is represented by the blue upper-left node in Figure 4.16c. Since the timestamp in G_c is equal to the one in G_a, the label changed. On the other hand, the green upper-right node maintains the same label, in fact, it holds that the timestamp in G_c is the precedent of the one in G_a.

4.7.3 Union graph representation

Starting from the dynamic representation of graphs and rules, it is necessary to define the union graph representation. The role of the union graph is to transform the dynamic graph into a sequence of static graphs, allowing the application of the gSpan algorithm.

Figure 4.17 illustrates an example of a dynamic graph and its union graphs, one for each couple of consecutive snapshots. Each union graph G_i^{i+1} is realized observing the following rules:

Unchanged edge or node If the node or edge does not undergo a change in the $i+1-th$ timestamp, then the annotation will record how many timestamps ago the edge/node changed. Considering the example of Figure 4.17, the upper-right node in the union graph G_2^3 is annotated with $B/-2$ because the last time when the node changed label was into B, 2 timestamps ago. The same happens for the edge between the two upper nodes in G_3^4: the annotation is $z/-1$ because 1 timestamp ago the edge appeared with label z.
Addition If in the $i+1-th$ snapshot the graph registers a new node or edge with label l, it will be denoted as $+l$. An example is the edge between the two left nodes in G_1^2, denoted with $+y$.
Deletion When a node or an edge with label l is deleted in snapshot $i+1$, then the annotation will be $-l/t$, with $t=1$. An example is the edge between the 2 left nodes in G_2^3, with annotation $-y/1$. Note that t is $-n$ if the deletion happened n timestamps ago instead of the current snapshot $(i+1)$, like in the case of the same edge between the left nodes but in G_3^4, denoted with $-y/-1$.
Relabeling The idea is the same as the deletion case, but instead of $-l$ the annotation will remark the previous and current label. So, if a node or edge changes the label from l_1 to l_2 in the $i+1-th$ snapshot, the annotation will be $l_1 \to l_2/1$. An example of Figure 4.17 is the upper-left node in G_2^3, denoted with $A \to B/1$. In the same way as the deletion case, if the relabeling happened n timestamps ago, the annotation will be $A \to B/-n$.
Time abstraction The union graph representation can also be extended using two methods of time abstraction. Basically, instead of considering how many timestamps ago the change happened, the extension will aggregate all negative timestamps as -1 and all positive timestamps as 1. The time abstraction can be applied at two levels:

- only on vertices, useful for graphs where the evolution mainly concerns the edges and not label vertices;
- on both vertices and edges, particularly suited for graphs with a lot of different patterns. With the time abstraction, a lot of patterns with different time delta will be aggregated.

4.7.4 DGR algorithm

DGR Miner algorithm combines the two typical phases of a graph evolution rule method, which are frequent patterns discovery and graph evolution rules extraction. Thanks to the peculiar graph representation, the predictive rules represented as condensed union graphs are already the final result. Basically, the algorithm discovers the frequent patterns, selecting the ones with support and confidence over a fixed threshold (see the next subsection for details about the support and confidence measure).

Frequent pattern discovery The pattern discovery phase leverage the same algorithm as many others GER methods, that is gSpan, modifying some details to make it fit their graph representation. The main feature to update is the way edges are represented because the 5-tuples used by gSpan $(i, j, l_i, l_{(i,j)}, l_j)$ cannot include the information about the direction of edges and relative timestamp. To solve this limitation the authors use a 9-tuple in the following format:

$$(i, j, l_i, t_i, d_{(i,j)}, l_{(i,j)}, t_{(i,j)}, l_j, t_j)$$

where $t_i, t_j, t_{(i,j)}$ corresponds to the relative timestamp of node i, node j and the edge (i, j) respectively. Moreover, $d_{(i,j)}$ specifies the direction of the link, so the possible values are $\leftarrow, \rightarrow, -$ (the last symbol corresponds to undirected edges). Using the 9-tuples, patterns can be ordered and the minimum DFS code can be found for each pattern. The modified version of gSpan is then applied to the set of union graphs obtained from the input dynamic graph.

4.7.5 Support and confidence

The support definition is based on the occurrence concept. A pattern P occurs in G_i, written as $P \sqsubseteq G_i$, if there exists a function $\Phi : V_P \to V_{G_i}$ that maps nodes in P to nodes in G_i. The support of a rule $P_a \to P_c$ is the ratio between the number of snapshots where P_a and P_c occur consecutively over the number of snapshots. Formally, the support is defined as follows:

$$\sigma(P_a \to P_c) = \frac{|\{i | P_a \sqsubseteq G_i, P_c \sqsubseteq G_{i+1}, 1 \le i \le n-1\}|}{n-1}$$

The confidence of a rule is intuitively the support of the rule over the support of P_a, formally defined as:

$$conf(P_a \to P_c) = \frac{\sigma(P_a \to P_c)}{\sigma(P_a)}$$

where: $\sigma(P_a) = \frac{|\{i | P_a \sqsubseteq G_i, 1 \le i \le n-1\}|}{n-1}$.

Chapter 5

Null models classification

Randomized Reference Models (RRMs) have emerged as a powerful tool in the study of complex networks, offering crucial insights into their intricate dynamics. These models involve the creation of random networks derived from empirical data, with the randomization process constrained by specific features of the input network. The goal is to generate random versions that match the original network in key characteristics such as diameter, size, or other relevant metrics. This approach allows researchers to distinguish between properties that arise from the network's inherent structure and those that are simply artifacts of its basic characteristics. A significant subset of RRMs employs uniform sampling of networks based on preserved features, drawing parallels with microcanonical ensembles in statistical physics. Recognizing this connection, Gauvin *et al.* [86] introduced the term *Microcanonical Randomized Reference Models (MRRMs)* and provided a comprehensive taxonomy of existing works in this field. In this chapter, we will summarize Gauvin's survey to establish the foundational concepts necessary for the discussions in subsequent chapters, highlighting the importance of MRRMs in unraveling the complexities of network structures and dynamics.

5.1 Microcanonical Randomized Reference Models - MRRMs

Null models are crucial for understanding both theoretical and practical aspects of networked systems, providing a baseline for comparison against which the observed patterns, features, and dynamics can be evaluated. By generating randomized or synthetic versions of the original temporal network, null models allow us to assess the significance and uniqueness of observed patterns, identify deviations from randomness, and uncover meaningful structures and processes in the data. According to [86], given a space $\mathcal{G}$ of all possible temporal networks (states) and the single observation $G^* \in \mathcal{G}$, all models that sample a random graph G from a conditional probability $P(G|G^*)$ are defined *randomized reference models* (RRMs). In this context, the most popular models include configuration models, Erdös-Rènyi (ER) models, and exponential random graph models [87]. Depending on the application scenario, it could become necessary to preserve specific features or properties while generating random temporal graphs. This is where *microcanonical randomized reference models* (MRRMs) come into play, providing a framework that allows us to retain specific characteristics of the graph while randomizing the rest. The concept behind this approach is preserving certain features while maximizing randomness. Relying on the principle of maximum entropy, the use of such models is theoretically justified since they offer the least biased approach among all possible degrees of freedom [86]. Taking the definition from [86], given the observation G^* in the state space $\mathcal{G}$, a MRRM is defined as a model that returns $G \in \mathcal{G}$ with probability

$$P_x(G|G^*) = \frac{\delta_{x(G),x(G^*)}}{\Omega_x(G^*)}$$

Here $x : \mathcal{G} \to F$ is a function that returns a property of a graph $G \in \mathcal{G}$ and δ is the Kronecker delta function. Finally, the normalization is given by $\Omega_x(G^*) = \sum_{G \in \mathcal{G} \delta_{x(G),x(G^*)}}$. Moreover, features can be combined to form more complex constraints. In this case, the MRRM constrained on the features $x_1, ..., x_q$ is denoted as $P[x_1, ..., x_q]$.

5.2 Representation of temporal networks

Gauvin *et al.* [37] categorized MRRMs into 4 categories, based also on the temporal network representation. So, before delving into the categorization, this paragraph will detail the two possible macro representation proposed by [37].

Temporal graphs can be either represented in a timeline representation or in a snapshot representation. Using the timeline representation, the temporal graph is defined as $\mathcal{G} = (G, \Theta)$, where $G = (V, E)$ is the static graph representing the pair of nodes interacting during the period of observation. While Θ is the set of timelines, one assigned to each of the links $(i, j) \in E$, with $\theta_{(i,j)} = ((t^1_{(i,j)}, \tau^1_{(i,j)}), ..., (t^n_{(i,j)}, \tau^n_{(i,j)}))$. Figure 5.1a shows an example of the

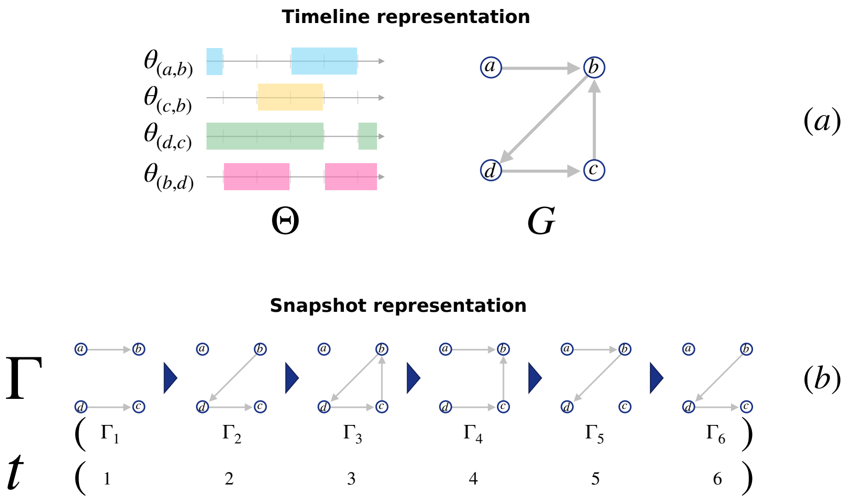


Figure 5.1: Reprentation of temporal graphs *according to Gauvin et al. [37]. (a) shows the timeline representation, which consists in a static graph G and a timeline multiset Θ. The snapshot representation depicted in (b) instead is composed of a sequence of static graphs Γ and a set of timestamps t.*

timeline representation of a temporal graph. On the other hand, Figure 5.1b shows the snapshot representation of the same graph $\mathcal{G} = (\Gamma, T)$. The Figure shows the sequence of static graphs $\Gamma = (\Gamma_1, \Gamma_2, ..., \Gamma_n)$, while the other element of the definition $T = (1, 2, ..., n)$ is the sequence of timestamps observed. According to the taxonomy proposed in Chapter 3,the timeline representation corresponds to a version of the evolving projection $G^{EP}_{[1,T]}$ where the timelines are defined by the labeling function f^{EP} instead of being the separate set Γ. On the other hand, the snapshot representation corresponds exactly to the sequence of static graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$ defined by the interval model.

5.3 MRRMs taxonomy

MRRMs can be classified according to two factors: *i)* preservation of temporal distribution or topology, and *ii)* the representation of temporal networks as either timelines or snapshots.

		Graph representation	
	MRMMs categories	Timeline representation	Snapshot representation
What's preserving	Topology	*Timeline shuffling*	*Sequence shuffling*
	Temporal distribution	*Link shuffling*	*Snapshot shuffling*

Figure 5.2: *Four primary categories of microcanonical randomized reference models. Each row represents the characteristic of the temporal network which is preserved, and each column refers to the representation of the temporal network, i.e. stream-based or snapshot-based.*

When models preserve the temporal distribution they favor capturing the temporal dependencies and dynamics of the original network. Conversely, other models might prioritize preserving network topology to gain insights into structural properties. The second factor revolves around the representation of temporal networks. In the timeline representation, the temporal ordering is crucial, with the graph capturing the static topology and additional time-series information describing the timing and duration of edges/events for each node. On the other hand, the snapshot model treats each time window as a separate graph (potentially with the same set of nodes), enabling a focus on individual time points. The combination of these two factors leads to four primary categories of MRRMs depicted in Figure 7.2. The following paragraphs will detail the categories separately.

5.3.1 Timeline representation: Timeline and link shuffling

Models in the **timeline shuffling** category aim to preserve network topology while using a timeline representation. This approach maintains the static network structure but shuffles timestamps within or between timelines, highlighting significant dynamics. The **link shuffling** category encompasses models that preserve the temporal distribution on a timeline-represented network. In this case, static links are shuffled while their associated timelines remain unchanged. Figure 5.3 illustrates a temporal graph $(G\text{'}\Theta)$ (left) and its most random timeline and link shuffling. Timeline shuffling is defined as $P[\mathcal{L}, E]$, where $\mathcal{L}$ represents the static graph's edge set and E the total event count. This example features 14 events (single-colored blocks representing instantaneous events) shuffled across 4 timelines (one per edge of G), while the static graph remains unchanged ($G^* = G$). Conversely, link shuffling MRR is denoted as $P[\Theta]$, preserving the timeline set Θ. The bottom-right of Figure 5.3 displays link-shuffled timelines with altered edge assignments, reflected in the randomized version G.

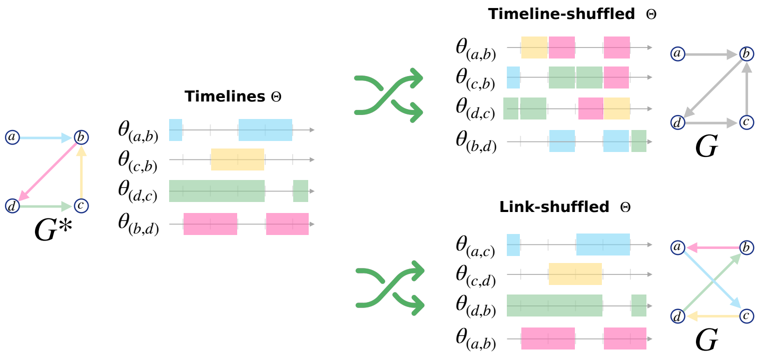


Figure 5.3: Timeline and Link shuffling MRRM. *Two randomization techniques applied to a dynamic graph that uses the timeline representation. On the left, we see the original graph G^* with its corresponding event timelines. The upper-right shows the timeline-shuffled method, where the static structure of G^* remains unchanged in G, but individual events within the timelines are randomly rearranged. The bottom-right depicts the link-shuffling method, where edges in G^* are switched while preserving the original timelines.*

5.3.2 Snapshot representation: Sequence and snapshot shuffling

In snapshot representation, MRRMs can be categorized as **sequence shuffling** when aiming to preserve topology. This method shuffles snapshot order while maintaining network topology within each snapshot. Conversely, when preserving temporal distribution on graphs in snapshot representation, the MRRM falls under the **snapshot shuffling** category. This approach involves shuffling edges within individual snapshots, allowing for isolated examination of time points. Figure 5.4 illustrates a temporal graph (Γ^*, t) -where Γ^* represents the graph sequence and t the timestamp sequence- along with its most random sequence and snapshot shuffling. Sequence shuffling MRRM is defined as $P[\Gamma]$, preserving snapshot topology. The example uses color-coding for each snapshot in Γ^* to highlight that randomization in sequence shuffling involves reordering unmodified snapshots. Snapshot shuffling, represented by MRRM $P[t]$, maintains the unmodified sequence t (shown below the graph sequence) while randomizing each snapshot's topology, as depicted in Figure 5.4.

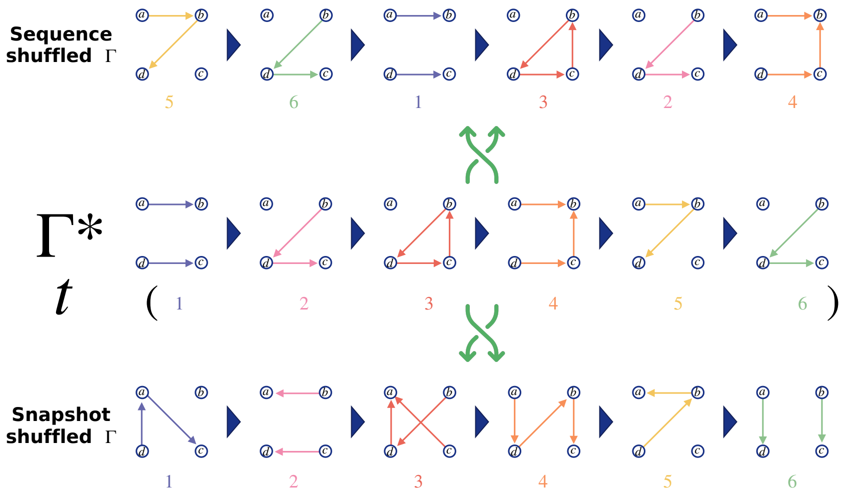


Figure 5.4: Sequence and Snapshot shuffling MRRM. *Two randomization techniques applied to a dynamic graph that uses the snapshot representation. In the middle, we see the original graph sequence Γ^* with its corresponding timestamps list t. The upper graph sequence shows the sequence-shuffled method, snapshots order is shuffled. The bottom graph sequence instead depicts the snapshot-shuffling method, where edges in each snapshot are shuffled, but the temporal distribution is unmodified.*

In general, MRRMs represent a valid tool for assessing the significance of measurements on real-world temporal networks and for extracting meaningful insights about their dynamics and structure. Indeed, the literature offers various applications and fields where MRRMs have been employed, including but not restricted to contagion processes [88], temporal motifs [89], and random walks [90].

Chapter 6

Case studies

This chapter introduces the key datasets and case studies that underpin the analyses throughout this thesis. Each dataset comprises a series of timestamped operations, transactions, or relationships, providing a temporal snapshot of network evolution. We begin by outlining the semantic content of each dataset, explaining what the data represents and its real-world context. Following this, we describe the modeling approaches applied to transform these temporal sequences into structured representations suitable for our analytical frameworks. This chapter thus lays the groundwork for the in-depth analyses presented in subsequent sections, offering both a clear picture of our data sources and the methodological foundation for their interpretation.

6.1 UC-social

One of the used dataset is obtained from a directed temporal network publicly available at `http://konect.cc/networks/opsahl-ucsocial/`, which collects 59835 message interactions among 1957 students of the University of California (Irvine) in an online community. Each interaction od the dataset (i, j, t) denotes a message sent from i to j at time t. We modeled the interactions as a directed temporal graph recording, for each couple of users interacting with each other, only the first one. According to the taxonomy of Chapter 3, the graph is a growing projection $G^{GP}_{[1,T]} = (V, E, f)$, with the edge-labeling function f that assigns the timestamp of the first interaction for each couple of nodes (i, j). The obtained graph contains 20296 edges connecting 1957 nodes (users).

6.2 DBLP datasets: citations and co-authoriship

DBLP, short for Digital Bibliography & Library Project, is a computer science bibliography website and it is the source of two datasets used in this thesis, recording different type of interactions among researchers: co-authoriships and citations. This section will detail the two datasets separatly.

DBLP-Citations The first DBLP datasets[1] [91, 92] records the citations between publications from 1975 to 2010, for a total of 49759 interactions. Even if data collect a period of 35 years, links are not equally distributed as shown in Figure 6.1. Consequently, we selected the most active period (20 years), from 1980 to 1999, and removed self-loops. We modeled the filtered list of

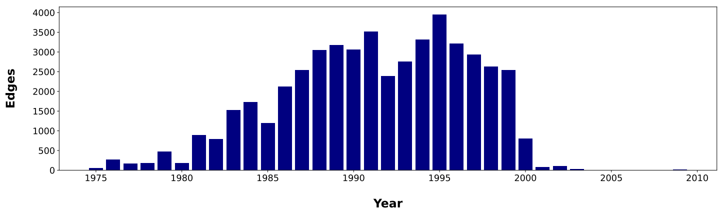


Figure 6.1: DBLP. *Number of edges year-by-year. On the x-axis the period covered by the original dataset, and on the y-axis the number of edges created in a specific year. After 1999, we observed a drastic drop in the creation of new edges.*

interaction (citations), as the UC-social one, into a graph that, according to the taxonomy proposed is a growing projection $G^{GP}_{[1,T]} = (V, E, f)$, with the edge-labeling function f that assigns the timestamp of the first interaction for each couple of nodes (i, j). Note that in this case there are no multiple interactions among the same couple of nodes (publication), so the first timestamp correspond to the only one. The obtained graph presents $47,654$ edges and $11,971$ nodes.

DBLP co-authorship The second DBLP-dataset used in this thesis is publicly available at `https://www-kdd.isti.cnr.it/GERM/` and gathers all the co-authorship interactions from 1992 to 2002 with a yearly granularity. The list of co-authorsips relations are modeled into a growing projection temporal graph $G^{GP}_{[1,T]} = (V, E, f)$, with each couple of co-authors being linked with an edge

[1] The dataset is publicly available at `http://konect.cc/networks/dblp-cite/`.

associated with the year of their first collaboration. The obtained network has 129073 nodes and 277081 edges, with 11 possible timestamps.

6.3 Enron email dataset

The Enron dataset [2] consists of 1148072 email communication records among Enron employees during a critical period (from 1999 to 2003). This dataset emerged from the investigation into the Enron scandal[93], which revealed extensive accounting fraud that led to the company's bankruptcy in December 2001 and significant financial losses for investors and employees. The Federal Energy Regulatory Commission (FERC) obtained these emails as part of their inquiry into the company's practices, and later made them publicly available for transparency and research purposes. We first aggregate timestamps on with a weekly granularity (meaning that we aggregated into the same timestamp all the edges within the same week) and then analyze the distribution of edges over weeks, shown in Figure 6.2. We selected the period from January 2001 to June 2001. This period lies within the most active one and is also temporally close to the scandal. The filtered datasets is composed of tuples (x, y, t) denoting emails sent from employee x to employee y at timestamp t. We modeled this dataset in a graph $G^{GP}_{[1,T]} = (V, E, f)$ respecting the growing projection type (see taxonomy in Chapter 3), with the edge-labeling function f that assigns the timestamp of the first email for each couple of employees (i, j). In this way, we obtained a temporal directed graph of $32,178$ nodes and $107,234$ edges. Since the input graph presented several connected components, we only considered the largest connected component of the graph, including 89.1% of nodes; finally obtaining a graph of $31,312$ nodes and $106,642$ edges.

6.4 Stack Overflow

The Stack Overflow dataset[3] captures $17,823,525$ interactions from the Stack Exchange website, specifically focusing on the `a2q` edge list, which records answers to questions.

Each edge (u, v, t) denotes user u answering user v's question at time t. This extensive dataset is composed of tuples (u, v, t) denoting a user u answering user v's question at time t. It involves $2,464,606$ users and spans from October 14, 2008, to March 6, 2016, totaling 2,774 days. From the distribution of edges

[2] The dataset is publicly available at `http://konect.cc/networks/enron/`

[3] Access the dataset at `https://snap.stanford.edu/data/sx-stackoverflow.html`

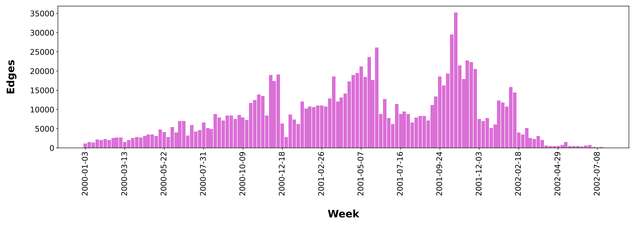


Figure 6.2: Enron. *Number of edges week-by-week. On the x-axis the period covered by the original dataset with a weekly granularity, and on the y-axis the number of edges created in a specific week.*

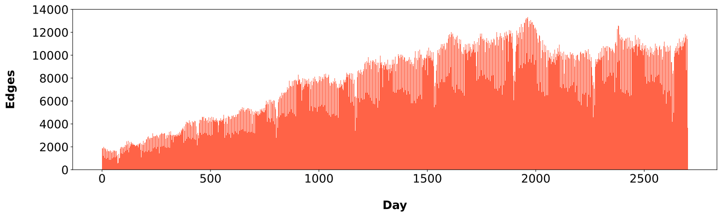


Figure 6.3: Stack Overflow. *Daily number of edges. On the x-axis the index of the days covered by the dataset, and on the y-axis the number of edges created in a specific day.*

per day depicted in Figure 6.3, we decided to filter the dataset considering edges happened from the 500^{th} to the 2500^{th} day, covering a period of $2,000$ days within the most active one. As for the previous datasets, we modeled the interactions (answers to questions) with a graph respecting the growing projection of taxonomy in Chapter 3, obtaining a graph $G^{GP}_{[1,T]} = (V, E, f)$ where the labeling function selects the first moment in time where two nodes interact. This process yields a temporal directed graph with $2,115,635$ nodes and $13,519,681$ edges. In fact, the Stack Overflow temporal network represents the largest case study used in this thesis.

6.5 Bitcoin Alpha

The Bitcoin system [94] allows anonymous transactions between users. Numerous trading platforms have emerged to enhance usability, many of which track user reputations to safeguard against fraudulent and high-risk users. The

Bitcoin Alpha dataset [95, 96] includes $24,186$ user ratings exchanged among $3,683$ users. These ratings are unique and predominantly positive ($22,650$ out of $24,186$). The dataset has been integrated into several major graph analysis and machine learning libraries like SNAP [97], due to its network structure and the wealth of additional information it provides, such as time and weight (ratings). According to the taxonomy in Chapter 3, once again we model the links into a growing projection graph $G^{GP}_{[1,T]} = (V, E, f)$ with tiemstamps on the edges, assigned by he labeling function f, recoding the moment of first (and only) trust expression. Even if the datasets includes trust and distrust operations (denoted with a positive or negative weight respectively), we only include trust links to make the network homogeneous. In this way, we obtain a network of $3,683$ nodes and $22,650$ edges.

6.6 Sarafu dataset

The Sarafu dataset represents the first case study described in this chapter to be part of the Web3, because of its use of the blockchain technology. Sarafu[4] is a digital CC token created by the Grassroots Economics (GE) Foundation [98], a humanitarian aid foundation.

Complementary currencies. Complementary currencies (CCs) in general are currencies that originate in various geographic situations to supplement the official national currency [99]. CCs can also be viewed as a fungible "voucher" or credit obligation redeemable for products and services, [100]. There are many instances of CC systems all around the world, with an estimated 3,500 to 4,500 CC initiatives in more than 50 nations since the 1980s [101, 102].

The Sarafu project Sarafu users may perform payments using mobile phones, transferring Sarafu digital tokens to other registered users. As described in Ussher *et al.* [100] the Kenyan Red Cross relied on Sarafu tokens to provide humanitarian aid during the COVID-19 pandemic: users registering were given free Sarafu tokens, backed by donors' money, to maintain the system running. The use of blockchain technology is a key component of Sarafu. While the Sarafu project has not used blockchain technology from its inception, it has used it to solve several important issues [100]. Among the motivations, we have enhanced transparency, as transaction data allows contributors to fully disclose the impact of their donations. Furthermore, data analysis can lead to more informed decision-making processes regarding, for example, future investments and project functioning, while it also helps the GE Foundation to

[4] Sarafu means "currency" in Kiswahili

find ways to improve user welfare and minimize potential misuse. The system first moved to a blockchain maintained privately called *POA*. The name is derived from its consensus protocol, Proof of Authority [103], PoA in short. The project then switched to a public blockchain named xDai blockchain in 2020 to lower transaction costs[100]. Finally, in May 2022, the project transitioned to a new blockchain built by the GE Foundation to better meet its objectives. Kitabu (“Ledger" or “Book" in Kiswahili) is the name of the new blockchain, which is based on the Proof of Authority consensus protocol. The Sarafu dataset consists in blockchain transactions of the sarafu token, with associated attributes like timestamp and amount of cryptocurrency exchanged. The dataset present also attributes on nodes, like business type or geographical information. In our study, we are focused on transactional data only, and this datasets includes 412,050 transactions involving 53,277 users, covering the period between January 2020 to June 2021. Following the growing projection modeling of the taxonomy described in Chapter 3, we build a temporal graph $G^{GP}_{[1,T]} = (V, E, f)$ where we record, for each couple of users that interacts at least once in the covered period, the timestamp of their first interaction (assigned by the function f). In the resulting temporal network, we have 40,343 nodes and 143,239 links.

6.7 NFTs sales dataset

Among Web3 platforms, we focused on NFT marketplaces. An NFT is a blockchain-based data unit with a double goal: first, it provides a unique certificate of ownership of a digital object. Second, it attests to the uniqueness and non-transferability of a digital asset. Thanks to this technology, it is possible to track down the complete history of ownership of an object and check its authenticity. In concrete terms, an NFT can represent a variety of digital items, including photographs, movies, and audio. As a consequence, several contexts, such as art, gaming, and sports collectibles, utilize NFTs to regulate and control digital objects. The birth of the NFT market can be traced back to late 2017 when the blockchain game Cryptokitties gained popularity. However, the market remained dominated only by Crypokitties until July 2020 when it started to grow and in March 2021 reached a peak of popularity, due to the selling of an artwork’s NFT for $69.3 million. This purchase allowed the author, Beeple, to reach one of the highest auction prices for a living artist. The case studies involved in this thesis are extracted from the NFTs sales collection by Nadini *et al.* [104], collecting transactions on multiple marketplaces (APIs): *Cryptokitties, Atomic, Opensea, Gods-unchained*, and *Decentraland.*

The data collection is composed of 6.1 million trades of 4.7 million NFTs in 160 cryptocurrencies, primarily Ethereum and WAX, covering the period from June 23, 2017 to April 27, 2021. Figure 6.4 shows when the different markets were launched precisely. In this thesis, we focus on two NFT marketplaces: CryptoKitties and OpenSea. These platforms were selected for their overlapping operational periods and significant market presence.
CryptoKitties, launched in 2017, is one of the earliest NFT projects. It a game based on the Ethereum blockchain technology, featuring collectible digital cats that users can breed and trade. It gained widespread attention, at times congesting the Ethereum network due to its popularity. The subset of the dataset in [104] related to transactions on Cryptokitties is composed of 725400 transactions involving 99984 wallets (users).
OpenSea, established in late 2017 (our first recorded transactions is dated February 4,2018), is also based on the Ethereum blockchain. OpenSea has evolved into the largest NFT marketplace, offering a wide array of digital assets including art, music, domain names, and virtual world items. The dataset by Nadini *et al.* [104] includes 1907262 transactions made on the OpenSea market.

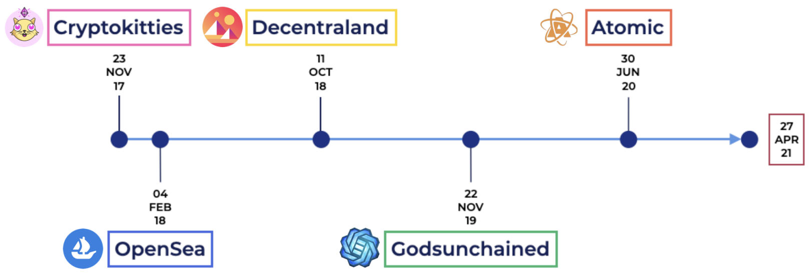


Figure 6.4: Timeline of NFT markets. *Markets are shown on a timeline ordered by the time of their first transactions. The dataset covers the period from the introduction of Cryptokitties on November 23, 2017, up to almost a year after the first transaction in the Atomic market.*

The modeling approaches for these datasets are diverse and context-specific. As such, we will present and discuss the specific modeling techniques in the respective chapters where these datasets are utilized, ensuring a clear connection between the data, the modeling strategy, and the research questions being addressed.

6.8 Steemit

The last set of dataset explored in this thesis is sourced in the Steemit platform. Steemit[5] is one of the pioneering systems of the Web3 ecosystem since it introduced the concept of a rewarding system in a social network [105, 106] and employed the Delegated Proof-of-Stake (DPoS) consensus algorithm for block validation in social network applications. In general, Steemit is a blockchain-based blogging and social media platform that stands at the intersection of content creation and cryptocurrency incentives. Steemit's users publish and share multimedia content, engaging with it mainly through comments and votes. Due to its social platform nature, users can follow others, allowing them to receive notifications whenever new content is posted by those they follow. Steemit is among the first platforms to adopt a reward mechanism, compensating users with cryptocurrency tokens for writing or voting on highly successful articles. The reward system and several other components of the Steemit ecosystem are realized through blockchain technology, specifically the Steem blockchain. In this section, we delve into the technical details of Steemit, exploring the inner workings of its token system, the mechanisms governing reward distribution, and the processes that define user interactions within this decentralized ecosystem. Moreover, details about the Steem blockchain and the block validation - witnesses - are provided. Finally, we detail the datasets we extracted from the Steemit platform.

The basic elements of blockchain-based social platforms Blockchain-based social platforms introduced a few peculiar aspects w.r.t. the mainstream online social media, mainly connected to their foundational element: the blockchain. To this aim, it is crucial to clarify the main terms, essential to understanding the Steemit ecosystem, and in general blockchain-based social platforms. *Steem* represents both the blockchain upon which the system is built and the native token (cryptocurrency) the rewarding system is based on[6]. Concurrently, *Steemit* denotes not only the platform itself - the software system - but also the overseeing company. In the micro-blogging platform Steemit, users primarily engage in the creation and curation (voting and commenting) of posts. These actions have an economical return since after a 7-day period post-publication, the value of the post or comment becomes fixed, leading to subsequent rewards for both the author (the user who created the post) and

[5] https://steemit.com/
[6] To differentiate the cryptocurrency from the blockchain, the former is also named STEEM.

curators (users who vote/comment on the post). As detailed later, the token system supported by the Steem blockchain plays a fundamental role during the creation of posts, in voting, and in the distribution of the generated wealth.

Token system In Steemit, the token system provides three different tokens: (i) STEEM as its native cryptocurrency, (ii) SBD with a value aligned to the US Dollar - token pegged to USD, and (iii) Steem Power (SP) representing a distinctive form of investment in the ecosystem. The schema of Figure 6.5 details the token system and the relationships among tokens: STEEM, the main platform token, can be exchanged for SP, through an operation called *power-up*. This operation is an investment that translates into platform advantages, such as enhanced voting influence and increased posting/voting capabilities. Conversely, users can convert SP back into STEEM through the *power-down* operation; however, to discourage disinvesting from the platform and getting more liquidity (STEEM or SBD), the conversation is a long process lasting 4 weeks[7]. Additionally, STEEM can be (externally) converted into a more universally recognizable currency, SBD, mirroring the value of the US Dollar. On the other hand, the conversion from SBD to STEEM can be done internally on the platform. The shift from SBD to STEEM unfolds over 3.5 days, accompanied by a conversion fee equivalent to the average value of STEEM in US Dollars during this time. Alternatively, for users seeking expedited transactions, an immediate conversion can be executed within the internal market at the current market price. Finally, users can buy and sell both SBD and STEEM through external systems. In fact, STEEM and SBD have the highest level of liquidity in the token system. With the creation of each new block, a

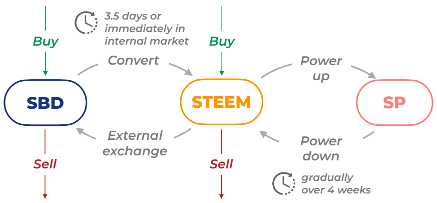


Figure 6.5: Steemit token system. *The illustration summarizes the three tokens making up the Steemit token system and their exchange operations. The arrows also report the main temporal constraints for exchanging SP into STEEM and SBD into STEEM.*

[7] https://steemit.com/faq.html

fresh batch of tokens is generated. Diverging significantly from Bitcoin, in the Steem ecosystem, the majority of tokens is not allocated to miners, termed *witnesses* in this context. Instead, they contribute to the reward pool, a fund designed for curators and creators. Precisely, the distribution of tokens from the new block unfolds as follows: 65% to the rewards pool, 15% to vesting authors (users holding STEEM), 10% to the Steem proposal system [8] (a platform for proposing and accepting developments within Steem), and the remaining 10% allocated to witnesses.

Posts and comments As mentioned earlier, posts constitute a key element of the Steemit blogging platform. Upon creating a post, authors are prompted to specify whether they prefer a 100% SP reward - all the reward is invested in the platform - or a split of 50% SP and the remaining 50% in STEEM - half of the reward is liquidity. During the 7-day reward-decision period, the value of a post may increase through additional upvotes, the removal of downvotes, a rise in STEEM value, or if other posts receive more downvotes. Conversely, the value may decrease with downvotes, increased upvotes on other posts, removal of upvotes, or a decrease in STEEM value. Platform-wide, post values are displayed in SBD, reflecting the average value over the last 3.5 days, as reported by witness users. After the 7 days, the final reward is divided, allocating 50% to the author (the user who created the post) and the remaining 50% distributed among curators (users who voted) proportionally to the SP they hold. The user's capacity to create posts is proportional to their Steem Power, with a minimum interval of 5 minutes between posts. Post modifications are possible within the 7-day reward-decision window, retaining all versions on the blockchain, while only the latest version is displayed in the front-end. Users can promote their posts by spending SBD, ensuring visibility in the "promoted" tab based on the amount spent. Additionally, users have the option to promote other users' posts. Comments mirror posts in that they are subject to voting, with authors and curators receiving rewards after the 7-day period.

Votes As a consequence of the mechanism of the reward distribution, engagement in the Steemit ecosystem revolves not only around posting but also voting, constituting the main pathway for earning, when executed thoughtfully. To safeguard against potential abuse, a systematic approach is in place. Firstly, a vote must be cast at least 5 minutes after the publication of a post; failing to adhere to this timeframe results in a portion of the potential reward reverting to the reward pool. Secondly, each user possesses a voting mana — a form of energy bar — that depletes with each vote. Users typically expend 2%

[8] https://steemit.com/sps/@flaws/what-is-the-steem-proposal-system-sps-and-why-is-it-very-important

of their mana when casting a vote with 100% voting power. However, users possessing 500 SP or more earn the flexibility to customize the intensity of mana utilized for each vote. The impact of a vote is directly proportional to the voting mana possessed by the voter (other than the SP level). Mana recharges by 20% daily, and a distinct voting mana is allocated for downvotes. Crucially, curation rewards are exclusive to upvote operations, while downvotes serve a distinct purpose. Primarily administered by influential users known as whales, downvotes function as a means of moderation, contrasting spam and copyright violations.

Blockchain system To conclude this comprehensive overview of the Steemit ecosystem, we delve further into the foundation of the platform: the Steem blockchain. One notable aspect is that every operation on the blockchain, such as posting, voting, commenting, and other actions, is inherently free of charge. However, to mitigate spam, the concept of resource credits (RC) comes into play. Users are allotted a limited RC per week, proportionate to their SP. During peak blockchain activity, operations may require more RC.

In terms of consensus algorithm for block validation, the Steem blockchain operates on the Delegated Proof-of-Stake (DPoS) protocol. This innovative approach involves delegating the task of mining/validating blocks to trusted accounts known as witnesses, who are the 21 most voted users for this role. With blocks mined every 3 seconds, each witness has the opportunity to mine a block approximately every 63 seconds. If a voted witness fails during their turn, the next one in the ranking assumes the responsibility in the subsequent turn. The first 20 witnesses are selected from the current voting rank, while the 21st is chosen from the remaining ranks with a probability proportional to their position. Witnesses are elected through votes from regular users, with each user having the capacity to vote for up to 30 witnesses simultaneously. This voting system is comparable to placing trust in 30 witnesses simultaneously. Should a user wish to vote for a 31st witness, they must remove their vote from one of the existing 30 witnesses.

6.8.1 Hive

Hive is a platform that shares the main characteristics of Steemit because it is actually born after an hard fork on the Steemit platform. The events leading to the Steem-Hive fork began in February 2020 when TRON, a gambling-oriented blockchain company led by Justin Sun, acquired Steem[9]. Initially, Steemit's

[9] `https://news.bitcoin.com/steemit-for-sale-tron/`

founder had allocated a reservoir of tokens intended solely for Steem ecosystem development, meant to be non-voting in governance issues [10]. However, post-acquisition, these guarantees were uncertain, prompting some active users to attempt freezing TRON's acquired tokens through a soft fork[11]. TRON, with assistance from cryptocurrency exchanges, managed to amass significant voting power on the platform, ultimately gaining control of over 51% of witnesses. This allowed TRON to elect its chosen witnesses and subsequently reverse the soft fork's effects[12]. In response to this perceived hostile takeover, Steem's original witnesses announced a hard fork[13], which occurred on March 20, 2020, giving birth to Hive[14]. As Hive shares the same pre-fork blocks with Steem, Hive witnesses took preventive measures by freezing or confiscating funds owned by those involved in the hostile takeover to safeguard the new platform. Hive also introduced innovations such as a delayed voting influence mechanism to mitigate potential future 51% attacks, providing the community with a window to respond proactively. The Hive platform is then considered in this thesis in the chapter dedicated to user migration.

6.8.2 Datasets

Steemit and Hive's open nature and comprehensive API access provide a unique opportunity for extensive data collection and analysis. Through these APIs, we can potentially retrieve every operation recorded on the blockchains, including a wide array of user interactions such as follows, comments, votes, transfers, and various other activities. This data offers a rich landscape for temporal network analysis, encompassing both social and financial interactions within a single ecosystem. The specific operations we obtained and the time periods covered will be detailed in the relevant chapters where this data is utilized. This approach allows us to tailor our data presentation to the particular research questions and analytical methods employed in each study, ensuring a clear connection between the data source, the modeling strategy, and the insights derived.

[10] `https://steemit.com/steem/@softfork222/soft-fork-222`

[11] `https://www.coindesk.com/tech/2020/02/24/justin-sun-bought-steemit-steem-moved-to-limit-his-power/`

[12] `https://www.coindesk.com/tech/2020/03/03/steem-community-mobilizes-popular-vote-in-battle-with-justin-sun/`

[13] `https://www.coindesk.com/tech/2020/03/17/steem-community-plans-hostile-hard-fork-to-flee-justin-suns-steemit/`

[14] `https://cointelegraph.com/news/hive-hard-fork-is-successful-steem-crashes-back-to-earth`

Part II

Stand Alone Rules

The field of graph evolution rules has seen limited exploration, with existing research often proposing algorithms without comprehensive analytical frameworks. These algorithms typically face constraints in their scope and applicability. This thesis aims to address these limitations by adding to the GERANIO framework part dedicated to stand-alone graph evolution rules, and introducing novel analytical tools.

We begin by developing a specialized null model for the GERM algorithm, inspired by Gauvin *et al.* [37]'s categorization. This timeline-shuffled null model prioritizes network topology preservation over temporal distribution, applied to temporal graph timelines. This approach aligns with GERM's graph representation and our focus on network dynamics rather than specific topology. The null model is applied to three distinct networks with varying time granularities: two social networks and one citation network. Results reveal the model's significant impact on rule evaluation and interpretation, highlighting under-represented rules and suggesting the influence of temporal factors and other mechanisms on network evolution.

Building on this, we introduce the GER profile, an analytical tool facilitating comparison of evolutionary behaviors across networks. This probability distribution of stand-alone rule frequencies characterizes network growth in diverse contexts. We analyze two NFT networks (CryptoKitties and OpenSea) and two BOSN networks from the Steemit platform (financial transactions and follow operations). Our analysis reveals both consistently frequent rules across networks and platform-specific rules, such as the prevalence of reciprocal traits in the only network with a social nature (follow operation on steemit).

After using the GER profile at graph level, we then extend the application to other scales: node and community levels. At node level, we develop a temporal behavioral node representation using GER profiles of ego networks. Analyzing the Sarafu complementary currency platform, we identify groups of nodes with similar evolution rules, revealing common interaction patterns and user behaviors.

Finally, at the community level, we compare GER profiles across four diverse networks (communication, social, citation, and online discussion), finding that similarly evolving communities tend to be proximate in the network.

These enhancements and applications advance the stand-alone GER framework, offering new tools and perspectives for understanding network evolution at multiple scales.

Chapter 7

Statistically significant rules

7.1 Introduction

Understanding and extracting knowledge from temporal networks is crucial to understand their dynamic nature and gain insights into their evolutionary characteristics. Existing approaches to network growth often rely on single-parameterized mechanisms, neglecting the diverse and heterogeneous behaviors observed in contemporary techno-social networks. To overcome this limitation, methods based on graph evolution rules (GER) mining have proven promising. GERs capture interpretable patterns describing the transformation of a small subgraph into a new subgraph, providing valuable insights into evolutionary behaviors. However, current approaches primarily focus on estimating subgraph frequency, neglecting the evaluation of rule significance. To address this gap, we propose a tailored null model integrated into the GERM algorithm, the first and most stable graph evolution rule mining method. Our null model preserves the graph's static structure while shuffling timestamps, maintaining temporal distribution, and introducing randomness to event sequences. By employing a z-score test, we identify statistically significant rules deviating from the null model. We evaluate our methodology on three temporal networks representing co-authorship and mutual online message exchanges. Our results demonstrate that the introduction of the null model affects the evaluation and interpretation of identified rules, revealing the prevalence of under-represented rules and suggesting that temporal factors and other mechanisms may impede or facilitate evolutionary paths. These findings provide deeper insights into the dynamics and mechanisms driving temporal networks, highlighting the importance of assessing the significance of the evolution patterns in understanding network evolution.

7.2 Background and related works

7.2.1 Graph Evolution Rules - GERs

Similar to association rules in data mining [107], a *graph evolution rule* - GER - consists of a precondition (referred to as the *body*) and a postcondition (referred to as the *head*). These rules can be interpreted as indicating that a subgraph matching the body is likely to evolve into the head, providing human-readable and explainable outcomes. For instance, Figure 7.1 depicts a graph evolution rule that indicates the presence of triadic closure in a directed graph. Graph evolution rules offer a powerful approach to uncovering complex mechanisms within temporal networks. In fact, they not only provide insights into the underlying dynamics but also facilitate the development of more precise models for predicting how the network will evolve. Furthermore, the collection of graph evolution rules extracted from a specific network can serve as a differentiating factor from other graphs [40], which may differently evolve driven by distinct mechanisms.

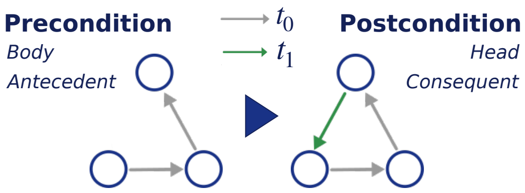
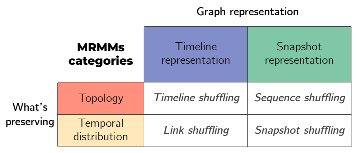


Figure 7.1: *Example of graph evolution rule - GER. On the left, the body of the rule, i.e. a graph with two links - grey arrows - created at time t_0, and on the right, the head of the rule represents the state of the body on the left side after evolving, indicated by the addition of a new link (green arrow) at the successive timestamp t_1.*

The identification of graph evolution rules has been the subject of different research works. In general, current state-of-the-art methods for detecting the topological evolutionary processes in a network follow a two-step methodology. Initially, rules are extracted through frequent subgraph mining, followed by the application of filtering techniques utilizing measures such as support and confidence, i.e. frequency-based properties. These steps collectively contribute to the identification and characterization of the evolutionary patterns of the network. One of the earliest methods introduced in the literature for extracting graph evolution rules is GERM, developed by Berlingerio *et al.* [29] in 2009. Its rules can detect undirected edge insertion events, with relative time, but the processes of removing edges and relabeling nodes and edges are not captured. Leung *et al.* [31], and later Ozaki *et al.* [32], introduced the LFR (Link Formation Rule) algorithm, which aims to capture the creation of directed links between source and destination nodes. Both GERM and LFR algorithms utilized the minimum image-based support [33] and gSpan-based techniques for frequent subgraph mining [30]. Ozaki *et al.* [32] extended LFR to an undirected version and proposed a method for identifying relationships between rules. LFR rules from both [31] and [32], similarly to GERM rules, do not capture edge or node deletion and relabeling. However, LFR rules represent a subset of the ones obtained with GERM, because it adds constraints focusing on the insertion of a new link between a given source and destination node. The targeted approach of LFR rules in reducing the search space for rule extraction accelerates the process, but it results in a reduction of the obtained results. Additionally, Vakulík [34] developed the DGR miner, which incorporates edge deletion and relabeling in the evolution rules. The most recent work on GER identification is the EvoMine approach, introduced by Scharwächter [35]. Evomine extracts frequent rules of events (including relabeling and deletion) occurring in subsequent time windows. Other works in the literature focus more on the evolution of attributes and give less importance to the structural evolution of networks and the rules governing their growth [108, 109].

7.2.2 Microcanonical Randomized Reference Models - MRRMs

The aforementioned methods for GER extraction primarily emphasize algorithmic aspects, often disregarding the evaluation of rule significance. In this study, we address these limitations by introducing a method to assess the significance of the rules, incorporating specific null models. Null models are crucial for understanding both theoretical and practical aspects of networked systems, providing a baseline for comparison against which the observed patterns, features, and dynamics can be evaluated. By generating randomized or synthetic versions of the original temporal network, null models allow us to assess the significance and uniqueness of observed patterns, identify deviations from randomness, and uncover meaningful structures and processes in the data. According to [86], given a space $\mathcal{G}$ of all possible temporal networks (states) and the single observation $G^* \in \mathcal{G}$, all models that sample a random graph G from a conditional probability $P(G|G^*)$ are defined *randomized reference models* (RRMs). In this context, the most popular models include configuration models, Erdös-Rènyi (ER) models, and exponential random graph models [87]. Depending on the application scenario, it could become necessary to preserve specific features or properties while generating random temporal graphs. This is where *microcanonical randomized reference models* (MRRMs) come into play, providing a framework that allows us to retain specific characteristics of the graph while randomizing the rest. The concept behind this approach is preserving certain features while maximizing randomness. Relying on the principle of maximum entropy, the use of such models is theoretically justified since they offer the least biased approach among all possible degrees of freedom [86].



		Graph representation	
	MRMMs categories	Timeline representation	Snapshot representation
What's preserving	Topology	*Timeline shuffling*	*Sequence shuffling*
	Temporal distribution	*Link shuffling*	*Snapshot shuffling*

Figure 7.2: *Four primary categories of microcanonical randomized reference models. Each row represents the characteristic of the temporal network which is preserved, and each column refers to the representation of the temporal network, i.e. stream-based or snapshot-based.*

MRRMs can be classified according to two factors: *i)* preservation of temporal distribution or topology, and *ii)* the representation of temporal networks as either timelines or snapshots. When models preserve the temporal distribution they favor capturing the temporal dependencies and dynamics of the original network. Conversely, other models might prioritize preserving network topology to gain insights into structural properties. The second factor revolves around the representation of temporal networks. In the timeline representation, the temporal ordering is crucial, with the graph capturing the static topology and additional time-series information describing the timing and duration of edges/events for each node. On the other hand, the snapshot model treats each time window as a separate graph (potentially with the same set of nodes), enabling a focus on individual time points. The combination of these two factors leads to four primary categories of MRRMs depicted in Figure 7.2. Models in the first category, namely *timeline shuffling*, focus on preserving topology while utilizing a timeline representation. It involves maintaining the static network structure while shuffling timestamps within or between timelines and allows highlighting significative dynamics. In contrast, the category of *sequence shuffling* aims to preserve topology but uses a snapshot representation. It shuffles the order of snapshots while keeping the same network topology in each snapshot. When a model preserves the temporal distribution on a network modeled through a timeline representation, it falls under the *link shuffling* category. Here, the static links are shuffled, while their associated timelines remain unmodified. Lastly, *snapshot shuffling*, the fourth category, preserves the temporal distribution on graphs represented as temporal snapshots. It involves shuffling the edges within each snapshot, enabling the examination of individual time points in isolation.

In general, MRRMs represent a valid tool for assessing the significance of measurements on real-world temporal networks and for extracting meaningful insights about their dynamics and structure. Indeed, the literature offers various applications and fields where MRRMs have been employed, including but not restricted to contagion processes [88], temporal motifs [89], and random walks [90]. However, to the best of our knowledge, the application of a null model on a graph evolution rule algorithm has not been explored, specifically for rules tracking the relative timestamp of edges without temporal window constraints. Although the LFR algorithm [31] introduces a null model, the rules in that context are highly constrained.

7.3 Methodology

In this section, we present the methodology employed in our study to assess the significance of the evolution of patterns in temporal networks and analyze their main properties and roles. First, we describe the original graph evolution rule mining algorithm (GERM), which serves as the foundation for our analysis. Subsequently, we outline the process of constructing the null model, a critical component in assessing the significance of observed patterns. Then, we explain the process of extracting over and under-represented rules, enabling us to identify statistically significant patterns. Finally, we compute the general mapping of patterns, which aids in understanding the overall trends within the graph. Through this methodology, we provide a robust framework for analyzing evolution patterns and uncovering their underlying significance. Among the different alternatives for GER extraction presented in Chapter 4, we opt for the GERM algorithm since it provides a more robust implementation and is able to identify GERs spanning many consecutive snapshots, see Section I.

7.3.1 Timeline shuffled null model

Inspired by the taxonomy of MRRMs for temporal networks proposed by Gauvin *et al.* [86], we implemented a microcanonical randomized reference model that falls in the timeline shuffling category. We based our selection of the proper null model on two guiding principles. Firstly, our focus is on analyzing the characteristics of the dynamics of the network, rather than its specific topology. Therefore, we specifically consider models that preserve the original topology. Secondly, we aim to maximize entropy, striving for highly randomized models that are as less biased and conservative as possible. Moreover, the graph representation adopted by GERM corresponds to a simplified version of the timeline representation described in [86]. In fact, GERM requires a unified topology merging all the temporal windows and then it adds a single timestamp to each edge - not a time series - corresponding to the first appearance of the interaction. For these reasons, the null model choice falls into the timeline shuffling category.

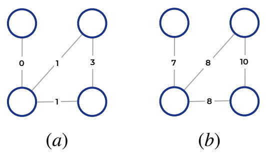


Figure 7.3: *Example of relative-time patterns. (a) and (b) are equal except for a time constant Delta = 7 for the link timestamps. The rightmost graph is representative of the class equivalence the graphs belong to.*

Formally, given an input temporal network $G^* = (V^*, E^*, f^*) \in \mathcal{G}$, our null model SM returns a $G = (V^*, E^*, f) \in \mathcal{G}$ with probability $P_x(G|G^*)$ where:

- V^*, E^* are respectively the node and edge set of the static input network;
- $f^* : E^* \to T^*$ is a function that maps the edges of G^* into a timestamp $t \in T^*$;
- $f : E^* \to T^*$ is a function that maps the edges of G into a timestamp $t \in T^*$;
- $\mathcal{G}$ is the state space, i.e. a predefined finite set of temporal networks among which the MRRM selects G;
- $P_x(G|G^*) = \frac{\delta_{(x(G),x(G^*))}}{\Omega_x(G*)}$;
- $\Omega_x(G*) = \sum_{G' \in \mathcal{G}} \delta_{(x(G'),x(G^*)})$;

and δ being the Kronecker delta function, with the feature x being the intersection of two features, namely the edge feature ($\mathcal{E}$) and the timestamp feature ($\mathcal{T}$). Specifically, δ is defined as follows:

$$\delta_{(x(G),x(G^*))} = \begin{cases} 1 & \text{if } \mathcal{E}(G) = E^* \text{ and } \mathcal{T}(G) = T^* \\ 0 & \text{otherwise} \end{cases}$$

According to the notation proposed in [86], we will refer to our model as $P[\mathcal{E}, \mathcal{T}]$, addressing the features that it has to maintain in sampling the randomized graphs. In other words, among all networks in the state space $\mathcal{G}$ having the same nodes set V^* and the same timespan (links can be assigned to a timestamp from 0 to $max(T^*)$), the null model SM samples a temporal network G having the same edge set as the input network G^*, i.e. $\mathcal{E}(G) = \mathcal{E}(G^*)$, and having the same set of timestamps - $\mathcal{T}(G) = \mathcal{T}(G^*)$) - but with a different mapping function f. The pseudo-code of the shuffle model is depicted in Algorithm 1.

Algorithm 1 Timeline shuffling model $P[\mathcal{E}, \mathcal{T}]$

```
Input: $G^* = (V^*, E^*, f^*)$, $T^*$ Output: $G$
1: $G = (V, E)$
2: $V = V^*, E = E^*$
3: $T = shuffle(T^*)$
4: $n = 0$
5: for $(i, j) \in E$ do
6:     $f(i, j) \leftarrow T_n$
7:     $n \leftarrow n + 1$
8: end for
```

7.3.2 Significative GERs

According to the *Bonferroni's Principle* [107], frequent patterns can be discovered even in random data. As the dataset size increases, the occurrences of these patterns also tend to increase. Such frequent patterns or events are considered false positives in the search for patterns that characterize the data we are analyzing. By applying the Bonferroni correction, the significance level of each test is adjusted to ensure that the probability of false positives is appropriately controlled. This correction helps avoid spurious findings and strengthens the reliability of the results. The *zeta score test*, also known as the standard score or z-score test, is commonly used to assess the significance of an observation compared to a reference distribution. In the case of identifying significant patterns using random models, the zeta score test allows us to quantify how deviant or exceptional a pattern is compared to what would be expected by chance alone.

In this work, we apply the GERM algorithm to an input graph G^* to get the real support s_p of each frequent pattern p. Then, we run the GERM algorithm on fifty different realizations through the MRRM described in the methodology section to extract the expected support μ_p and its standard deviation σ_p for each pattern $p \in \bigcap_{i=1}^{50} SM_i$, where $\bigcap_{i=1}^{50} SM_i$ is the set of patterns that are frequent in all the realizations of the null model. Note that the z-scores are computed over the mentioned set of rules $(\bigcap_{i=1}^{50} SM_i)$ since the support measure (μ_p) for each run is required; this aspect will be further discussed in the results section. The expected support corresponds to the average support of the pattern over all the 50 realizations of the model SM. Formally, it is defined as follows:

$$\mu_p = (\sum_{i=1}^{50} s_p^i)/50$$

where s_p^i is the support of p in the $i-th$ realization of the null model. Similarly, the standard deviation σ_p of each pattern $p \in \bigcap_{i=1}^{50} SM_i$ is computed on the supports of p over the realizations of SM. Finally, the z-score of the pattern p is computed as follows:

$$z_p = \frac{s_p - \mu_p}{\sigma_p}$$

After computing the z-score for each pattern $p \in \bigcap_{i=1}^{50} SM_i$, a pattern is deemed significant or overrepresented when its z-score exceeds the critical value of 1.96. This critical value corresponds to a significance level of 0.05, assuming a normal distribution. On the contrary, patterns with $z < -1.96$ are significantly more frequent in the randomized model's realizations, thus being under-represented or uncommon in the observed data. Analyzing such patterns can provide insights into the absence or suppressed occurrence of certain dynamics or relationships within the dataset. It is important to investigate both positive and negative z-scores to gain a comprehensive understanding of the patterns' significance and potential implications in the context of the analysis.

7.3.3 Mapping of temporal patterns across null model realizations

One challenge encountered during the computation of the z-score of patterns p was the lack of a canonical form for identifying the same pattern across the different outcomes of the GERM algorithm on the realizations of the null model. Indeed, the application of GERM on each realization generates patterns with their respective edge lists and supports, identified by independent incremental IDs. To overcome this issue and facilitate the comparison of results across multiple models, it is necessary to extract equivalence classes for each pattern, incorporate timestamp information, and assign a global ID to each temporal pattern. This process ensures consistency and allows for meaningful comparisons of patterns and their properties, even across different datasets. A preliminary observation that significantly contributes to reducing computational time is the frequent occurrence of patterns composed of the same set of edges within the realizations of the null model. Leveraging this observation, an initial step in the mapping procedure involves extracting the unique set of edge lists while preserving their respective occurrences. The pseudo-code of the procedure is reported in Algorithm 2. It requires as input the variable *germ*, which refers to the results of GERM on each realization of the MRRM (frequent patterns and their support). The algorithm returns a dictionary *edge_set* whose keys are the set of edges found in all the realizations, and the values are the list of occurrences of the key edge list in the form (p, m) with p being the original ID of the pattern and $m \in [1, 50]$ the ID of the realization it appears in.

Algorithm 2 General mapping: set of edges

```
Input: results of GERM on each realization of the null model $germ$
Output: $edges\_set$
1: $edges\_set = dict()$
2: $m = 0$
3: for $model \in germ$ do
4:     for $p, pattern \in model$ do
5:         push($edges\_set[pattern_{edges}]$, $(p, m)$)
6:     end for
7:     $m \leftarrow m + 1$
8: end for
```

However, there is still a chance that the edge lists in the dictionary keys include isomorphic subgraphs. To avoid redundancy we check the absence of isomorphic patterns, before assigning a global identifier to each pattern/edge list. Algorithm 3 describes the procedure: it initializes the i variable to the maximum existing key in the *mapping* dictionary. The reason for that is to create a global mapping for each dataset, so when processing the results we may have some patterns that have been already enumerated. We iterate on the edge lists and their occurrences collected by Algorithm 2. After initializing the temporary *id* to zero, we search in mapping if there is already a pattern isomorphic to the one associated with the edge list we are considering. If so, we assign *p* to *id*. Afterward, if after the search *id* is still zero (there is no isomorphic pattern in *mapping*), then the enumeration variable *i* gets increased, and assigned to *id*. At this point, *mapping* is updated with the new temporal isomorphism class *i*, with the subgraph $G(edges)$ obtained by the so far unseen edge list. At the end of each iteration of the outer loop on *edge_set*, the new data dictionary *new_shuffle_germ* is updated with the new global *id* and the same information as the original one.

Algorithm 3 General mapping: isomorphism check

```
Input: $edges\_set$
Output: $new\_shuffle\_germ$
 1: $mapping = dict()$
 2: $new\_shuffle\_germ = dict()$        ▷ Inspired by Python, dict() creates an
    associative map
 3: $i = max(mapping.keys)$
 4: for $edges, occurrences \in edges\_set$ do
 5:     $id \leftarrow 0$
 6:     for $p, pattern \in mapping$ do
 7:         if $G(edges)$ $is\_isomorphic$ $G(pattern)$ then
 8:             $id \leftarrow p$
 9:             $continue$
10:         end if
11:     end for
12:     if $id = 0$ then
13:         $i \leftarrow i + 1$
14:         $id \leftarrow i$
15:         $mapping[i] = G(edges)$
16:     end if
17:     for $p, m \in occurrences$ do
18:         $new\_shuffle\_germ[m][id] = germ[m][p]$
19:     end for
20: end for
```

7.3.4 Case studies and graph modeling

We applied our methodology (see Methodology section) to two different human-centered temporal datasets, described in Section I. The first one is DBLP, the bibliographic network representing co-authorship relationships, while the second one is UC-social, that encodes information about students of an American university that exchange messages on an internal platform. We used the modeling of DBLP network described in Section I, obtaining an undirected graph of 129073 nodes and 277081 edges, with 11 possible timestamps (year of publication). The second dataset, UC-Social, is modeled through a directed temporal graph, however, the GERM algorithm only handles undirected graphs. So, we processed the directed graph G_d to create a mutual undirected graph G_m: each edge $(u, v, t_1) \in G_d$ is inserted into G_m if and only if $(v, u, t_2) \in G_d$. In this case, the timestamp of $(u, v) \in G_m$ is $min(t_1, t_2)$. In this way, we obtain an undirected graph with 1280 nodes and 12916 edges, that reflects reciprocal relationships. To make the analysis more tractable, we aggregated the original timestamps of edges by weekly and monthly granularities. This allows us to track the mesoscopic evolution of the network over time without being overwhelmed by a large number of possible timestamps. With the aggregation, we obtain two different graphs UC-monthly and UC-weekly, having the same size and order, but with the former having a coarser time granularity. Precisely, the UC-monthly graph has only 7 possible timestamps, while UC-weekly has 28 possible timestamps.

It is worth noting that the two datasets offer different advantages: the first one provides a large temporal network, while the second one allows for arbitrarily tuning the time granularity.

7.4 Findings

In this section we analyze the main results, starting from a quantitative description of the rules found in both the real graphs and their corresponding null models. Then, we discuss the distribution of z-scores, and evaluate the impact of the patterns that were not included in the z-score computation. Finally, we also group the patterns by the time they take to form to gain a deeper understanding of the graph evolution.

7.4.1 GERM outcomes on real and randomized networks

GERM algorithm was applied to the three networks as well as to their timeline shuffled counterparts, setting a maximum of 4 edges (patterns involve at

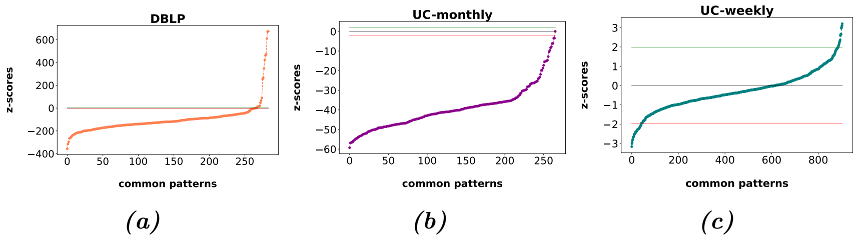


Figure 7.4: *Distribution of the z-scores of GERs extracted from (a) DBLP, (b) UC-monthly, and (c) UC-weekly. The thresholds above and below which patterns attain statistical significance are depicted by green and red horizontal lines, respectively.*

most four links) and generating 50 realizations of the null models. The minimum support values were selected starting from a value of 5000, which is the original choice of GERM's authors for the DBLP dataset. As concerns the UC networks, the support values were selected based on the minimum value needed to obtain non-empty outputs, starting from 5000. Table 7.1 reports the selected support thresholds and the corresponding number of rules identified. Specifically, the *GERM* column reports the number of rules found running the GERM algorithm with the aforementioned parameters on the original/real graphs, while the "*mean shuffle*" and "*union shuffle*" columns refer to the application of the algorithm on the timeline shuffled graphs (null model). The former indicates the average number of patterns found by executing the GERM algorithm on 50 randomized graphs. The latter (*union shuffle*) indicates the different rules/patterns found in the union of the 50 runs. The union is computed utilizing the general mapping algorithm described in the Methodology section.

dataset	**support**	**GERM**	**mean shuffle**	**union shuffle**
DBLP	5000	296	3795	3871
UC-monthly	150	266	1269	1378
UC-weekly	600	999	1039	1235

Table 7.1: *Overview of GERM outcomes on the three temporal networks. "GERM" column displays the number of rules identified in the real graphs. The "mean shuffle" column reports the average number of patterns over 50 timeline shuffled graphs. The "union shuffle" column presents the number of distinct rules discovered through the union of 50 realizations.*

Upon examining the results in Table 7.1, we observe that on average in null models there are more patterns than in the original graphs. Further, this difference is consistent across all 50 realizations of the randomized graphs, as we can note from the small difference between the average number of patterns (fourth column) and the union of the rules (fifth column). This observation indicates that in timeline shuffled realizations there are a consistently higher number of patterns having negligible support in the real temporal networks, i.e. timestamps and the ordering they induce in the real graphs impact the frequency of the evolution rules.

7.4.2 Analysis of z-scores

The application of a null model on the graph evolution rules enables the identification of significant rules specific to the temporal graph under consideration, extending beyond frequency-based measures. As described in the methodology section we compute the z-score for the patterns existing in all the realizations of the null model and in the real graph. Figure 7.4 shows the distribution of the z-score for the three temporal networks, where horizontal lines specify the noteworthy thresholds: the grey line stands for the zero level (patterns with identical support in both real and null graphs), while the green and red horizontal lines indicate the over and under-representation thresholds (± 1.96). While the under-representation of patterns is a consistent trend across all three datasets, there are notable differences in the distribution of z-scores across the three temporal networks. First, the UC-weekly case (see Figure 7.4c) is a special case because the majority of patterns (93.1%) are concentrated within the ± 1.96 region, meaning that their supports in the null or real graphs do not differ so much, i.e. they are not significative. In the other networks, almost every pattern is under-represented; for instance, in the DBLP case (see Figure 7.4a), the over-represented rules are only 16 against 264 under-represented. Second, in the DBLP case, z-scores present a very wide range of values, meaning that the supports of patterns in the real graph are extremely lower or higher than the ones in the null models. Third, in the UC-monthly network, all patterns are under-represented over the null model (see Figure 7.4b). We will investigate the underlying reasons in subsequent sections.

7.4.3 Frequency of GERs in real and randomized networks

While the common set of patterns obtained from GERM on the original graph and its randomized versions provides valuable insights into over and under-represented graph evolution rules, the patterns that are not in the intersection

WHERE IS FREQUENT	In G^*	Not in G^*
In all realizations of SM	🟩	🟥
In some realizations of SM	🟪	🟦
In any realizations of SM	🟪	⬜

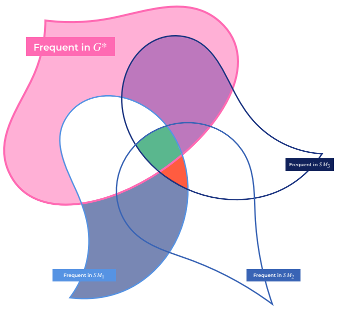


Figure 7.5: *Tabular and visual representation of possible cases a rule/pattern may be involved in. Here, we depict 3 realizations of the null model (SM). A pattern in the green set is a GER in the real network G^* and a GER in all the realizations, while if it is present in some realizations but not all, it is purple. If the pattern is not a GER in the real network - its support is below the threshold - it may belong to the red set, i.e. it is a GER in all the realizations, or it is in the blue set, i.e. it is a GER in some realizations but not all. Finally, patterns belonging to the pink case are GERs in the real network, but in all the realizations their support is below the threshold.*

DBLP (Frequent in GERM)

Frequent in how many realizations of the null model		Frequent	Not frequent	*TOT*
	Frequent in all	284	3431	**3715**
	Frequent in some	3	153	**156**
	Not frequent	9		
TOT	**3871**	**296**		

(a)

UC-monthly (Frequent in GERM)

Frequent in how many realizations of the null model		Frequent	Not frequent	*TOT*
	Frequent in all	266	871	**1137**
	Frequent in some	0	241	**241**
	Not frequent	0		
TOT	**1378**	**266**		

(b)

UC-weekly (Frequent in GERM)

Frequent in how many realization of the null model		Frequent	Not frequent	*TOT*
	Frequent in all	904	0	**904**
	Frequent in some	95	236	**331**
	Not frequent	0		
TOT	**1235**	**999**		

(c)

Figure 7.6: *Tabular representation of the different cases depicted in Figure 7.5, for the different temporal networks (a) DBLP, (b) UC-monthly, and (C) UC-weekly. Columns in each table indicate whether a GER extracted by GERM is frequent or not in the real graph. Rows indicate how frequently a GER has been extracted by GERM over the 50 realizations of the null model. Each record reports how many GERs belong to the specific case.*

of all runs of GERM over the realizations may be worth attention too. In fact, when running GERM algorithm on the real graph and on the 50 realizations of the null model, we obtain a set of frequent rules from each of the 51 runs, generating different scenarios depending on the frequency of a pattern in the real and randomized realizations. In Figure 7.5 we summarize all the possible scenarios combining a tabular representation and a set representation. According to the above representation, up to now, we have analyzed the patterns belonging to the green set, i.e. the set containing patterns appeared to be frequent in all realizations of the null models (SM_i) and even in the real graph (G^*). On the opposite side, we have the pink and red sets: the first one contains all patterns that were frequent only in the real graph (returned by GERM) but not in any randomized network: these are reasonably considered as over-represented patterns since the support in the null model is always lower than the real one. On the other hand, the patterns in the red set (frequent in all realizations of the null model but not on the real graph) are reasonably under-represented for the complementary reason, and so equally worthy of attention. Finally, since we generated many realizations of the null model, it is likely that some patterns are not always frequent (blue and purple sets). Still, if they are frequent in most of the 50 realizations, they may be worthy of analysis. In essence, by relaxing the initial conditional requirement, as required in the definition of the z-score, that restricts the evaluation of statistical significance solely to GERs extracted in all realizations of the null model ($p \in \bigcap_{i=1}^{50} SM_i$), we can broaden the analysis of significance to encompass a wider range of evolution rules.

We evaluate the extent to which patterns can be reintegrated into our analysis by examining the tables presented in Figure 7.6, which provide a comprehensive overview of the pattern distribution across various scenarios for the three temporal networks. The analysis reveals that the red and green sets are generally larger than the blue and purple sets, indicating that we can consistently enlarge the set of under-represented GERs in DBLP and UC-monthly. On the other hand, the pink set was nearly non-existent in all datasets, so the extension of over-represented patterns is very marginal. Furthermore, the size of the purple set is found to be smaller than the blue set, suggesting that patterns that are not frequent in all the realization of the null model, are probably uncommon in the real graph as well. In general, the expansion of assessable GERs aligns with the trend observed in the analysis of the z-scores, wherein a majority of the evolutionary rules demonstrate under-representation, particularly within the DBLP and UC-monthly datasets. The notable prevalence of under-represented GERs underscores how temporal constraints and/or evolutionary mechanisms within the three temporal networks may hinder the manifestation of certain evolution rules. These factors are loosened in the realizations of the null model.

Finally, we look at the support of the patterns in each set and investigate any similarities or differences that may emerge. Figure 7.7 depicts the support distributions of each set for all three networks. While not all datasets contain all five sets, it is evident that the supports for the purple and blue sets are rather low, particularly when compared to the green and/or red sets. This last finding suggests that patterns that are not frequent in all the realizations of the null model may not be worthy of attention, given that they only marginally exceed the minimum support threshold. Therefore, we do not consider the purple and blue sets in further analysis. On the other hand, the pink set, which is exclusive to real networks, represents a small but significant set of patterns that cannot be detected in the realizations of the null models.

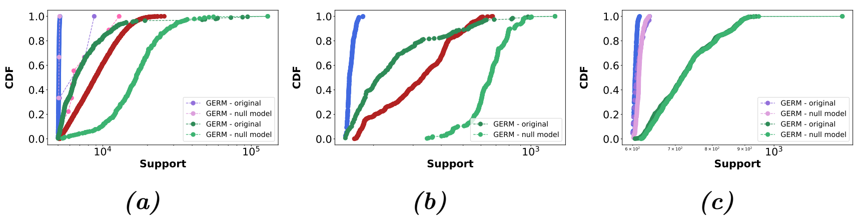


Figure 7.7: *Distribution of the support of rules/patterns grouped by different cases described in the Figure 7.5. As for the support of the GERs in the realizations of the null model, we report the distribution of the mean support. The x-axis has a logarithmic scale.*

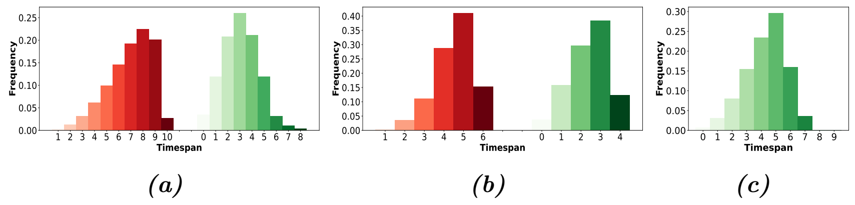


Figure 7.8: *Distribution (frequency) of the timespans of the evolutions rules in the three temporal networks (a) DBLP, (b) UC-monthly, and (c)UC-weekly. We grouped timestamps according to the class their pattern belongs to. The palettes of histograms recall the colors assigned to each class in Figure 7.5.*

7.4.4 Analysis of timespans

A further advantage of GERM is that the returned GERs are provided with the time the body takes to form, i.e. its timespan *tspan*. Indeed, the timespan of a

GER corresponds to the maximum timestamp in the body of the rule. We focus on the maximum timestamp because we consider relative-temporal patterns, with the minimum timestamp always set to zero. In the context of statistically significant GERs this information is important since we can discover signals that certain constraints or evolution mechanisms may favor or contrast slow or fast formation of specific subgraphs. Figure 7.8 shows the frequency distribution of the timespans of the rules in the three datasets, grouped according to the set a pattern belongs to, namely the red and green sets. We observe that GERs within the green sets exhibit relatively lower timespans, typically around 3, indicating their prevalence within shorter temporal intervals and a relatively fast formation process. On the other hand, the red sets include patterns characterized by higher timespans, suggesting their persistence and prevalence over longer durations, and a slower formation process. This disparity in timespan distribution between the green and red sets provides valuable insights into the temporal dynamics within the analyzed datasets, suggesting that in the real graphs the frequent patterns are the ones that happen in a shorter time interval. In fact, the higher timespan patterns are more common in the null model but not in the real graph. The distinction in terms of timespan between the green and red sets is particularly evident when considering the support of these patterns. For instance, a significant majority (around 89%) of the over-represented patterns in the DBLP graph exhibit a $tspan < 3$, while the 65% of most frequent patterns in the red sets present a $tspan \geq 4$. This observation provides additional support for the hypothesis that various factors and mechanisms influence the rapid formation of specific subgraphs, while other factors act against the formation of certain subgraphs, thereby slowing down the overall formation process.

7.4.5 Discussion

Finally, we illustrate a few examples showing the enhanced value provided by the null model extension on the graph evolution rule algorithm. By analyzing these patterns, we aim to show how the assessment of the significance can offer valuable insights beyond the outcomes traditional frequency-based mining algorithms provide.

In the case of the DBLP temporal network, the introduction of the null model reveals patterns with more intricate structures than usual chains or triangles. For instance, the GER in Figure 7.9a is resulted to be significative, meaning that its support in the real graph is significantly more frequent than in the null model. However, in terms of support rank, it falls beyond the 100^{th} position. Without the null model, such a pattern might have been overlooked

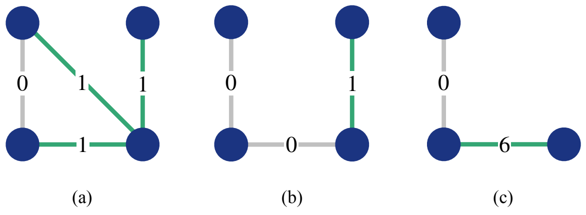


Figure 7.9: *Example of GERs extracted from the DBLP temporal network. Edges in green belong to the post-condition only.*

in a semantic analysis, yet it holds important information about the graph's evolution. Indeed, this GER represents two authors who start their collaboration at a specific time (t_0) and subsequently collaborate with a common co-author the following year, the target of a further collaboration with another author external to the initial pair.

On the other hand, the GER in Figure 7.9b exemplifies a pattern with high support (14^{th} position) but a remarkably low z-score (-216). This means that its having a high support may not mean that is actually a pattern worth attention in explaining the dynamics of the graph, as its support becomes substantially higher when the network timestamps are shuffled. Without the null model extension, we might have considered the dynamic where a triangle between three authors does not close in the subsequent year as important, while in reality, it may not hold such significance.

Another observation could be drawn by looking at the lower tail of the z-score distribution: Figure 7.9c depicts a rule that has one of the lowest z-scores (-257.7) that also has a large timespan (6 years). This is telling us that this kind of pattern presents a really higher support in the models, suggesting that an author that starts a collaboration at a certain t_0, may start another one in 6 years. However, this occurrence is not as common as in a random graph where temporal dependencies are loosened. Therefore, it represents a common behavior that could be even more frequent whereas temporal constraints or other mechanisms would not come into play.

By presenting these diverse examples, we aim to underscore the added value of the null model extension in capturing nuanced dynamics during the evolution of temporal networks. These examples showcase how the introduction of the null model framework enables us to uncover patterns that possess distinctive characteristics, highlighting their significance in understanding network evolution.

7.5 Conclusions

In this study, we disentangle the evolution of different temporal networks by the identification of statistically significant graph evolution rules The assessment of the statistical significance results from the introduction of a proper null model applied to the GERM algorithm, the first and most stable method for mining graph evolution rules. The null model preserves the static structure of the graph while shuffling timestamps, ensuring the temporal distribution is maintained and introducing randomness to the sequence of events. By employing a z-score test, statistically significant rules that deviate significantly from the null model are identified. Although the significance of the identified GER has been almost neglected, our findings show that the introduction of a null model impacts the evaluation and interpretation of rules. First, a few highly frequent rules are not significant at all, only a few are over-represented, while the majority of the GERs are under-represented. So, by shuffling timestamps, we weaken or remove specific temporal factors or mechanisms that may inhibit or favor evolution paths. Furthermore, we also extended this observation to the speed of the formation process of subgraphs, where rules expressing fast formations are over-represented. As a future extension of this work, we plan to identify the mechanisms and the factors acting on the over and under-representation of the GERs and assess their role in the dynamics of subgraph formation.

Chapter 8

Profiling Web3

8.1 Introduction

Web3, one of the novel paradigms which may drive the evolution of the future Web, is offering an invaluable volume of data stored in the supporting blockchains. Researchers from different fields such as network science, computational social science and data mining, might benefit from these large collections of temporal and heterogeneous data capturing different kinds of interaction among people and between people and the platforms. In this study we focus on a specific issue related to these modern techno-social systems, i.e. the understanding of the rules driving their growth. To reach this goal, we performed an analysis based on graph evolution rules - GERs - on different networks gathered from Web3 platforms such as Steemit or OpenSea. Graph evolution rules mining is a frequency-based method for evaluating network evolution which does not require any prior growth process for disentangling how networks evolve. By comparing the evolution rules of social network platforms and asset trading services through GER profiles, we observe that some evolution rules are common to all Web3 platforms, regardless of the system specificity. On the other hand, in specific cases, the frequency of graph evolution rules is influenced by the nature of the platform: whereas social and token-transfer networks are characterized by rules which increase network transitivity and reciprocity, NFT trading networks, especially those specialized in a specific type of digital asset, are driven by rules which form trading chains. These findings suggest that the GER approach and the GER profiles are a good starting point to get insights into the evolutionary behavior of a network and to define a classification of graph evolution rules.

8.2 Background and related works

The Web3 paradigm is quite a new framework in the Web landscape, especially as far as regards the aspects directly related to the blockchain technology supporting platforms and services. To introduce this paradigm, we provide the reader a brief overview of the Web3 platforms we treated in our analysis: blockchain online social networks and NTF trades.

8.2.1 Blockchain online social networks

Blockchain technology has enabled the development of blockchain online social networks (BOSNs), providing data storage and validation for these platforms. In their core BOSNs replicate the main user experience of the main micro-blogging and social media platforms such as Twitter, Reddit or Medium, but they introduce token-economy aspects, such as a reward system based on cryptocurrencies that promote high-quality content. In fact, in these systems cryptocurrencies can be created, exchanged, and used for validating both social operations (follow, vote, comment) and economical transactions (transfer, borrow tokens).

In this context, one of the most attractive and spread platforms is Steemit [110]. Steemit is a blockchain social network launched in March 2016, hosted on the Steem blockchain. Steemit users can exchange goods and services using the dedicated cryptocurrency, called STEEM. Furthermore, the cryptocurrency powers a reward system that encourages network growth by compensating users for their participation on the platform. Web3 platforms such as Steemit offer a rich data source for understanding the system's dynamics and the networked structure of its components, so much so that the literature about BOSNs analysis is growing. Some works leverage user content for bot detection [111] or text mining tasks [112]. Other works focus on the relationship between blockchain technology and social networks [113, 114, 115]. For example, Chonan [116] and Kim *et al.* [105] have analyzed the social network structure of the Steemit platform, while Guidi *et al.* [117] have studied the graph of follow operations, and then focus on other operation types [118]. When studying dynamic systems BOSNs, temporal information plays an essential role, so it is important to model the data as dynamic graphs and study its temporal aspects. For example, Ba *et al.* [119] have studied how cryptocurrency and graph evolution are related to each other. The same authors have also conducted an analysis on the network burstiness [120], focusing on the link creation process and the claiming of rewards. Finally, the interplay between social and economical network layers has been investigated in [121] to cope with user migration across Web3 platforms.

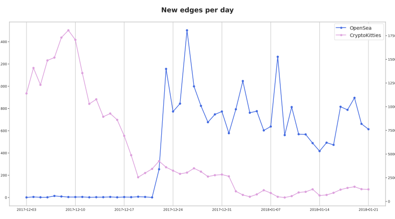
8.2.2 Non-fungible tokens - NFTs

An NFT is a blockchain-based data unit with a double goal: first, it provides a unique certificate of ownership of a digital object. Second, it attests to the uniqueness and non-transferability of a digital asset. Thanks to this technology, it is possible to track down the complete history of ownership of an object and check its authenticity. In concrete terms, an NFT can represent a variety of digital items, including photographs, movies, and audio. As a consequence, several contexts, such as art, gaming, and sports collectibles, utilize NFTs to regulate and control digital objects. The birth of the NFT market can be traced back to late 2017 when the blockchain game Cryptokitties gained popularity. However, the market remained dominated only by Crypokitties until July 2020 when it started to grow and in March 2021 reached a peak of popularity, due to the selling of an artwork's NFT for $69.3 million. This purchase allowed the author, Beeple, to reach one of the highest auction prices for a living artist.

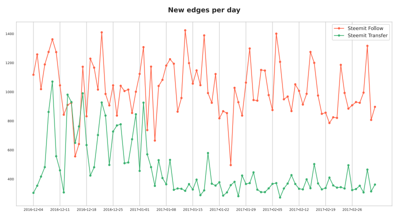
The peculiar growth history of the NFTs market can explain why the literature on them is currently in rapid growth. Nadini *et al.* [104] conducted the first comprehensive quantitative overview of the NFTs market, including the overall statistical properties, its evolution over time, a network-based analysis, and a study about the predictability of NFT sales. Other works present a more focalized analysis, for example, Vasan *et al.* [122] analyzed the cryptoart ecosystem, while Franceschet [123] focused on the creators-collectors network. Other research studied the role of social media attention on NFT trends [124], and the financial advantage that experienced users gain in the NFT trading context [125].

8.3 Methodology

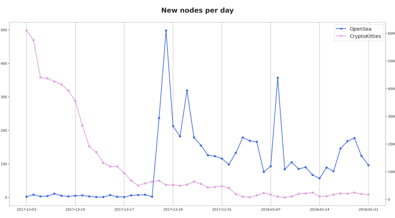
To analyze the evolution of the Web3 networks, we applied a graph evolution rules algorithm called *EvoMine* [35]. In the next sections, we are going to define how we model transactions and social operations into a graph representation. Then, we detail the method to extract and compare the evolutionary profiles. Details about the EvoMine algorithm can be found in Chapter 4 while a brief overview of graph evolution rules literature can be found in the previous chapter (Chapter 7).



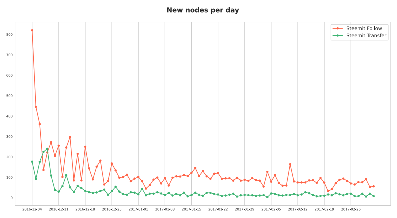
(*a*)



(*b*)



(*c*)



(*d*)

Figure 8.1: Number of daily new nodes and edges. *Plots on the same column represent the same dataset (NFTs - first column, Steemit - second column), while plots on the same row represent either new edges or new nodes. In (a) and (b) the plots show how the number of new edges on a daily basis, respectively in the NFTs and Steemit datasets. On the other hand, (c) and (d) depict the number of new nodes per day, respectively in the NFTs and Steemit datasets.*

8.3.1 Representation and modeling

We model the transactional data gathered from Web3 platforms into directed, temporal graphs. All the four different datasets share the same operation structure: every transaction is a tuple (s, d, t), composed of a source s that performs an operation (follow, money transfer, or NFT exchange) towards a destination d at timestamp t. Following the taxonomy proposed in Chapter 3, we modeled the data into $G_[(1, T)]^{GP} = (V, E, f)$, where f is an edge-labeling function assigning to each edge (i, j) the first time user i had an interaction with user j.

Case Studies We conducted our analysis on datasets that represent the two main trends in Web3 platforms. On one side we deal with the blockchain online social network Steemit, an example of converting the online services of the current social media into applications for the Web3 world. On the other side, we analyzed an example of platforms and assets which are peculiar to the Web3 paradigm, since they required characteristics of the blockchain technology: NFTs. Both datasets are described in Chapter 6. Here, we detail the specific subsets of the datasets we used in this work. As for the Steemit platform, we fix the period from December, 1 2016 to March, 1 2017, focusing on two specific types of operations: *follow* and *transfer*, which represent respectively the most common type of social and financial operations. Applying the modeling described above, we obtain a follow graph with 11004 nodes and 92803 edges, and a transfer one with 2815 nodes and 42452 edges. For the NFT dataset, we focused on two key markets: CryptoKitties and OpenSea. We chose CryptoKitties as it was the first major NFT project, and OpenSea because it is the largest NFT marketplace. Our analysis concentrated on the initial transaction periods for each platform: December 1, 2017 to January 19, 2018 for CryptoKitties, and February 4, 2018 to March 26, 2018 for OpenSea. With the modeling cited above, the cryptokitties graph presents 58906 nodes and 255947 edges while the OpenSea one has 4870 nodes and 23251 edges.

As a first step to understanding the evolution of these networks, we observe the number of daily new nodes and edges, depicted in the plots of Figure 8.1. Specifically, Figure 8.1a and Figure 8.1b show how the number of new edges changes over time; while Figure 8.1c and Figure 8.1d depict the growth of emerging nodes. The plot related to the NFT sales datasets (8.1a) highlights a change in popularity of the two markets: Cryptokitties has an initial peak, but then the number of new edges rapidly decreases, on the other hand, OpenSea presents the opposite behavior: a fast increase of the activities after the middle of December 2017. The same observations stand for the trend of daily new nodes, depicted in Figure 8.1c. As regards the Steemit follow (social) and transfer graphs, they share a common trait: even if values are lower in the case of the transfer graph, the two networks show a similar trend in the number of new nodes, which reaches a certain degree of stability after initial oscillations. A difference between the evolution of the graphs emerges when observing Figure 8.1b, approximately after one month, the number of new transfer edges starts a decreasing trend, while the trend of the follow graph is characterized by a higher volume of operations while keeping wide fluctuations.

8.3.2 GER Profiles

The goal of this study is to give a thorough analysis of the evolution rules obtained, rather than just focusing on numerical observations (like the number of rules found). To do so, we define a vector-based representation of the graph evolution rules by which we can summarize the evolutionary behavior of a network. The vector representation is called *GER profile*. This vector indicates the distribution of each kind of rule, so we first identify all the temporal subgraph isomorphism classes. Note that we worked on the union graphs of the resulting rules, so the isomorphism classes consider the topological structure but also the temporal information. After the identification of all the subgraph isomorphism classes, the vector $v(a)$ is computed for each application a of the EvoMine algorithm. Specifically, each element of the GER profile is defined as follows:

$$v_i = \frac{\sigma_{event}(r_i)}{\sum_{j=1}^{n} \sigma_{event}(r_j)} \tag{8.1}$$

where $\sigma_{event}(r_i)$ is the event-based support of the rule r_i and n is the number of distinct GERs identified by EvoMine. Given a temporal network, its GER profile represents a footprint of its evolution as well as a compact representation of its growth.

Distance. As an application of the GER profiles, we can exploit them to assess how the growths of two different networks follow similar evolution rules. It is possible since the GER profile is essentially a probability distribution over the space of the graph evolution rules. In this case, to measure how dissimilar the distributions are, we compute a pairwise distance for all the applications, i.e. for all the temporal networks gathered from Web3 platforms. We use the *Wasserstein* distance [126], also known as *Kantorovich–Rubinstein metric* or *Earth mover's distance*. The last name is related to the analogy that sees each distribution as a unit amount of earth and the metric as the minimum cost of turning one pile into the other (amount of earth that needs to be moved multiplied by the mean distance). Formally, the Wasserstein distance W_p of two distributions u, v is defined as follows:

$$W_p(u,v) = \left(\inf_{\pi \in \Gamma(u,v)} \int_{\mathbb{R}^d \times \mathbb{R}^d} \|x - y\|^p d\pi \right)^{\frac{1}{p}}$$

where $\Gamma(u,v)$ is the set of all joint probability measures on $\mathbb{R}^d \times \mathbb{R}^d$ whose marginals are u, v.

8.4 Findings

We apply the EvoMine algorithm described in Chapter 4 to the datasets described in the previous Section. In this section we analyze the results, giving a quantitative description of the graph evolution rules found, studying the results on the single datasets, and then comparing their GER profiles to highlight common aspects and differences.

8.4.1 Quantitative descriptions of results

EvoMine was applied to the four datasets specifying the directed nature of the graphs, a maximum number of 3 edges per evolution rule, and the absence of edge/node colors/labels. As regards the minimum support of patterns, we tried different steps of support, starting by $s = 150000$ and then decreasing until reaching a non-empty output. The chosen support values are shown in Table 8.1, that report also the number of rules returned by the algorithm. Note that the output has been filtered in order to obtain just *meaningful rules*, namely, rules whose union graphs present edges with two distinct timestamps, so that they really describe an evolution or a growth process.

Table 8.1: *Support and number of rules obtained for each network.*

Graph	**Support**	**Number of GERs**
Steemit Follow	50000	23
Steemit Transfer	30000	22
NFT Cryptokitties	150000	12
NFT OpenSea	10000	21

From this pure numerical observation of the number of rules obtained, we can observe that the dataset about Cryptokitties NFT market stands out with respect to the others with a lower value of rules. In general, given the thresholds chosen for the support, the number of rules describing the evolution of the Web3 networks is relatively low.

We investigate more about the difference in the different outputs by analyzing the intersections of the four sets of rules. Figure 8.2 shows a graphic representation of the four sets of results, divided into two Venn diagrams to give a more intuitive idea. From the diagram on the left, we deduct that Cryptokitties results set is a subset of OpenSea one, that recursively is a subset of the Steemit Follow network. However, Steemit Transfer - a transfer network - shares almost the entire output with OpenSea - a trading network, except for one pattern that is not present in Steemit Follow either. Figure 8.2 also illustrates the patterns in the difference sets between Steemit Follow and Steemit Transfer, i.e. ($steemit\_follow \setminus steemit\_transfer$), and ($steemit\_transfer \setminus steemit\_follow$). Note that in this work, we represent graph evolution rules as a unique temporal pattern, discerning the timestamps of edges with a gray (for timestamp t_0) or green (for timestamp t_1) color. From the figure, we can observe that the rules present only in the Steemit Follow

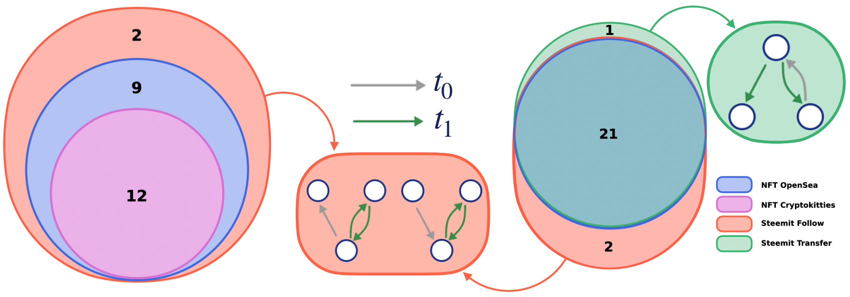


Figure 8.2: *Venn diagrams that show how the sets of graph evolution rules found in the four different graphs intersect with each other. From the left, the diagram shows how Cryptokitties results are a subset of OpenSea ones that in turn are completely covered by Steemit follow output. The graphic in the middle shows the two evolution rules that are present only in the Steemit follow set. The following diagram depicts the relation between OpenSea, Steemit transfer and Steemit follow. Finally, the rule on the right represents the one present only in the Steemit transfer set.*

graph (GERs in the orange rounded rectangle) suggest an instant reciprocal behavior, while the pattern in the Steemit Transfer graph (GER in the right-most green circle) can represent an expansion-oriented pattern. For a more detailed explanation of the rule interpretation see the Discussion Section.

8.4.2 GER Profiles

We apply the method described in the Methodology Section to get values that can measure the differences between evolutionary behaviors in the four different datasets. We apply the Wasserstein distance between pairs of GER profiles, obtaining the values shown in the distance matrix depicted in Figure 8.3. First, the values are generally low, meaning that the distribution over the different kinds of rules is rather homogeneous. This suggests a first finding about the overall trait of the evolution processes characterizing Web3 networks: the types and the frequency of the evolution rules are quite uniform across the platforms. That indicates that in our set of Web3 platforms, there is not a manifest outlier that is driven by special evolution rules. Second, a more specific analysis of the distance matrix shows that Cryptokitties network is the one that differs the most from the other ones, a further insight that has deserved a more detailed discussion in the next Section.

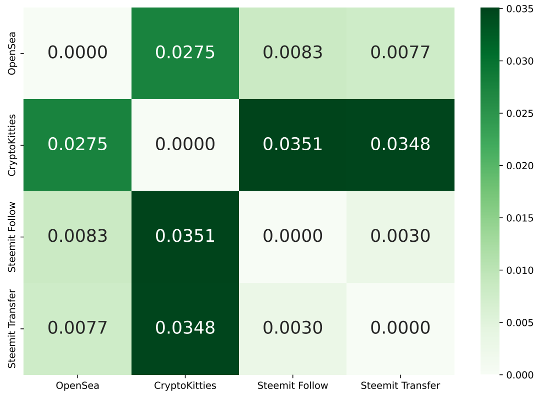


Figure 8.3: *Wasserstein distance matrix between GER profiles of each graph. The distance matrix has been depicted as a heatmap, where the green intensity is proportional to the distance.*

8.5 Discussion

In this section, we are going to propose semantic interpretations of some rules, starting from the distribution given by GER profiles described in the Methodology Section. Our discussion mainly relies on Figure 8.4, which represents a graphical representation of the GER profiles of the four temporal networks. The first evident feature concerns the distance from Cryptokitties GER profile with respect to others, which confirms the results highlighted by the computation of the Wasserstein distance. The other three vectors, especially the Steemit ones, are very similar, with a few exceptions. In the following paragraph, we deepen these differences and we propose a graphical representation of the patterns that correspond to the indexes where the distributions present noticeable differences.

A first evident difference concerns the first rule. In fact, rule 0 is not present in the Cryptokitties results set, and it corresponds to the closed triangle depicted in Figure 8.5c and in Figure 7.1. This rule expresses the classical triadic closure process typical of social networks where in the body we have an open directed triad and in the head the formation of link between the extremes of the open triad closes the triad, forming a directed triangle. Its absence can be explained by the nature of the network, i.e a trade network, and by the fact that only a single type of NFT can be exchanged on this market, i.e. cats. In fact, if there is only one type of object to sell/buy, it is very uncommon to create a closed directed triangle of sales. On the contrary, it is more likely to create chains of sell operations or to have expansion-oriented behaviors, i.e. an account buys more digital assets of the same type from different sellers. This intuition is confirmed by the higher values associated with rules from 8 to 13, they all embed an expansion mechanism of the source node (node with zero in-degree). Figure 8.5f shows an example of this kind of expansion, where the most left node, first creates a link towards the bottom node, and in the next timestamp (green arrow), it expands to a third node, that in turn expands to a fourth node. In this case, the rule indicates that a new chain is creating in a new direction starting from the source node. As for the triadic closure rule (Figure 8.5c), it is worth noting that, while in a social network as Steemit Follow it is a quite expected rule, in trade and transfer networks such as Steemit Transfer and NFT OpenSea is an unexpected trait which, especially in a transfer network, may deserve further investigations as it might be linked to malicious actions.

Even rules 6, 7, corresponding to Figure 8.5d-e, present support equal to zero in the Cryptokitties scenario. Here, the head of the rule is chain of transfers between wallets or users which originates from a single link (gray arrow) making the resulting chain. The absence of this type of chains might be related to a problem in the time granularity. Note that the graphs are built aggregating all transactions performed on the same day, and EvoMine algorithm can only catch evolution rules between consecutive timestamps. So, the evolution rules discovered highlight the evolution that happens on consecutive days. These chains may happen and be frequent in the graph but within the same day or on non-consecutive days. In fact, rules 6 and 7 generate chains of sell actions, which are likely in trade networks but they may actualize in more than a day, especially when there is only a type of object to be sold.

Finally, rule 15 marks a checkpoint from which the distribution of all graphs but Cryptokitties lose their common trend, up to rule 20. For example, rules 17 and 20 (respectively Figure 8.5a-b) reflect an instant reciprocal behavior, because in both cases there is an initial link between two nodes (gray edge), and one of them creates a link with another node, that reciprocates it in the same timestamp (two green arrows). The cited rules are not frequent in the three economical/trading networks (NFTs sales and Steemit Transfer), but are present in the most common rules for the only social network considered (Steemit Follow). This suggests that reciprocal behavior is less common in transfer and economical networks with respect to social networks, especially considering the daily granularity of the outcomes.

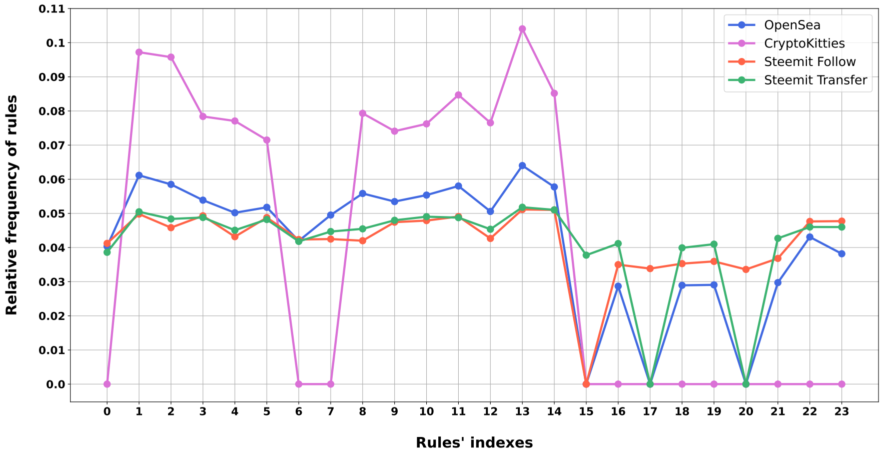


Figure 8.4: GER profiles for each Web3 network. *The plot shows, for each graph, the relative frequency of each kind of rule identified. Specifically, the x-axis specifies the rules indexes (from 0 to 23), while on the y-axis the relative frequency of each rule over the entire graph is specified. In the Discussion Section we comment on rules with indexes 0, 6, 7, from 8 to 13, 17, and 20.*

8.6 Conclusions

Blockchain-based platforms and online services are the backbone of the Web3 paradigm, one of the candidates for guiding the evolution of the future Web. Given the important role Web3 platforms might have in the future, it is crucial to understand which are their specific properties, how people behave within these platforms, and how design principles inspired by decentralization and token-based economy may influence how people interact with each other and with the platform functionalities. In the case of Web3 paradigm, researchers may be facilitated in coping with these issues, since the underlying blockchains publicly offer a large volume of temporal and heterogeneous data capturing interactions that occur in these techno-social systems. In this work we have dealt with a few types of the temporal networks generated by Web3 platforms by disentangling their rapid growth. Our methodology is based on methods rooted in frequent subgraph mining. Specifically, by the state-of-art algorithm EvoMine, we identify the most frequent graph evolution rules which capture the essential paths of growth of different blockchain-based platforms. In fact, methods based on subgraph counting are mechanism-agnostic, i.e. they do not make any assumption on the process generating the links, and return human-

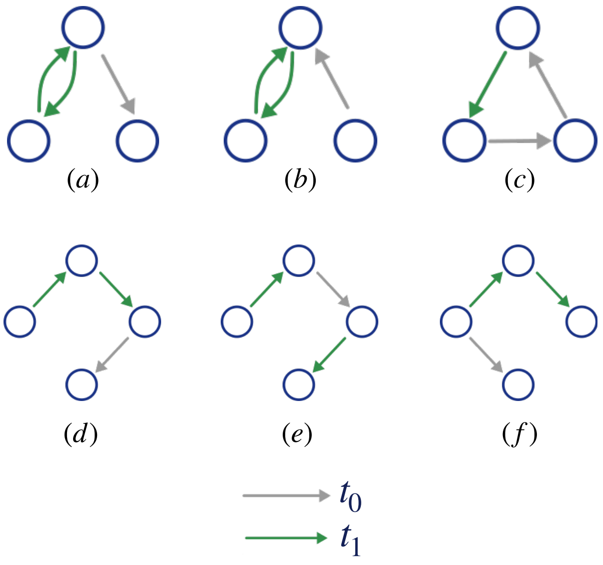


Figure 8.5: *Examples of graph evolution rules found. Each graph is a condensed representation of a graph evolution rule, where the gray edges belong to the body of the rule (precondition) and the green edges belong to the head (postcondition).*

readable and explainable description of the network evolution, w.r.t. methods for dynamic graph representation learning. By comparing the evolution rules of social network platforms and asset trading services, we observe that GER profiles - a vector-based representation of the network evolution - are able to identify evolution mechanisms strictly related to the nature of every single platform: whereas social and token-transfer networks are characterized by rules which increase network transitivity and reciprocity, NFT trading networks, especially those specialized on a specific type of digital asset, are driven by rules which form trading chains or expand node neighborhood. From this perspective, an approach based on GER profiles may be adopted to characterize the nature of new Web3 networks, so to identify which kind of network we are observing.

The findings and the methodology presented in this work open up a few research directions which might be explored in future works. From a methodological viewpoint, the graph evolution rules returned by EvoMine are constrained to the choice of the cut-point timestamp, making the tuning of this parameter an important element for correctly identifying significant evolution rules when the link formation process is not stationary. Moreover, most of the methods for extraction of evolution rules do not provide a statistical significance of the outcomes. So, in this context, a definition of a proper null model is mandatory to evaluate the significance of the rules. On the other hand, as for the characterization of the growth of Web3 platforms, future research directions may regard the creation of an extensive dataset repository collecting temporal and heterogeneous networks from blockchain-based platforms, or a special focus on the stationarity of the evolution rules along with the entire growth of the networks.

Moreover, results suggest that the methodology explained can be leveraged in models that aim at studying the evolutionary behaviors of dynamic networks. For instance, graph evolution rules can be embedded to predict how a network will evolve or can be adopted to inform data-driven models for network evolution. Another possible employment of GERs could concern the identification of change points in the temporal version of the GER profile so as to identify whether changes in the growth dynamics and in the mechanisms leading it are occurring.

Chapter 9

Temporal node evolutionary representation

9.1 Introduction

Studying real-world dynamic networks and their evolution is crucial for understanding the complex systems that govern various domains, from social interactions to financial transactions. The evolution of these networks provides insights into the underlying mechanisms driving their changes, which can be pivotal for applications such as node segmentation, prediction of future states, and role discovery. Among the various approaches to studying network evolution, graph evolution rules (GERs) stand out since they produce human-readable outcomes without requiring any pre-assumptions about the underlying evolutionary mechanisms. In this work, we leverage GER to derive evolutionary node profiles (NEPs), capturing the distinct patterns of how nodes change over time within the network. These profiles allow us to identify groups of accounts characterized by similar evolution rules, revealing common interaction patterns. As a case study, we apply our approach to Sarafu, a complementary currency platform with rich temporal data, representing a contemporary human complex system that integrates humanitarian aid, collaboration, and financial aspects. By analyzing Sarafu's network using our GER-based method, we identify two distinct evolutionary traits, uncovering significant behaviors that contribute to the platform's operation. Our findings suggest the effectiveness of using graph evolution rules in real-world dynamic networks, showcasing their potential to enhance our understanding of the node-level dynamics of complex systems.

9.2 Related work and background

In this section we summarize the related works on networks and node representation based on (temporal) subgraphs as well as works about our case study: Sarafu, the complementary and humanitarian aid crypto-currency. For details on the frequent graph mining approach we use to characterize the dynamics of ego-networks (Evomine) and the broader literature please refer to Chapter 4 (a shorter version can also be found in Chapter 7).

Node representation based on (temporal) subgraphs. Recent advancements in node vectorial representation have leveraged frequent subgraphs, motifs, and graphlets to enhance the richness of temporal graph embeddings. One notable approach is the Neural Temporal Walks (NeurTWs) [127], which leverages structural and tree traversal properties along with time constraints to capture dynamic patterns in temporal graphs. This method allows for an effective characterization of temporal nodes through representative motifs. Another prominent study embeds nodes based on their structural roles within the network, providing versatile representations for dynamic and evolving graphs [128]. On the other side, among the not-neural approaches, Hulovatyy *et al.* [58] proposed a vectorial representation of nodes using dynamic graphlets. This method adopts a common approach in this context which is to decompose networks into smaller segments [129, 130, 82] to characterize node behavior over time. Along this line, a strategy proposed by Longa *et al.* [60, 131], suggests adopting an egocentric perspective. This method tracks the evolution of node neighborhoods across temporal layers, collecting egocentric temporal subgraphs at each time step and condensing them into egocentric temporal motifs (ETMs), facilitating efficient identification of recurring interaction patterns in dynamics contexts through comparison against a null model.

Complementary currency. Complementary currencies (CCs) are alternative currencies that supplement national currencies in various geographic contexts. Viewed as fungible vouchers redeemable for goods and services, there have been 3,500 to 4,500 CC projects in over 50 countries since the 1980s. Among these projects, Sarafu is a complementary currency on a blockchain created by the Grassroots Economics (GE) Foundation. Users make payments via mobile phones, transferring Sarafu tokens to other registered users. During the COVID-19 pandemic, the Kenyan Red Cross used Sarafu to distribute humanitarian aid, with new users receiving free tokens backed by donor funds. Sarafu has been the subject of several studies since the GE Foundation provided an anonymized dataset of user transactions spanning a year and a half.

For instance, a dataset paper offering context and background of the platform has been provided by Mattsson *et al.* [132], while Ussher *et al.* [100] analyzed the dataset and the Sarafu project's history. Mqamelo [133] studied the impact on local economic engagement, and Mattsson *et al.* [134] modeled money circulation within Sarafu's network. Finally, Ba *et al.* [135, 136] analyzed cooperation behaviors within the Sarafu network, highlighting cooperation patterns, the significance of group accounts, and the role of the geographical positions of accounts.

9.3 Methodology

From a node-centric perspective, our main aim is to represent nodes based on the mechanisms that characterize the evolution of the interactions surrounding them. The methodology to get this kind of representation is based on two main tasks: *a)* the extraction of the ego-networks from the overall temporal network describing the system, i.e. interactions surrounding every single node; and *b)* the identification of the mechanisms/rules driving the evolution of each ego-network through the computation of the graph evolution rules. In this section, we detail these two main tasks and propose a vector-based representation for nodes, rooted in graph evolution rules, namely the *node evolutionary profile - NEP*. See Section 4 for details about EvoMine, the GER algorithm selected for this work.

9.3.1 Ego-networks from temporal networks

In this work, we model the set of interactions or transactions among the members of a networked system following the definition of *temporal network* $\mathcal{G} = (V, E)$ proposed in [137, 21], where:

- V is the set of users in the system; and
- $E = \{(u, v, t) \mid t \in [1, T], (u, v) \in V \times V\}$ is a set of timestamped directed links (u, v, t). Each link corresponds to an interaction/transaction from node u to user v that occurs at time t.

To accomplish the first task, we extract the temporal ego-network from the temporal graph $\mathcal{G}$. For each node u, its ego-network $S(u) = (V_u, E_u)$ corresponds to the temporal subgraph induced by u's neighborhood, including u itself. Formally, the set of nodes is defined as $V_u = u \cup N(u)$ where $N(u)$ is the neighborhood of node u. The set of edges is defined as $E_u = \{(v, w, t) \mid (v, w, t) \in E, v \in V_u, w \in V_u\}$, i.e. all temporal links whose endpoints are in V_u.

9.3.2 Node Evolutionary Profile

In static and temporal networks the distribution of measures based on static and/or temporal subgraphs of different order and size has been used for encapsulating the static and dynamical signature from both a network- and node-level perspectives. In this sense, the graphlet-degree vector in [138] and its extension to the temporal setting given by the dynamic graphlet degree vector [58] represent some of the extents to derive a network or node representation including the (temporal) subgraphs a node is involved in. Here we proposed a similar representation for nodes which relies on the graph evolution rules characterizing the evolution of nodes' ego-network.

We denote the vector representation as *Node Evolutionary Profile - NEP* and it represents the distribution of the graph evolution rules for the ego-network $S(u)$ of the node u. The construction of the node evolutionary profile is based on the graph evolution rules and their supports computed by EvoMine on each ego-network $S(u)$; while a vector representation common to all nodes is supported by the unique and common identifiers for rules based on canonical form. Specifically, each element of the Node Evolutionary Profile $nep(u)$ is defined as follows:

$$nep(u)_i = \frac{\sigma_{event}(r_i)}{\sum_{j=1}^{n} \sigma_{event}(r_j)} \tag{9.1}$$

where $\sigma_{event}(r_i)$ is the event-based support of the rule r_i in the u's ego-network and n is the number of distinct GERs identified by EvoMine on the whole set of nodes. In short, given an ego-network of a node u, its NEP represents a signature of its evolution as well as a compact representation based on the dynamics of the interactions among the neighbors of u and with the neighbors and u.

9.3.3 Case study

As a case study, we leverage Sarafu, described in Section 6, that encompasses a total of 412,050 economic transactions (link) involving 40,343 users (nodes).The rich temporal data contained within the Sarafu dataset makes it an ideal case study for applying our dynamic graph evolution rules approach. The dataset represents a contemporary human complex system that integrates humanitarian aid, collaboration, and financial aspects. By analyzing this dataset, we aim to capture and characterize the temporal behavioral patterns of nodes within this transaction network, providing insights into the dynamics of digital currency exchanges in a humanitarian context.

9.4 Results

We applied the described methodology to the Sarafu transaction network. Through Node Evolutionary Profiles (NEPs) we point out interesting insights into the dynamics of ego-networks. By identifying distinct interaction patterns and traits, we showcase the efficacy of NEPs in capturing temporal behaviors. The results highlight two primary evolutionary traits within the network that can be extracted by clustering NEPs. This analysis of Sarafu serves as a showcase of the potential applications of NEPs, demonstrating their capability to uncover the dynamics of ego-networks in complex transactional networks, which can be extended to various other domains and contexts.

9.4.1 Preprocessing and filtering

The extraction of the ego-networks from the original transaction network of Sarafu has returned 40343 ego-networks, which were reduced to 16030 after applying the filter on consecutive snapshots. This important reduction in the number of valid ego-networks indicates that more than half of the accounts do not show interactions with and among their neighbors in consecutive timestamps[1]. First, we investigate, for each ego-network, the number of included interactions to assess if, even in this case, this quantity has a heavy-tail trait as most of the phenomena concerning real-world networks and human dynamics [139]. To this aim, in Figure 9.1 we report the cumulative distribution of the number of interactions (ego-network size) in each ego-network. The distribution of the ego-network size is skewed, with the majority of nodes having very few interactions within their ego-networks. This observation may impact the outcomes of our analysis since if the temporal subgraph from which we identify the evolutionary profile is too small, it does not have enough data to actually describe the temporal behavior of a node. For this reason, based on the distribution, we only consider nodes whose ego-network presents at least 116 interactions, corresponding to the 80^{th} percentile of the ego-network size distribution, shown in Figure 9.1 by the blue dotted line. Thus, we obtain 3207 ego-networks, whose size distribution is depicted in Figure 9.1 with a green line. In short, the analysis of ego-network sizes allowed us to identify the most significant accounts in the Sarafu networks in terms of transaction activity; and at the same time highlighted that most economic activities are handled by a small portion of the accounts in the system.

[1] We are aware that the reduction of ego-network is quite important but not applying the filter on consecutive timestamps would have altered the dynamics within ego-networks.

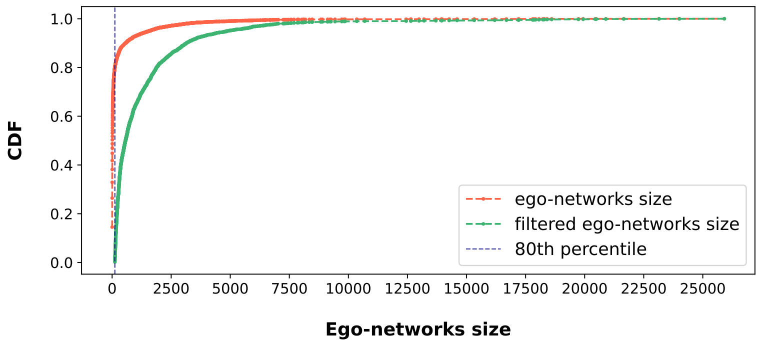


Figure 9.1: Ego-network size distribution. *In red the CDF of ego-network size from the 16030 ego-networks after applying the filter on consecutive snapshots. The vertical blue dotted line indicates the* 80*th* *percentile (116 transactions) of the red distribution. In green the distribution of ego-network size for the most active ego-networks (3207 elements).*

9.4.2 NEPs

After the preprocessing and filtering phase, we apply the EvoMine algorithm on the 3207 ego-networks. In particular, we fixed 1 as the minimum support σ - the algorithm returns all the graph evolution rules in the graph - and a maximum number of three edges per pattern, using the event-based support. As a result, after applying the general mapping procedure, we obtained 40 different graph evolution patterns, which correspond to the dimensions of the node evolutionary profiles. We collect all the NEPs by stacking them into a matrix, and we visualize it through a heatmap, where rows represent ego-networks and columns indicate the IDs of the graph evolution rules. The matrix of all the NEPs is displayed in Figure 9.2. From a column-wise inspection, we observe that in general, only a limited set of graph evolution rules characterizes the dynamics of the transactions in ego-networks. Indeed, there is concentration of frequency in rules 0, 1, 2, 3, 4, 10 and 15, with the first rule (0) being the most frequent for many ego-networks. We report these frequent GERs on the right side of Figure 9.2. We note that six out of seven rules describe star- or chain-like expansions starting from an empty precondition, while rule 10 expresses transactions that become reciprocal in just one day. Moreover, from a data perspective, the skewed distribution of rules points out it is very likely that if we reduce the dimensionality of NEPs most of the information in the data will be maintained. On the other side, from a row-wise inspection, we

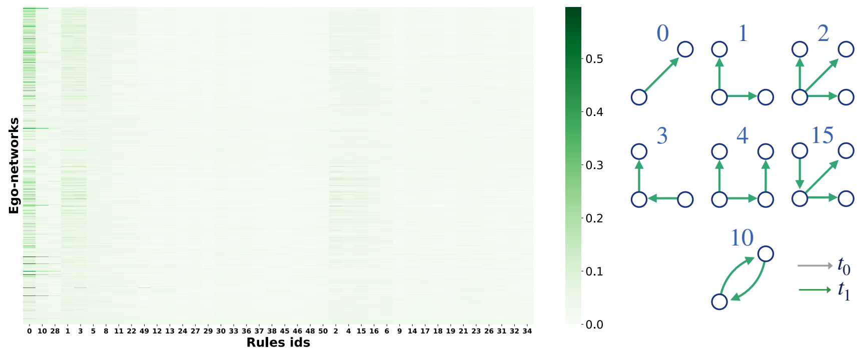
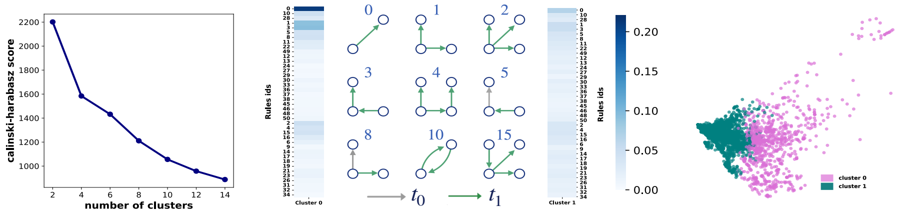


Figure 9.2: NEPs. *Visualization as a heatmap of the matrix obtained by stacking all the NEPs. Each row corresponds to an ego-network while the dimensions of NEPs (rules) have been ordered by multiple criteria which capture the increasing complexity of the pattern: the first criterium is the order - number of nodes - of the rule, then we consider the size and finally the number of timestamps (1 or 2). The intensity of the green color is proportional to the frequency of the GER. On the right the most common GERs represented through the compact visualization described in Figure 7.1.*

observe a certain level of variability in NEPs, so there is not a common trait characterizing the dynamics of the interactions in ego-networks; even if a few representative traits are identifiable.

9.4.3 NEPs clustering

The analysis of NEPs has pointed out two principal observations that are fundamental for showcasing how NEPs can be exploited for data discovery tasks: *i)* only a few GERs are frequent in NEPs; and *ii)* NEPs are varied but with a limited level of variability. Based on these observations, we focus on identifying a few classes that may represent dynamic traits of ego-networks evolution in Sarafu. We apply a clustering pipeline on the NEP matrix to identify the different traits. Taking advantage of the first observation on NEP dimensions, we first performed dimensionality reduction by Principal Component Analysis (PCA) to reduce NEP dimensions while preserving most of the information in the NEP matrix; then we ran a hierarchical clustering algorithm on the transformed NEPs to identify groups of ego-network showing the same evolutionary trait. As for the PCA algorithm, we choose the number of components using the explained variance ratio. Fixing a cumulative percentage of explained variance to 0.99, we still halve the dimensions from 40 to 24.



(*a*) (*b*) (*c*)

Figure 9.3: *In a) The Calinski-Harabasz score as a function of the number of clusters k returned by the agglomerative clustering algorithm. In b) the NEP centroids of the two clusters along with the most frequent, on average, GERs. In c) NEPs are visualized in 2 dimensions with points and colored according to the membership of one of the two clusters.*

The transformed NEPs feed an agglomerative clustering algorithm using Ward as the linkage strategy since it is less sensible to noise and should return more even clusters in terms of size. When using agglomerative clustering, one has to select the best value for the number of clusters k, a fundamental parameter not known as apriori that must be fixed before running the algorithm. In this case, we select k as the value that maximizes the Calinski-Harabasz (CH) score of the agglomerative clustering by varying k from 2 to 14 with step 2. According to the trend of the CH scores reported in Figure 9.3a, we choose $k = 2$. Thus, there are two main traits characterizing the dynamics of the transactions occurring within ego-networks, and consequently two groups of accounts. In Figure 9.3c we display these two groups of accounts in a 2D representation returned by applying PCA, while in Figure 9.3b we report the centroids of the two clusters to highlight the difference between the two prototypal behaviors. In particular, we note that the average behavior characterizing the dynamics of the interactions in ego-networks of accounts belonging to the cluster 0 is dominated by the rule 0 - the creation of a single link at time $t + 1$ when the precondition is empty - which on average accounts for the 20% of the rules involved in the dynamics of ego-network. The remaining rules also characterize the cluster 1, but on average they are spread more uniformly than in cluster 0. In short, the dynamics of transactions in the ego-networks of the accounts in the first clusters are mainly driven by star- and chain-like expansion rules that appear in a successive timestamp without a precondition, where the appearance of a single link is dominant; on the contrary, the dynamics in the second cluster are more homogeneous even if they lead to the same kind of expansion rules. The most important graph evolution rules are the same, while it is the rule frequency that differentiates the two dynamics traits.

Given these two traits and the networked nature of our dataset, we finally wonder if accounts that usually interact by exchanging transactions are characterized by the same rules describing the dynamics of their ego-networks. To cope with this question, we first proceed by computing a static projection of the graph sequence describing the Sarafu temporal network, then we compute the assortativity of the network using the clusters returned by the clustering algorithm as categorical attribute. In detail, in the static projection two accounts are connected if they interact at least once during the observation period, and the directed links are weighted according to the number of transactions sent by the source node toward the target node. Moreover, the construction of this graph is limited to the 3027 accounts in the analysis. In this setting, the attribute assortativity is 0.590 and indicates a tendency for accounts to interact with other accounts that have a similar ego-network dynamic. In general, we stress the fact that by utilizing node evolutionary profiles, it is possible to develop applications that highlight properties and relationships between accounts based on the dynamics of the extracted ego-networks.

9.5 Conclusions

In this study, we introduced a method for representing ego-network dynamics through subgraph-based evolution rules, enabling a nuanced analysis of temporal behaviors within networks. Applying Node Evolutionary Profiles (NEPs) to the Sarafu transaction network revealed significant insights, identifying two main interaction traits: one dominated by the single-link expansion over other star- and chain-like expansions, and another with a more homogeneous distribution among the same expansion rules.

These findings underscore the potential of NEP-based representations to reveal underlying behavioral patterns in complex networks. Analyzing ego-network dynamics with graph evolution rules supports various applications, such as distinguishing user behaviors in financial transaction networks like Sarafu, crucial for operational improvements and strategic decision-making. Beyond identifying behavioral traits, the NEP-based approach enhances the understanding of interaction dynamics, aiding in the development of applications that highlight the properties and relationships between accounts.

Chapter 10

Community evolutionary profile

10.1 Introduction

The study of dynamic networks in computer science has become crucial, given their ever-evolving nature within digital ecosystems. These networks serve as fundamental models for various networked systems, usually characterized by modular structures. Understanding these structures, also known as communities, and the mechanisms driving their evolution is vital, as changes in one module can impact the entire network. Traditional static network analysis falls short of capturing the full complexity of dynamic networks, prompting a shift toward understanding the underlying mechanisms driving their evolution. Graph evolution rules (GERs) have emerged as a promising approach, explaining how subgraphs transform into new configurations. In this paper, we comprehensively explore GERs in dynamic networks from diverse systems with a focus on the rules characterizing the formation and evolution of their modular structures, using EvoMine for GER extraction and the Leiden algorithm for community detection. We characterize network and module evolution through GER profiles, enabling cross-system comparisons. By combining GERs and network communities, we decompose network evolution into regions to uncover insights into global and mesoscopic network evolution patterns. From a mesoscopic standpoint, the evolution patterns characterizing communities emphasize a non-homogeneous nature, with each community, or groups of them, displaying specific evolution patterns, while other networks' communities follow more uniform evolution patterns. Additionally, closely interconnected sets of communities tend to evolve similarly. Our findings offer valuable insights into the intricate mechanisms governing the growth and development of dynamic networks and their communities, shedding light on the interplay between modular structures and evolving network dynamics.

10.2 Background

This section provides a summary of the main results on the evolution of communities as well as vector-based representation of evolution patterns. For details about the graph evolution rules algorithm please refer to Chapter 4.

10.2.1 Community evolution

In recent years, the study of community evolution within dynamic networks has emerged as a focal point in the field of network science. As complex systems evolve and adapt over time, understanding how communities within these systems change and reconfigure is of paramount importance. Many studies delve into identifying events that define the life cycle of communities, such as splitting and merging [71, 140, 73, 141]. For instance, Takaffoli *et. al.* [142] considered five events that a community can undergo (split, survive, dissolve, merge, and form), and proposed a community matching algorithm to efficiently identify and track similar communities over time. Another direction in the context of dynamic community detection involves event prediction: many studies characterize the evolution of communities within an event prediction framework [143, 144, 145, 146]. In this context, an experimental platform called *EPredictor* [147] has been developed to enable testing, verifying, and validating models related to community evolution prediction in dynamic social networks. Moreover, Ilhan *et. al.* [148] enhanced community event prediction with a framework, called *FIEP*, that identifies the network's most representative features before the community evolution process. In this paper, we deal with community evolution from a different perspective, since we are interested in decomposing the evolution of communities in local dynamics rules - GERs - strictly related to the mechanisms driving the growth, rather than identifying and predicting events communities may undergo. Specifically, we will employ community detection algorithms as tools for the identification of target communities asking which are the microscopical evolution patterns leading to the actual structure of the groups. From this perspective, we do not apply algorithms for tracking the life cycle of communities since they are mainly focused on the identification of large events involving groups as a whole, i.e. splitting, merging, and appearance; rather we rely on an approach based on temporal subgraph mining since it is more suitable for extract localized temporal behaviors within the network and communities. In this sense, our approach is also complementary to works aiming to predict the evolution of communities, since it may provide a set of features based on temporal subgraphs that are not model-dependent or hand-crafted for a specific context, but, at the same, keep a good level of readability since they are not the outcome of neural architectures applied on graphs.

10.2.2 Vector-based subgraph representation

In existing literature, some studies have suggested adopting a vector representation method to assess and categorize networks based on the frequency of subgraphs. For instance, in the seminal work by Milo *et. al.* [53] the so-called *significance profile* records the z-scores of subgraphs and it is used for network comparison and analysis. In some works, the vector-base subgraph representation is used as network embedding for machine learning tasks [149, 150, 151, 152, 153]. For instance, Yu *et. al.* [152] proposed a node-level embedding by only considering undirected statistically significative subgraphs, i.e. network motifs; while Tu *et. al.* [154, 155] have incorporated direction and time to build their graph-level embedding, called subgraph ratio profile. Along this line, a strategy proposed by Longa *et. al.* [60] builds a temporal subgraph-based representation of nodes' neighborhood by collecting egocentric temporal subgraphs at each time step and condensing them into egocentric temporal motifs (ETMs). Finally, the approach of Xu *et. al.* [153] accomplish both graph-level and node-level embedding. Integrating the subgraphs distribution with important features, they proposed a multi-view motif network representation framework (MMNR) that allows good performance in machine learning tasks.

10.3 Methodology

The comprehension of the main patterns driving the evolution of large dynamics networks and how these phenomena are related to the modular structure of the networks passes through a methodology that combines methods for the extraction of evolution patterns, namely graph evolution rules, and algorithms for the identification of modular components in a network, i.e. community detection algorithms. In this section we provide a description of the overall methodology, and the representation of evolution patterns in the entire networks and within communities. The details about the algorithms for the identification of graph evolution rules and communities can be found at Chapter 4.

10.3.1 GER profiles

Through the application of the EvoMine algorithm, we obtain the number of discovered rules and the identification of the most frequent ones. However, our aim is to create a tool that simplifies the process of extracting insights from these results. To this aim, we introduce a vector-based representation of the graph evolution rules, called *GER profile*, that allows us to concisely and visually summarize the network's evolutionary behavior. The GER profile is a vector that reflects the distribution of support for each rule type. Therefore, our initial step involves establishing a canonical form to consistently identify the same pattern across the various applications (graphs) of the EvoMine algorithm. After the identification of all the subgraph isomorphism classes and the consequent assignment of a general id to each of them, the GER profile $v(G, s)$ of graph G can be computed from the rules' support list obtained running EvoMine on G with support threshold s. Specifically, each element of the vector $v(G, s)$ is characterized as:

$$v(G,s)_i = \frac{\sigma_{event}(r_i)}{\sum_{j=1}^{n} \sigma_{event}(r_j)} \tag{10.1}$$

where $\sigma_{event}(r_i)$ is the event-based support of the rule r_i and n is the number of distinct GERs identified by EvoMine. The distinct GERs we are referring to can be only the ones identified by the run of EvoMine on a specific graph G, but they can also be the union of all rules found on different graphs. In the second case, if a rule r_x is not present in the output of G - it is either not present at all or it has support lower than the threshold s - then $v(G, s)_x = 0$. The purpose of considering the union of the rules found on different networks is to have a common outcome set to allow the comparison of GER profiles inferred from different graphs.

10.3.2 Community GER profiles

The methodology discussed so far mainly focuses on the entire evolving graph and provides a global characterization of the evolution patterns of the graphs. However, such a general overview neglects how dynamic networks are organized into interconnected modules or communities. So we shift our attention from a comprehensive analysis of the entire graph to a more mesoscopic viewpoint.

Specifically, we apply the EvoMine algorithm and compute the GER profiles changing the input graphs: the communities obtained through a community detection algorithm based on modularity optimization [156]. Specifically, we employ the Leiden algorithm [157] to identify communities within the input graph, as it represents the state-of-the-art and most used algorithm for community detection based on modularity optimization. Subsequently, we focus on the top communities, i.e. those with the highest number of nodes and whose union covers the 80% of nodes. For each selected community, we generate a node-induced subgraph and apply the EvoMine algorithm to analyze its evolution. In the end, we obtain a matrix of GER profiles $M(G, comms, s)$ of cardinality $n \times c$, where c is the number of communities and n is the total number of rules, i.e. the union over all the communities. By computing the GER profile for each community, we are now able to observe and compare the distinct evolutionary patterns exhibited by different modules in the network.

10.3.3 Support choice method

A critical aspect of extracting graph evolution rules through the EvoMine algorithm is choosing the support threshold s: if the threshold is set too low, it may lead to slower computations and the inclusion of excessively infrequent rules. Conversely, a threshold set too high may result in the omission of potentially important rules with support just below the threshold. Hence, we introduce a heuristic to determine an appropriate support threshold for each specific graph. Our starting point is the rules derived from setting the absolute minimal support threshold (set at 1), enabling the identification of all existing evolution rules within the graph. Computing the GER profile, we obtained the distribution of supports over all the rules. In the process of threshold selection, the central criterion is the coverage of the rule distribution that we can obtain while increasing the support value w.r.t. the initial one with minimal support. Formally, given the GER profile of the graph G built from the minimal support evolution rules $v(G, 1)$ defined in Equation 10.1, the coverage $cov(G, s)$ relative to the support threshold s is defined as

$$cov(G, s) = \sum_{i \in ger_s} v(G, 1)_i$$

where $ger_s = \{g | \sigma_{event}(g) \geq s\}$. In other words, the coverage relative to support s is the sum of the support of all graph evolution rules ($g \in ger_s$) whose absolute support is greater than the current threshold s. The method can be applied to both the community GER profiles and the overall graph GER profile. When focusing on the community GER profiles, we take into account the mean and standard deviation of coverage across various communities and aim to establish an individualized threshold for each community. In this case, it is crucial to set individual thresholds for each community due to variations in the size of their node-induced subgraphs, making a single threshold impractical. Setting the steps for evaluating the coverage variation while the support increases is a challenging task. To address this, we considered the *maximum support*, a concept also discussed in Coscia et *al.* [158]. In the context of event-based support, which counts the number of event graphs where a rule occurs, the maximum support equates to the total number of events in the graph. In our specific case, where the input is a directed graph and we exclusively consider edge insertions and omit edge deletions, the maximum support corresponds to the number of edges of the considered graph. The maximum support allows us to evaluate the coverage for specific steps $S = \{\frac{i}{20} \cdot max\_sup | i \in [1 \ldots 10]\}$. In particular, we calculated the mean and standard deviation of coverage across all communities for each step, and subsequently, we selected a support threshold that guarantees a high average coverage and limits the coverage dispersion at the same time. On the other hand, as for the GER profile of the total graph, we compare the coverage with the number of rules that are excluded if we increment the support threshold. Specifically, given the minimum support GER profile $v(G, 1)$ and the support threshold s, we compute the number of lost rules $lost(G, s) = |v(G, 1)| - |v(G, s)|$, i.e. the difference between the minimal support profile and the one with support s. Finally, observing the variation of $cov(G, \lambda)$ (distribution coverage) and $lost(G, \lambda)$ (number of rule lost), $\forall \lambda \in S$, we determine the threshold that provides a trade-off between these two quantities.

10.3.4 Case studies

We applied our methodology to four diverse datasets: Sarafu, DBLP-cite, Enron, and Stack Overflow. These datasets were selected for their modular structures and to represent a range of network types, varying in both dynamics and scale. Detailed information about the nature of these datasets and our modeling approach can be found in Chapter 6, while Table 10.1 reports the size of the obtained graphs.

Network	Nodes	Edges
Sarafu	40323	143239
DBLP	11971	47654
Enron	32128	107234
Stack Overflow	2115635	13519681

Table 10.1: *Number of nodes and edges for each graph.*

10.4 Findings

In this section we first discuss the GER profiles characterizing the evolution of the entire graphs, providing both quantitative and qualitative descriptions. Then, we focus on mesoscopic aspects, analyzing the GER profiles of the different modules/communities within the dynamic networks described in the previous section to characterize how each specific community has evolved and which are its leading evolving mechanisms. While delving into both the global and community profiles, we concurrently address the issue of selecting the proper support parameter by applying the heuristic described in the methodology Section. Moreover, it is worth noting that the EvoMine algorithm requires two main parameters: the minimum support for a rule to be considered frequent - s - and the maximum number of edges in the postcondition graph (that considerably helps to reduce the computational effort of the algorithm). While for the choice of the support threshold we dedicated a specific method, as far as concerns the maximum number of edges we always set it to 3 as in previous works [40]. Finally, we conduct a comprehensive discussion of the intricate mechanisms governing community dynamics aiming to shed light on the similarities, distinctions, and deeper implications of these dynamics within diverse networks. Specifically, we delve into the discussion regarding the community GER profiles - evolution rules within communities - and subsequently, we proceed to compare and discuss the results concerning the evolution of neighboring communities.

10.4.1 Global evolution

Support choice

In the process of determining the support parameter for our analysis, we looked for a balance between computational efficiency and the identification of valuable rules. As described in the Methodology Section, we initially set a support threshold of 1, allowing us to capture an extensive array of potential rules and patterns within the dataset. Subsequently, we systematically increased this threshold and monitored its impact on the number of rules retained and the distribution of the coverage. In the case of the entire dynamic network, we identified the inflection point where the distribution of the coverage and the reduction in the number of rules (*lost*) are balanced. The trend of these two quantities as a function of the support threshold[1] has been reported in Figure 10.1. By definition, the distribution of the coverage shows an increasing trend, while the number of rules missing decreases as the support threshold approaches the minimal threshold. This corresponds to $\frac{1}{n} \cdot max\_sup$, with $n = 20$ for the first three datasets with medium sizes, while for Stack Overflow, where the graph dimensions are significantly larger, $n = 2,000$. In the case of Sarafu network (see Figure 10.1a), the selected support parameter is $\frac{6}{20} \cdot max\_sup = 8530$, since $max\_sup = 142177$. The rationale driving this choice is that we get a good level of coverage (more than 70%) and we limit the number of lost rules to ten, as highlighted by the yellow box in the figure. In the other two networks (DBLP and Enron, respectively in Figure 10.1b and 10.1c), we selected the highest value following a stable series of observations. Specifically, we opted for $\frac{8}{20} \cdot max\_sup$ for both datasets, which corresponds to a support threshold set to 3790 and 8174 for DBLP and Enron networks, respectively. Concerning the Stack Overflow network, the threshold is $\frac{45}{2000} \cdot max\_sup = 304107$ since it represents a pivotal point in both coverage and lost trends. More precisely, the $\frac{45}{2000} \cdot max\_sup$ threshold marks the highest value before a peak in the number of rules lost, coinciding with a decline in the coverage percentage.

Quantitative results

By applying EvoMine with the set of parameters previously defined we obtained the number of rules reported in Table 10.2. By observing the table it is worth noting that not only Sarafu and Enron networks do present the same number of rules, but they also correspond in terms of type. This observation does not imply that they have similar evolutionary behavior (frequencies of the rules may differ), but only that the evolution patterns that can be observed are the same. Another key insight is that for larger networks (DBLP and Stack Overflow), the number of frequent rules is lower. From the perspective of evolutionary mechanisms as graph evolution rules, this observation suggests that

[1] For each dynamic network the support threshold is expressed as a percentage of the maximum support.

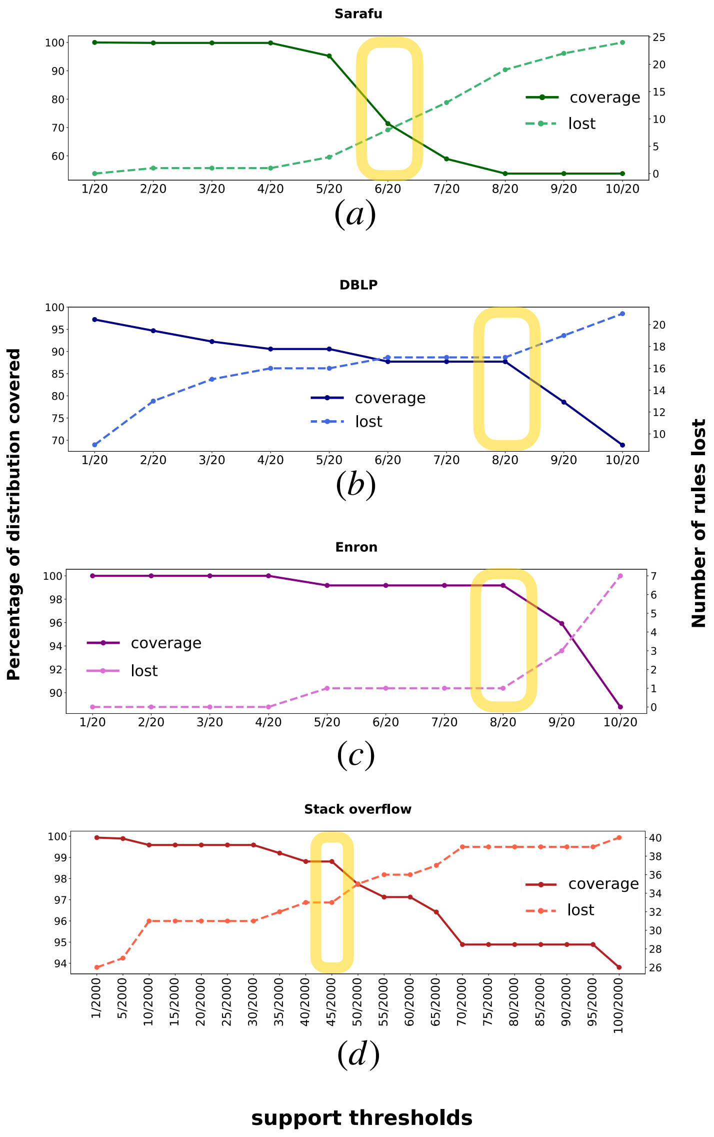


Figure 10.1: Coverage and lost rules. *In (a), (b), (c) and (d) the coverage and the number of lost rules by varying the support threshold. On the x-axis, label ticks indicate the fraction in the formula* $\frac{i}{n} \cdot max\_sup$. *For the first three datasets (Sarafu, DBLP, and Enron)* $n = 20$ *while for Stack Overflow* $n = 2000$, *due to its different magnitude. In the yellow boxes, we highlight the support threshold representing a trade-off between coverage and lost rules.*

rules driving the formation and evolution of these networks are quantitatively limited so we can describe the entire growth of networks with few of them.

From a computational viewpoint, it is worth noting that for the Stack Overflow dataset, given its substantial size, we adopted a parallel strategy for the EvoMine algorithm to analyze its temporal dynamics efficiently. For the timespan covered by the dataset $\{t_1, t_2, \dots t_n\}$, we apply EvoMine on each pair of consecutive timestamps instead of the whole temporal directed graph. This replicates the internal behavior of EvoMine with the event-base support, which computes the support for each rule across consecutive timestamps (event graphs) and aggregates the results. Our approach however enables parallel computation on different timestamps, thus saving time and computational resources w.r.t. the original implementation of the algorithm. Despite the external parallel execution, the results remain comparable in the aggregation phase due to an isomorphism check integrated into the algorithm. This check, as implemented in [159], assigns a unique identifier to classes of isomorphic rules, ensuring consistency in the analysis across different timestamps.

Network	**Support threshold**	**Number of GERs**
Sarafu	8530 ($\frac{6}{20}$)	40
DBLP	3790 ($\frac{8}{20}$)	15
Enron	8174 ($\frac{8}{20}$)	40
Stack Overflow	304107 ($\frac{9}{400}$)	18

Table 10.2: *Support threshold and number of rules obtained by EvoMine for each network.*

GER profiles

The graph evolution rules returned by applying EvoMine determine the GER profiles depicted in Figure 10.2. As described above, a GER profile is a distribution over rules; specifically, the GER profiles displayed are built of the union of the evolution rules extracted from the four networks. If a rule falls below the chosen support threshold, the corresponding frequency in the GER profile is set to zero. This situation only occurs in the DBLP and Stack Overflow cases (blue and orange line) because they have 25 and 22 rules fewer than the other two networks. From the plot in Figure 10.2, it becomes evident that Sarafu and Enron globally exhibit similar evolutionary behaviors. On the contrary, comparing the GER profiles of DBLP and Stack Overflow with the others is more challenging due to differences in scale. The DBLP and Stack Overflow profiles differ significantly from Sarafu and Enron and highlight a different evolutionary profile within each other. However, they share a common characteristic: they are less homogeneous in terms of rule types, often presenting remarkable peaks in their profiles. A more detailed analysis based on all the rules reported in the Appendix highlights that the main difference between Enron/Sarafu and DBLP/Stack Overflow is due to evolution rules involving reciprocal links in the precondition or postcondition. Indeed, in DBLP and Stack Overflow networks, these rules are almost completely missing or sporadic. In the case of DBLP, this observation means that reciprocal citations are not expected to occur, while in Stack Overflow this phenomenon can be reasonably ascribed to the different roles of the accounts, i.e. beginners who pose questions and experts who mainly comment to answers or other posts. This latter aspect is further supported by observing that in Stack Overflow the most frequent rules describe neighborhood expansion mechanisms originating from a single source node, i.e. an expert who replies to different accounts. On the other side, in DBLP the most frequent evolutionary patterns result in the creation of chain-like postcondition graphs, where A cites B who cites C who cites D. As for this latter aspect, it is to note that these chain-like citations actualize in only one year.

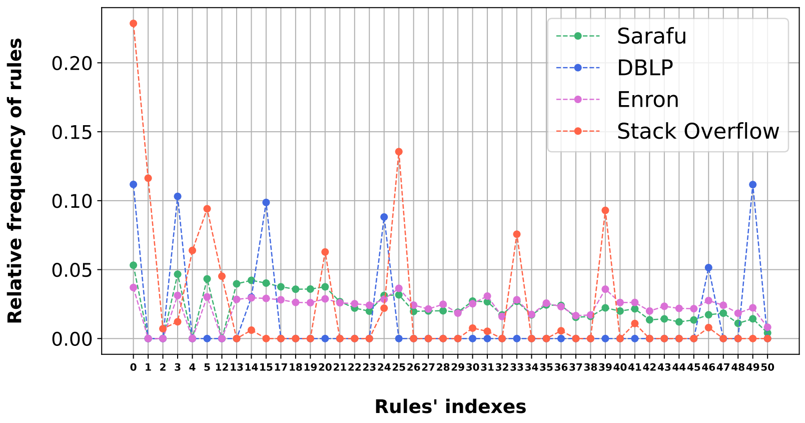


Figure 10.2: GER Profiles. *Visualization of the GER profiles associated with the four networks. On the x-axis the IDs of the evolution rules in ascending order, and on the y-axis the relative frequency of each rule.*

Moreover, the approach we adopted for computing the GER profiles in Stack Overflow, i.e. merging the graph evolution rules extracted from each snapshot and properly aggregating the different frequencies, allows us to analyze the consistency and stability of the different evolution rules during the network evolution. Specifically, for each snapshot of the Stack Overflow network, we extract its GER profile and stack them over time. The resulting trend of the GER profiles over time is depicted in Figure 10.3. In the figure, we can observe two main traits characterizing the growing patterns of Stack Overflow: *i)* all the most frequent rules (except for rule 0 - discussed later) are stable and consistent over time, since their frequencies are almost equal over the observation period; and *ii)* the frequency of rule 0 has slightly increased over time. The first observation indicates that *the contribution of the main evolution mechanisms is stable along the evolution of the network*, i.e. there are growing rules that constantly act on the formation of specific future subgraphs. Second, since rule 0 captures the formation of a new link among two not-connected nodes, we reasonably assess that during the last period of Stack Overflow unconnected accounts are more likely to form relationships than in the first period.

In general, the GER profile trend is a further tool to describe a network's evolution patterns and evaluate the stability of the evolution rules.

10.4.2 Evolution patterns in communities

The above findings have highlighted that from a global perspective, the evolution patterns characterizing the growth of the four networks are very similar when the size of the network is comparable, while they differ as the size grows. Consequently, from a mesoscopic viewpoint, we wonder whether these similarities among networks still hold when we shift our focus to the community level. To this aim, we conduct a comprehensive examination of the modular structure of the four networks by selecting those communities suitable for further analysis. Subsequently, we delve into the presentation of the community GER profiles, addressing the central research question concerning the consistency of evolution patterns across modular networks.

Leiden communities

Here, we comment on the outcomes of the Leiden algorithm applied to four selected networks by discussing each case separately.

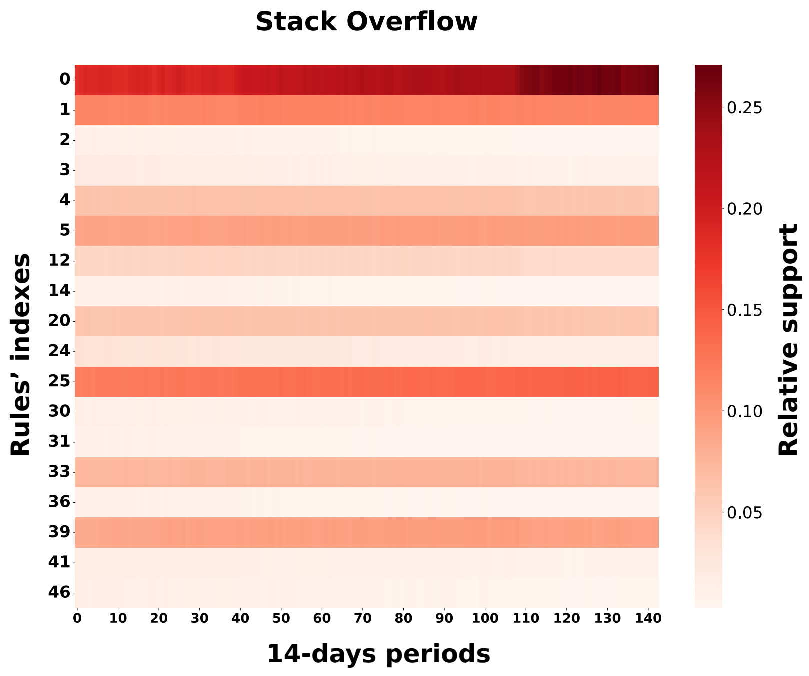


Figure 10.3: GER profile trend. *Visualization of the GER profiles stacked over time for the Stack Overflow network. Each column represents the GER profile for a given 14-day snapshot, and the rows indicate the evolution rules. The color profiles are proportional to their frequencies.*

We start by delving into the communities characterizing the **Sarafu** network, where the application of the Leiden algorithm yields a total of 684 communities. In particular, the first 20 largest communities, whose nodes and edges count are depicted in Figure 10.4a and 10.4b, exceed the 80% of node coverage (80.43%). In the following analysis, we will focus on this representative set. Figure 10.4b illustrates the count of edges within individual node-induced subgraphs obtained from the respective community's set of nodes. It is to note the peculiarity of community 11: while all other communities ordered by number of nodes almost follow the same ranking for number of edges, community 11 shows the maximum size even if it does not have the highest number of nodes, meaning it is the densest among the 20 largest communities.

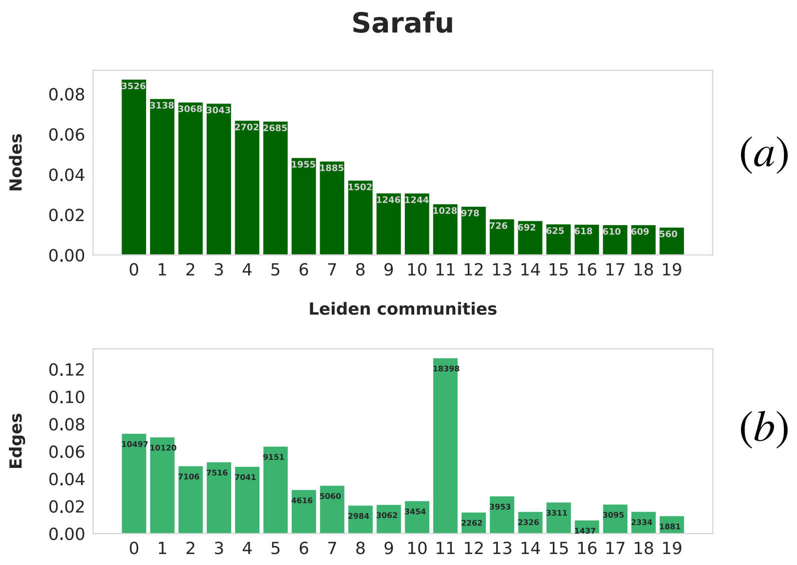


Figure 10.4: Sarafu communities. *In (a) the percentage of nodes in each of the* 20 *largest communities. On top of each bar is the number of nodes in the community. In (b) the percentage of edges in the node-induced subgraph for each community. On top of each bar is the number of edges in the node-induced subgraph.*

Moving on to the second dataset in our analysis, we delve into the **DBLP** citation network. The Leiden algorithm applied on the DBLP graph results in 47 communities. In Figure 10.5a we display the 11 largest communities in terms of node set size, along with the percentage and the absolute number of nodes they include. The first eleven communities include 84.37% of all the nodes of the graph, and each one of them includes at least 5% of nodes. For the subsequent analysis related to the DBLP network, we focused on these top 11 communities. Figure 10.5b shows the number of edges included in each node-induced subgraph obtained by the set of nodes in each community. As expected, the number of edges closely aligned with the number of nodes, as communities with more nodes, also tended to have more edges.

Continuing our exploration of communities, we turn our attention to the **Enron** email dataset. In this case, the Leiden algorithm identifies a total of 114 communities, where the 17 largest ones cover almost 70% of nodes, and each of them includes at least the 1.5% of nodes. As in the previous cases,

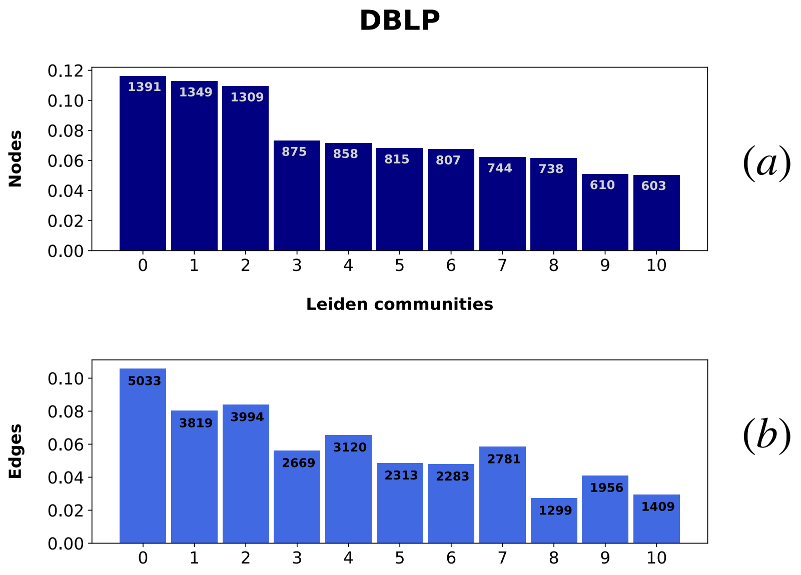


Figure 10.5: DBLP communities. *In (a) the percentage of nodes in each of the 11 largest communities. On top of each bar is the number of nodes in the community. In (b) the percentage of edges in the node-induced subgraph for each community. On top of each bar is the number of edges in the node-induced subgraph.*

we consider only this set of communities for further analysis. In the Enron network, communities ordered by the number of nodes (see Figure 10.6a) are not aligned with the rank induced by the number of edges (see Figure 10.6b). In fact, there are some communities, for instance, numbers 4, 9 and 14, that have a significantly lower number of edges w.r.t. communities with a comparable number of nodes, indicating them being less dense than the others.

Finally, the Leiden algorithm applied to the Stack Overflow network returns 8217 communities, with nodes predominantly concentrated in the first ones. We narrow our focus to the first 10 communities for further analysis, as they cover over 97% of all nodes. The distribution of nodes within these selected communities is illustrated in Figure 10.7. The ranking based on the number of nodes (Figure 10.7a) closely mirrors that of the number of edges (Figure 10.7b), except for the second community (indexed as 1), which exhibits lower density compared to the subsequent one (indexed as 2).

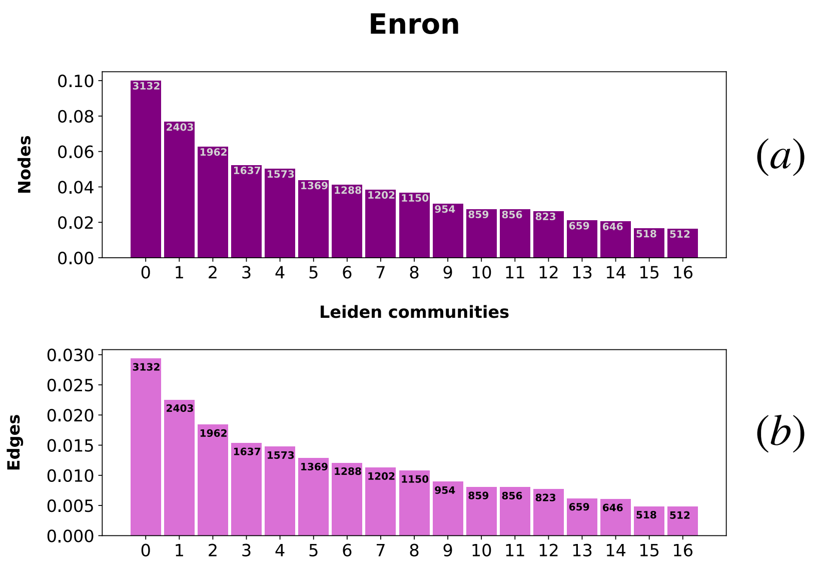


Figure 10.6: Enron communities. *In (a) the percentage of nodes in each of the* 17 *largest communities. On top of each bar is the number of nodes in the community. In (b) the percentage of edges in the node-induced subgraph for each community. On top of each bar is the number of edges in the node-induced subgraph.*

In short, three networks exhibit a good modular structure, confirmed by their high modularity score, i.e. Sarafu modularity is 0.89, in DBLP the modularity is 0.64 and Enron got 0.72 as modularity score; while Stack Overflow is characterized by a less pronounced modular structure (its modularity is 0.5). Moreover, in all the networks the top 10-20 largest communities include most of the nodes of the network, while the rank induced by the node set size is not always aligned with the rank induced by the number of edges.

Support choice for each community

Before delving into the discussion about the GER profile of each community, we have to deal with the choice of the support threshold for each community, as we did for the global perspective. As detailed in the Methodology Section, it became evident that choosing a single parameter for each community within

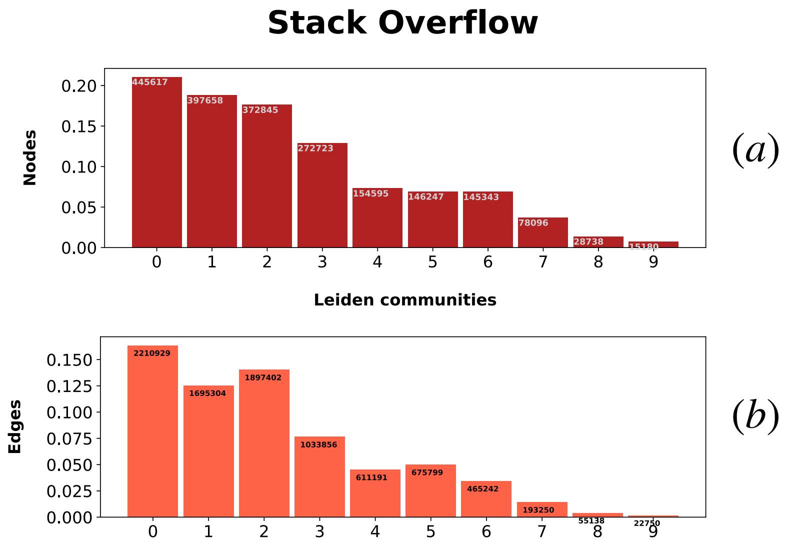


Figure 10.7: Stack Overflow communities. *In (a) the percentage of nodes in each of the* 10 *largest communities. On top of each bar is the number of nodes in the community. In (b) the percentage of edges in the node-induced subgraph for each community. On top of each bar is the number of edges in the node-induced subgraph.*

the same total graph is not ideal, as communities exhibit varying orders[2] and sizes. To return a support threshold valid for all the communities, for each step of the support values in S, we compute the distribution of the coverage on the set of communities and extract its average and standard deviation. In Figure 10.8 we report the average coverage (darker line) and its standard deviation (lighter-colored area) as a function of the support threshold, for the four networks. For instance, the point aligned with the support value $\frac{5}{20} \cdot max\_sup$ represents the mean of all the coverage values for all the communities, by applying the EvoMine algorithm with support equal to $\frac{5}{20} \cdot max\_sup$. By inspecting the above plots, we selected the support threshold corresponding to a good trade-off between a high average coverage and a minimal dispersion of the coverage. For Sarafu and DBLP datasets, the critical value is $\frac{4}{20} \cdot max\_sup$,

[2] Number of nodes.

because after that the standard deviation becomes wider and the mean value quickly decreases. In the Enron case it is less evident, but in $\frac{5}{20} \cdot max\_sup$ the standard deviation of the coverage starts slightly increasing, while the average coverage is still 95%. Finally, in the case of the Stack Overflow dataset, the threshold is $\frac{4}{20} \cdot max\_sup$: the plot in Figure 10.8d shows a more homogeneous standard deviation for all thresholds (the colored area behind the red line), however, the choice of threshold is $\frac{4}{20} \cdot max\_sup$ since from that point the standard deviation increases while the coverage starts decreasing.

Community GER profiles

Once we properly defined the support threshold for each network, we can analyze how graph evolution rules are distributed within the modules or communities; and establish whether communities evolve homogenously or not; and if nearby communities evolve similarly.

In the case of the **Sarafu** graph, by applying the EvoMine algorithm with support equal to $\frac{4}{20} \cdot max\_sup$ on each node-induced community subgraph, we obtain the same set of evolution rules for each community, whose cardinality is 40. After applying a generalized mapping for rule types to facilitate easy comparison, we compute the GER profiles for each community and visualize them using the heatmap depicted in Figure 10.9. In the heatmap, each column corresponds to the GER profile of a community, and each row represents an evolution rule. The darker the cell is, the more frequent the specific rule in the corresponding community is. The Sarafu heatmap of community GER profiles reveals two distinct regions (set of rules) where more frequent rules characterize certain groups of communities. In fact, since the colormap is general for all the GER profiles across each community, the communities with more uniformly distributed rule frequencies are easily identifiable - for instance, communities 1, 5, 13, etc... On the contrary, we can immediately distinguish the communities where the frequency is more concentrated on a few rules because cells are nearly white. In fact, in communities 2, 3, 4, 6, 7 and 8 the rules with identifiers greater than 41 are very infrequent. Even if the number of rules is limited, it is not immediate to summarize the evolution mechanisms characterizing these communities. According to the list of GERs depicted in Figure 10.21 in the Appendix, the rules characterizing this group of communities are composed by a precondition of two edges and only a new link in the postcondition, i.e. they have an insertion rate lower than rules from 41 to 50. In short, the *overall mechanisms driving their evolution in these communities are more conservative than those of the remaining communities in the Sarafu network.* In the Sarafu

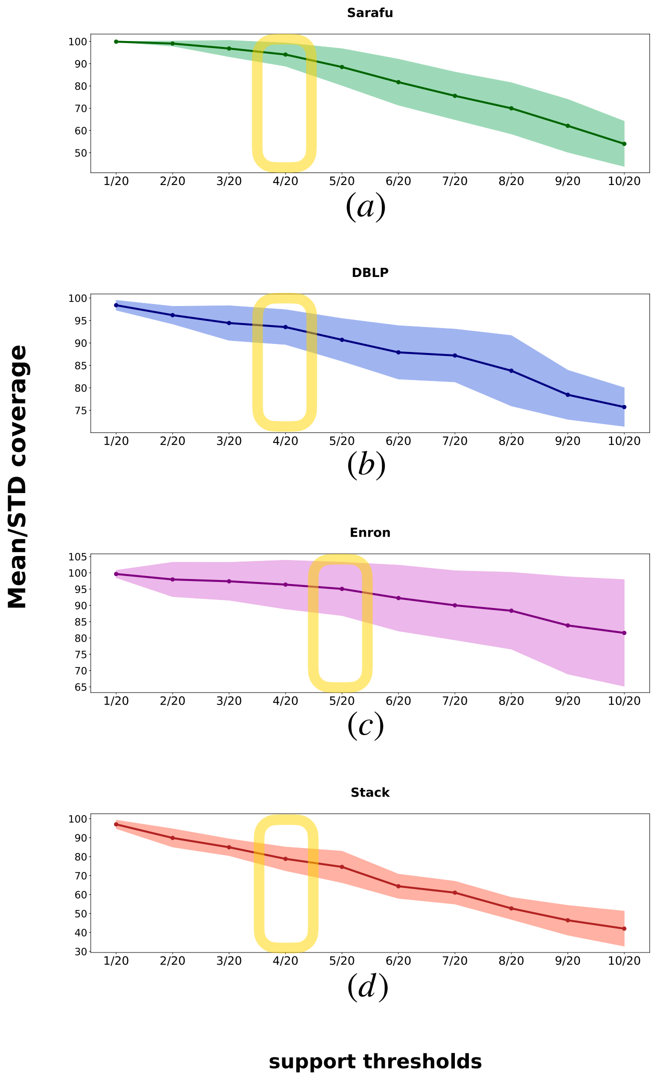


__Figure 10.8: Support threshold selection.__ In (a), (b), (c) and (d) the average and standard deviation of the coverage distribution derived by the GERs of each community as a function of the support threshold utilized by EvoMine, for Sarafu, DBLP, Enron, and Stack Overflow respectively. The dark line represents the trait of the average coverage, and the lighter-colored area indicates the standard deviation. The yellow boxes indicate the selected support thresholds according to the criteria described in the text.

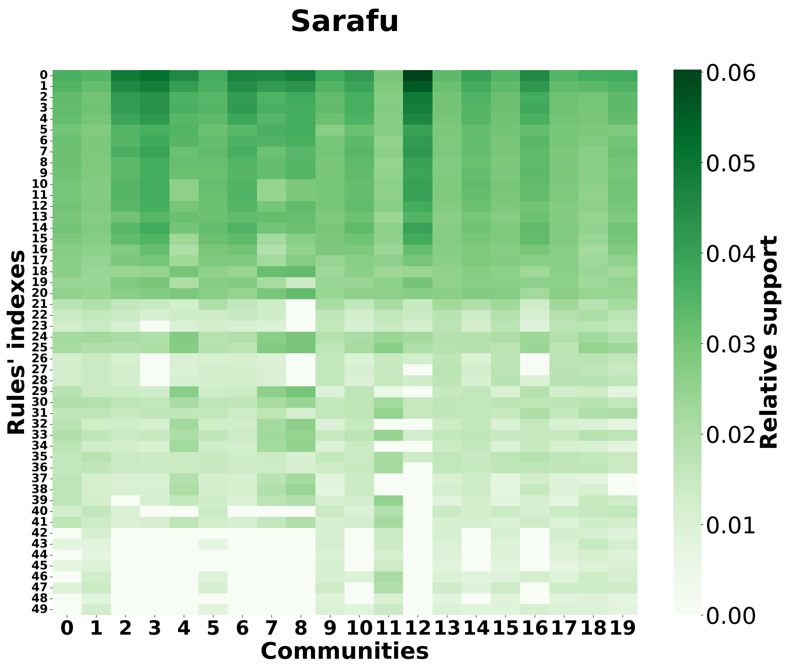


Figure 10.9: Communities GER profiles: Visualization of the GER profiles of the communities in the Sarafu network. Rows correspond to evolution rules, and columns correspond to the communities.

context, it may indicate accounts that are less inclined to establish new trading relationships.

As for the second dataset - **DBLP** - we employ the EvoMine algorithm to each node-induced community subgraph, setting a support threshold equal to $\frac{4}{20} \cdot max\_sup$. This yields a consistent set of rules in each community, even if with some oscillations. In fact, the number of rules fluctuates between 12 and 15. Figure 10.10 shows the GER profiles of the communities through a heatmap which is interpreted in a way similar to the Sarafu dataset (seeFigure 10.9). White cells indicate that the associated rules are not frequent (according to the support threshold) in those community subgraphs. Since the number and the set of evolution rules are not consistent across communities, in Figure 10.11 we report the number of evolution rules for each community. As two communities having the same number of rules may be characterized by different GER

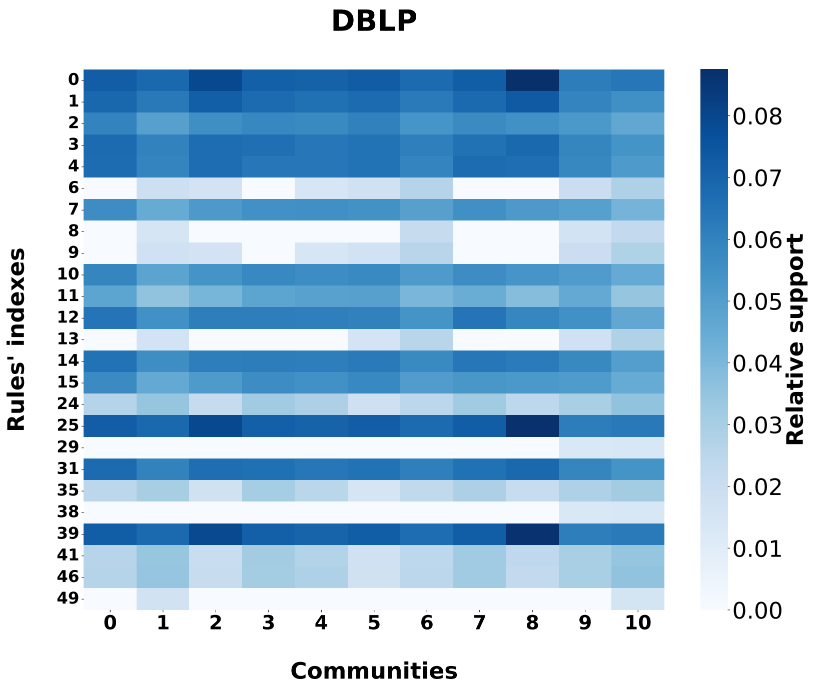


Figure 10.10: Communities GER profiles. *Visualization of the GER profiles of the communities in the DBLP citation network. Rows correspond to evolution rules, and columns correspond to the communities.*

profiles, we observe cases as, for instance, community 2 which has the same number of rules as communities 3 and 4 (12 rules), but the distribution in the first is less uniform. Despite a few subtle differences, in the DBLP network, the community GER profiles exhibit a more remarkable consistency across different communities w.r.t. the Safaru case.

As discussed in the previous subsection, the most frequent GERs in DBLP are related to the creation of chains of citations involving two to four DBLP contributions. The uniformity of the community GER profiles across the community indicates that *there are evolutionary mechanisms leading to the formation of communities that are common to different computer science topics.*

When we focus on the **Enron** network, where we apply the EvoMine algorithm with a support threshold set to $\frac{5}{20} \cdot max\_sup$, we observe a very specific and evident trait depicted in Figure 10.12. Indeed, it is clear that the evolu-

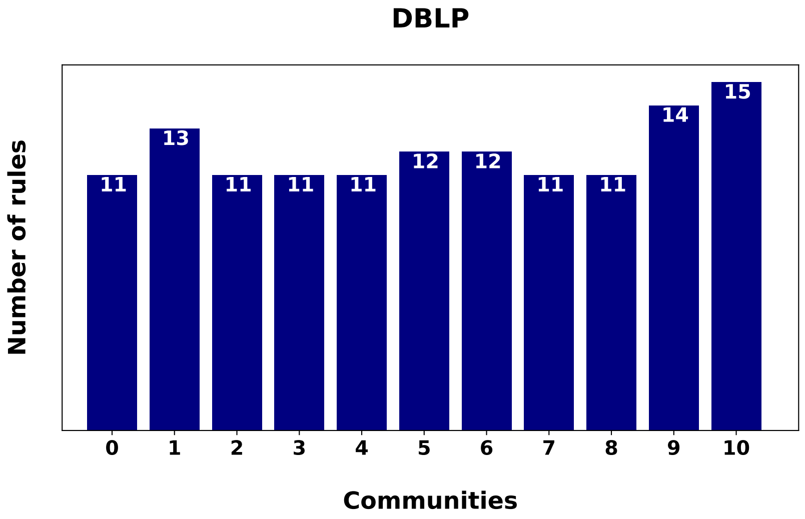


Figure 10.11: *Number of evolution rules for each community extracted from the DBLP citation network.*

tion of the communities follows two distinct traits: one characterized by very few rules, as in the case of community 4 or 9, and one denoted by a more homogenous evolutionary profile. Specifically, the number of rules for each community is depicted in the barplot of Figure 10.13. Note that some communities ($\{4, 9, 13, 15, 16\}$) have a limited number of rules, and remarkably, these few rules (rules $0, 25, 39$) are consistently frequent across all of these communities. Specifically, these rules capture neighborhood expansions of a single source which occur in a week only. In short, *nodes belonging to these communities tend to expand their contact list in a week, i.e. they are intensive senders*. It is worth noting that this kind of specific behavior characterizing only a few communities has not emerged from a global analysis of the GER profiles. Still, it has been made possible by the extraction of the community GER profiles.

In analyzing the **Stack Overflow** dataset, we apply the EvoMine algorithm to individual node-induced community subgraphs, utilizing a support threshold set at $\frac{4}{20} \cdot max\_sup$. This methodology produces consistent rule sets within each community, although slight fluctuations occur, with the number of rules varying between 9 and 4. The GER profiles of these communities are reported in Figure 10.14 with the same visualization of the previous datasets,

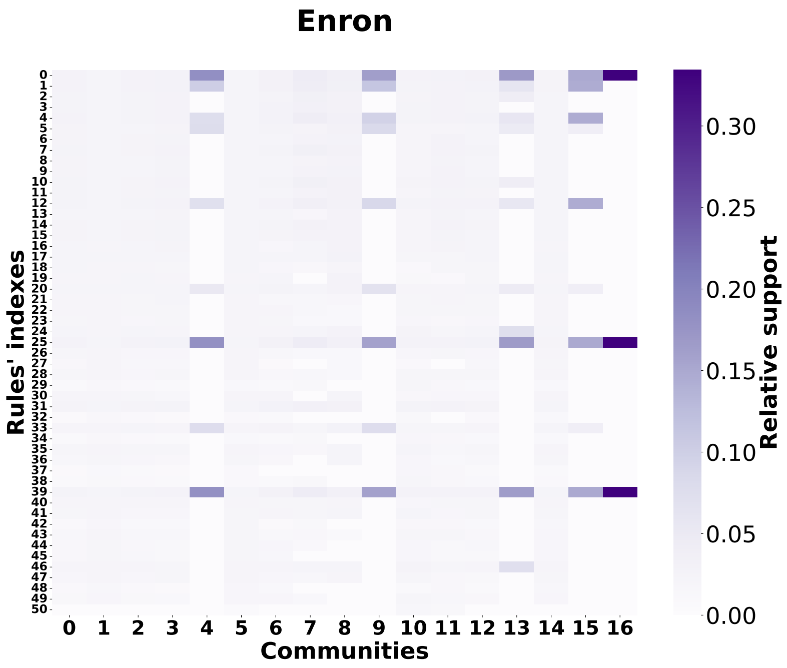


Figure 10.12: Communities GER profiles. *Visualization of the GER profiles of the communities in the Enron network. Rows correspond to evolution rules, and columns correspond to the communities.*

where lighter cells indicate less frequent rules within community subgraphs. Additionally, in Figure 10.15, we observe variations in the number of evolution rules across communities. Notably, communities sharing the same number of rules may exhibit distinct GER profiles. For instance, while communities 1 and 2 showcase a homogeneous distribution across the initial rules, community 7 demonstrates concentration on the first and third rules, with less uniformity among others. Overall, despite nuanced differences, the *GER profiles of Stack Overflow network communities exhibit a consistency similar to the DBLP case*, contrasting with the Sarafu and Enron datasets. Indeed, in all the communities is prevalent an *evolutionary mechanism related to the nature of the service, i.e. the evolution is mainly driven by accounts that reply to other accounts who sporadically reply back (reciprocal links).*

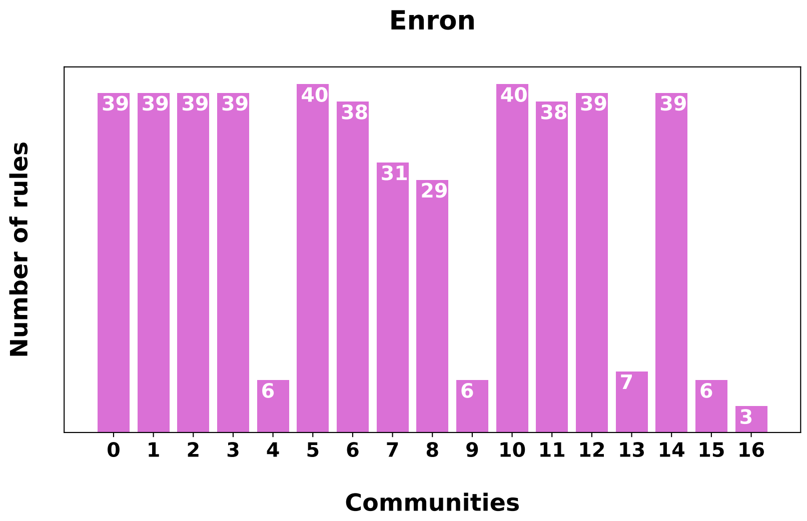


Figure 10.13: *Number of evolution rules for each community extracted from the Enron network.*

To sum up, in Sarafu and Enron networks the identification of evolution patterns characterizing communities through graph evolution rules and their vector-based representation by GER profiles clearly indicate that *the evolution process is far from being homogeneous over the entire network, rather each community or small group of communities has its specific evolution patterns.* On the contrary, in the DBLP and Stack Overflow networks, communities follow similar evolution patterns. In general, these observations further emphasize the importance of a mesoscopic perspective in the analysis of complex networks and stress the importance of unfolding the evolution of mesoscopic structures such as communities, groups, or modules to get a deeper comprehension of the mechanisms leading the growth of dynamic networks.

Since the community graph of the Enron network has shown a peculiar core-periphery configuration, we focus on the Enron case showing how GERs and GER profiles can characterize the dynamics of the network periphery. In fact, we have already observed that there are five communities (4,9,13,15 and 16) with a significantly low number of rules - less than 7 - and these few rules are common to all these communities. Specifically, the most frequent in all the five communities are GERs 0, 25 and 39, depicted in the first row of

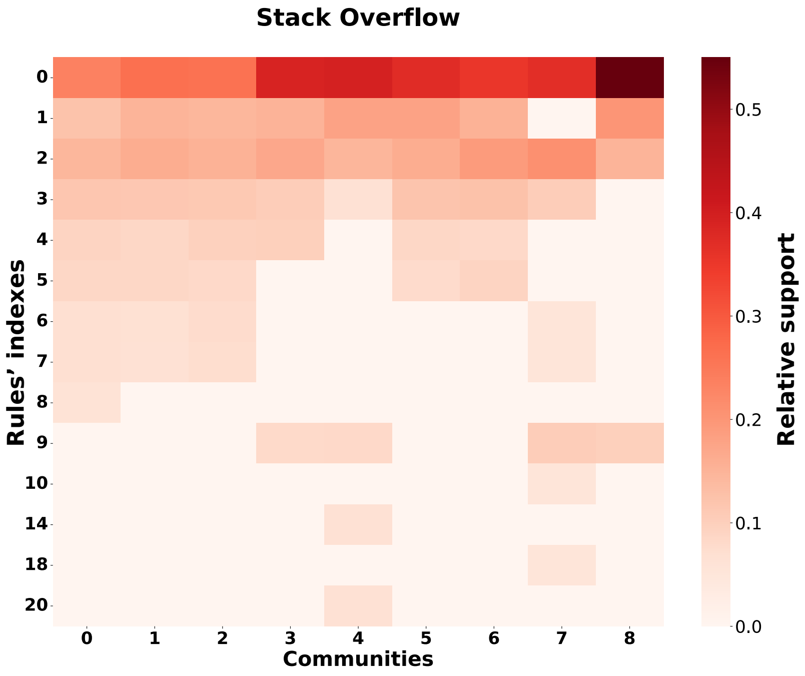


Figure 10.14: Communities GER profiles. *Visualization of the GER profiles of the communities in the Stack Overflow network. Rows correspond to evolution rules, and columns correspond to the communities.*

Figure 10.16 (from (a) to (c)). In the figure, we also report a set of rules that are less frequent but present in almost all five communities, i.e. GERs 5, 20 and 33 (second row in Figure 10.16 - from (d) to (e)). Note that the rules in Figure 10.16 represent the postcondition of the rule, while the edge(s) added to the precondition is (are) the one(s) highlighted in green. If all edges in a rule are green (with a timestamp equal to t_1), then the rule describes the formation of 3 edges in the same timestamp, which are not present in the previous one. All these rules exhibit an expanding behavior: all edges start from a single node. The difference between the first row (most frequent) and the second one is given by the time: while those in the first row record edges created in the second timestamp, i.e. they capture a faster formation of the postcondition, the second row includes rules where the evolution is slower and it happens in two consecutive moments - in this case two weeks. In general, the GER

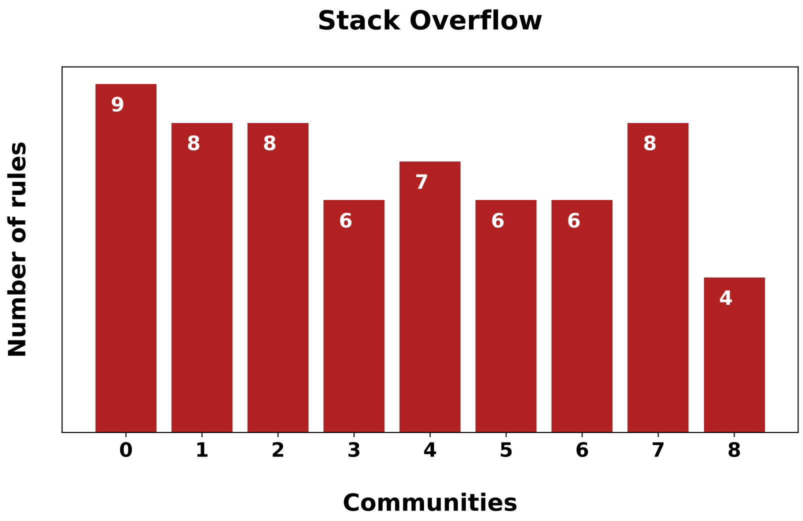


Figure 10.15: *Number of evolution rules for each community extracted from the Stack Overflow network.*

analysis indicates an expansive behavior of the periphery dominated by fast expansive rules.

10.4.3 Evolution of neighboring communities

After showing the heterogeneity of the evolution of the four dynamic networks, we turn our attention to addressing the third research question: do interconnected modules, specifically closely-knit communities, exhibit similar evolution patterns or do they evolve independently? To address this question, we examine the community graph alongside the total graph, and subsequently, we compare them with the community profiles shown in the previous section.

In the case of the **Sarafu** network, in Figure 10.17a we display the relationships among the communities of the graph through a more compact representation given by the community graph. In a community graph, each node is a community c_i, and there is an edge (c_1, c_2, w) between communities c_1 and c_2 if there is at least an edge that connects a node in c_1 to a node in c_2. The weight w represents the fractions of existing links across c_1 and c_2 over all the possible links across communities c_1 and c_2, i.e. $w = \frac{|\{(u,v) \in G | (u \in c_1 \wedge v \in c_2) \vee (u \in c_2 \wedge v \in c_1)\}|}{|c_1| \cdot |c_2|}$,

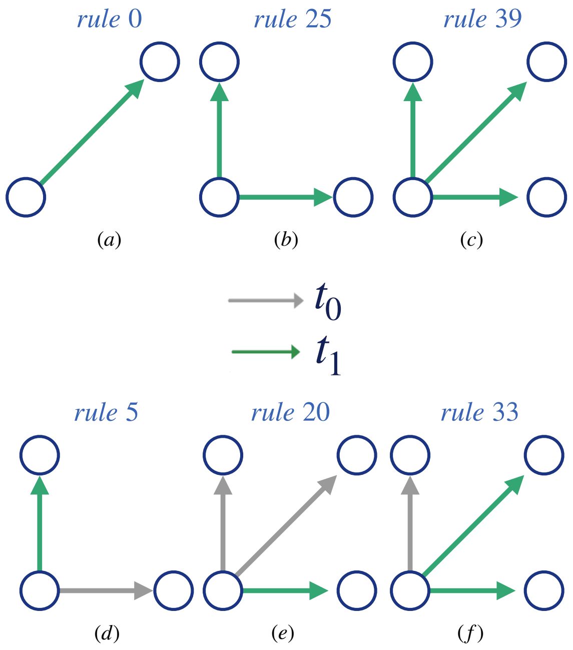


Figure 10.16: *Six GERs characterizing the communities in the Enron networks. Edges belonging to the precondition graph are grey and edges inserted in the postcondition graph are green.*

with G being the general graph, and $|c|$ denotes the number of nodes in a community c. In the graph visualization, edges are colored based on their weights (percentage of cross-community links), nodes are sized based on the weighted degree and they are colored based on the community they represent. Observing the community plot in Figure 10.17a, we can discern three groups of community: two closed within themselves and the third one composed of communities more isolated. In Figure 10.17a these sets are highlighted with shapes of different colors: the pink and blue shapes include two groups of communities that are more connected within each other, except for community 19 which stands in the middle. The orange group includes the most isolated communities, less connected to all the remaining communities (weak connections are indicated by the lighter green color of the edges). Specifically, the first group (pink shape) is composed of $\{12, 14, 5, 16, 10, 0, 3, 2, 6, 19\}$, the second (blue shape) is made by $\{15, 13, 11, 17, 9, 1, 19\}$ and the remaining communities $\{4, 7, 8, 18\}$ (orange shape) are less interconnected and act as connections between the two previous main groups. Moreover, in Figure 10.17b, c and d we report the community GER profiles of the communities included in the pink, blue and orange sets, respectively, and we fixed the range of values of the colormap to have a uniform association between level of darkness of the cell and magnitude of support. By observing the profiles we notice that the partition of communities is reflected in their community GER profiles. In fact, while the blue group includes only profiles uniformly distributed over rules, the pink one contains GER profiles with peaks of support concentrated in a small set of rules (the upper segment). Finally, the isolated set of communities does not present GER profiles with a clear common trait. In the case of Sarafu, *close and tight-knit sets of communities follow similar evolution dynamics*, as captured by GER profiles.

Moving on to the DBLP network, Figure 10.18 shows how nodes are spread among communities and the relationships among them as expressed by the community graph. In DBLP, communities are generally more connected to one another, as we can see from the almost uniform size of nodes in Figure 10.18b. This is also reflected in the community GER profiles depicted in Figure 10.10. The frequencies of the rules are less homogeneous (even in the number of rules) but among communities the GER profiles are similar. However, we can still see an interplay between the closeness of a community to the other ones and the similarity of their GER profiles. For instance, community 8 has one of the GER profiles that differs the most from the other ones, and, indeed, it is one of the communities on the edge of the graph, with weak connections with the other ones. On the contrary community 4, whose GER profile represents the median behavior, is one of the central communities (see Figure 10.18a), with one of the highest weighted degrees.

Then, in Figure 10.19 we depict the community information of the Enron network adopting the community graph representation. In the case of the Enron network, the community GER profiles (see Figure 10.12) have highlighted the presence of a typical behavior - more homogeneous among all the rules - and a peculiar trait, with a concentration of fewer rules. This is also reflected in the core-periphery-like configuration as highlighted by the shapes drawn in Figure 10.19a. The orange shape includes the central communities, which are more connected to each other, while the grey shape captures the peripherical communities. This trait is strictly linked to the community GER profiles displayed in Figure 10.19b and c. In fact, the GER profiles of the central communities (Figure 10.19c) are the ones with a standard behavior, with a uniform distribution over the evolution rules. On the other hand, the profiles of the peripherical group, i.e. communities 15, 16, 13, 9, are composed of GER

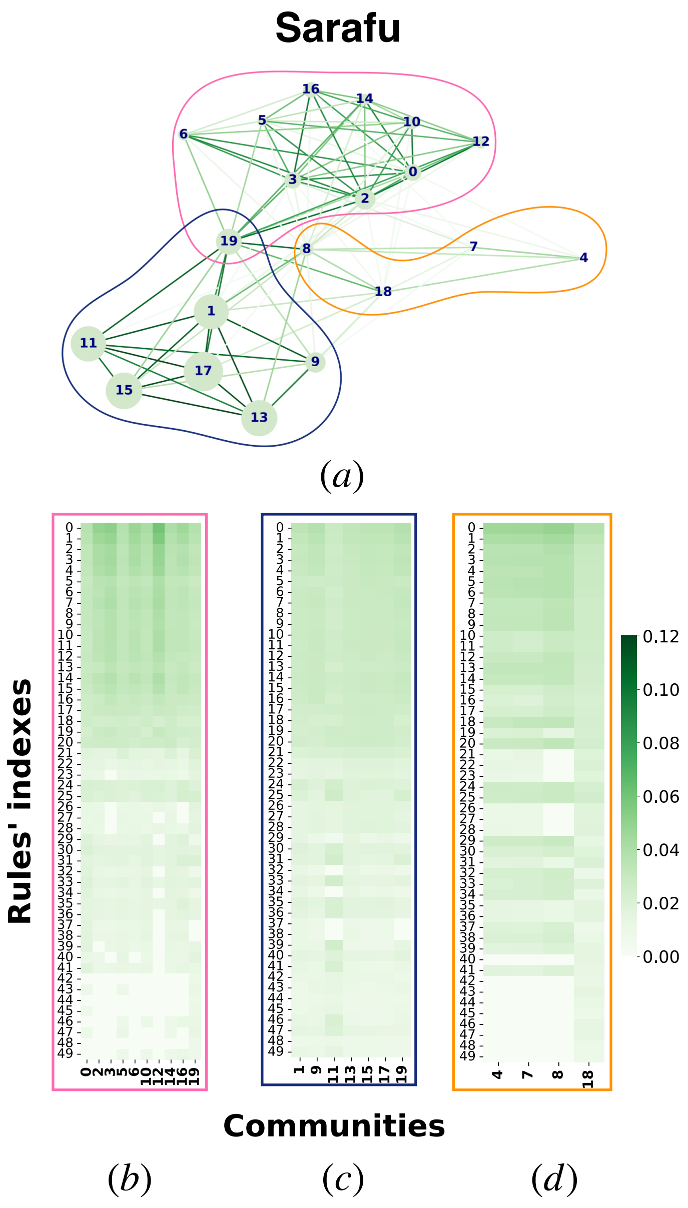


Figure 10.17: *Rappresentation of the community graph of the Sarafu network (a) where communities are groups in three sets (orange, blue and pink shapes). For each set, we report the concatenation of the GER profiles of the communities belonging to ((b), (c) and (d)).*

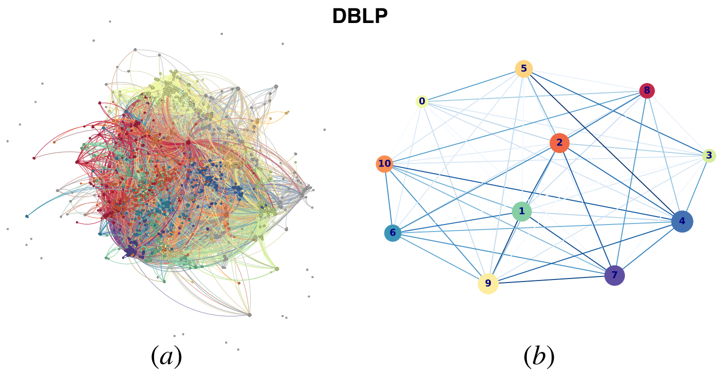
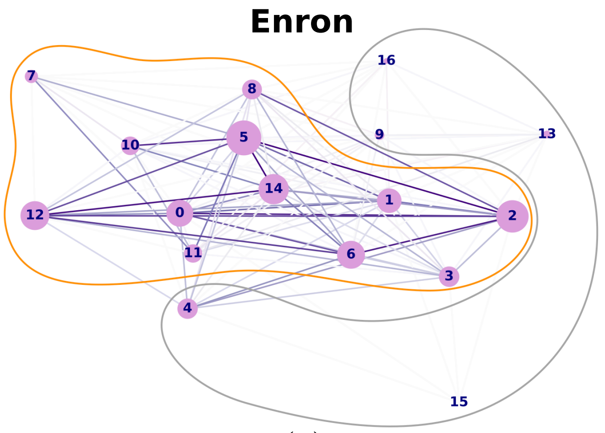


Figure 10.18: *In (a) visualization of the DBLP network where nodes are colored according to the community they belong to. In (b) the community graph of the DBLP network, restricted to the most representative communities.*

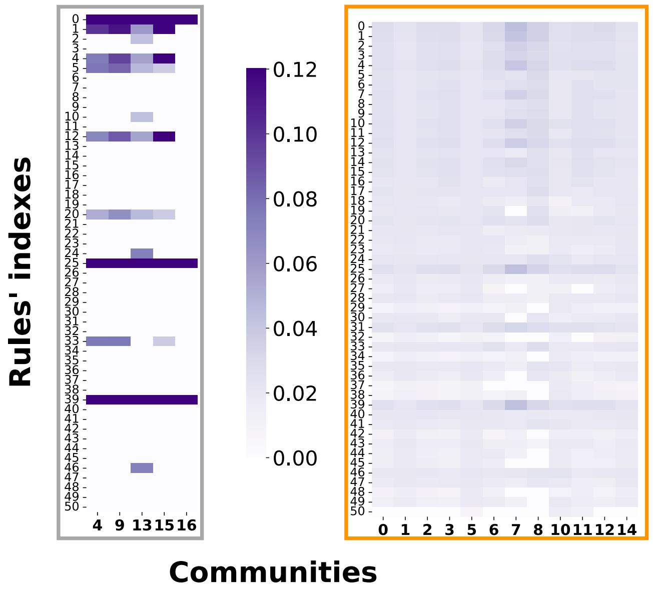
profiles where only a small fraction of rules are frequent, i.e. the set depicted in Figure 10.16b. Even in the case of Enron, *it is quite evident the relation between how close and connected communities are and the evolution dynamics they follow.*

Finally, the community plot of the Stack Overflow networks shows a community graph with a big group of nodes highly connected. As in the DBLP graph, it is reflected by the homogeneous GER profiles. However, an interesting interplay emerges between a community's proximity to others and the similarity of their GER profiles. For example, community 7, distinguished by its distinct GER profile, appears as the most isolated within the community graph. Conversely, communities 0 and 2, characterized by highly similar GER profiles, are also closely connected within the network of communities. This observation suggests a relationship between a community's structural connectivity and the coherence of its GER profile.

To summarize, the answer to the third research question is not as sharp as for the previous questions, however, it is quite evident that *close or tight-knit sets of communities evolve following similar evolution patterns*. This common trend also impacts the DBLP and Stack Overflow cases, where community GER profiles are more similar because of the compact structure of their modules.



(*a*)



(*b*) (*c*)

Figure 10.19: *Rappresentation of the community graph of the Enron network (a) where communities are groups in two sets. For each set, we report the concatenation of the GER profiles of the communities belonging to ((b) and (c)).*

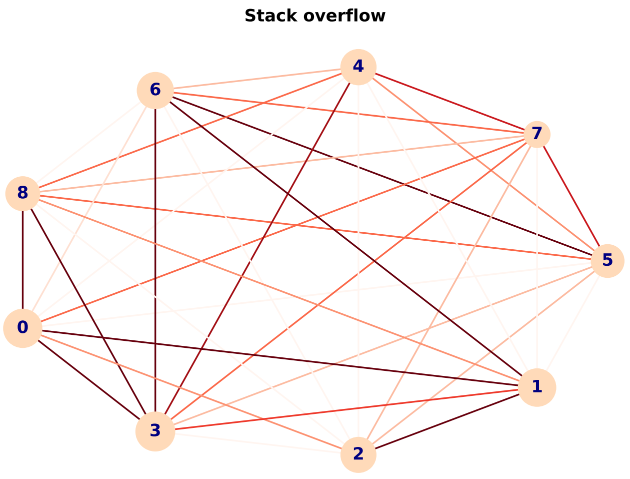


Figure 10.20: *Rappresentation of the community graph of the Stack Overflow network*

10.5 Conclusions

In conclusion, our study examined different dynamic networks through the lens of graph evolution rules (GERs), shedding light on their intricate dynamics. Our work highlights the significance of GERs as a valuable analytical tool for unraveling the complex evolution of dynamic networks and underscores the importance of considering modular structures in network analysis. In particular, we show the effectiveness of an approach based on temporal subgraph mining in discovering evolving subgraph patterns and localizing interesting evolutionary patterns in modular networks. Through this tool, we delved into diverse networked systems, revealing both global and mesoscopic patterns of network evolution. From a global perspective, we observed remarkable similarities in the growth patterns of Enron and Sarafu, as reflected in GER profiles, while larger networks, such as DBLP and Stack Overflow, behaved and evolved differently. These observations stress the effectiveness of GER profiles, as a tool to represent and easily compare the evolution of temporal networks. Moreover, our mesoscopic analysis unveiled novel and interesting evolutionary patterns leading to the formation of current communities. In fact, through the community GER profiles, we assess the non-homogeneous nature of community evolution, with each community or small group exhibiting specific evolutionary patterns.

Furthermore, GER profiles not only allow us to disentangle the different evolutionary patterns that led to the formation of the different communities, but they even support the assessment of similarity among communities close in the network, regarding their evolutionary patterns. This has made evident the influence of interconnected sets of communities on similar evolution trends, i.e. close communities evolve similarly.

This insight into the interplay between modular structures and evolving network dynamics is crucial for a deeper understanding of these complex systems. As we move forward, future research in this domain may explore advanced GER mining techniques to uncover even subtler patterns and mechanisms underlying network evolution. Additionally, investigating the application of our approach to a broader spectrum of dynamic networks across various domains could provide valuable insights into their shared and distinctive characteristics. Indeed, our methodological pipeline can be applied to any temporal network that grows over time and is characterized by a current modular structure. We also showed that the method can reasonably scale to large temporal networks, however, since the framework is modular, there is space for improving the computational complexity by adopting more efficient approaches for extracting graph evolution rules or different algorithms for community detection. Furthermore, the development of novel visualization tools and methods for GER-based analysis could enhance our ability to comprehend the intricate dynamics of networks. By uncovering the principles that govern the evolution of such networks, researchers and practitioners can make informed decisions for network management, optimization, and adaptation.

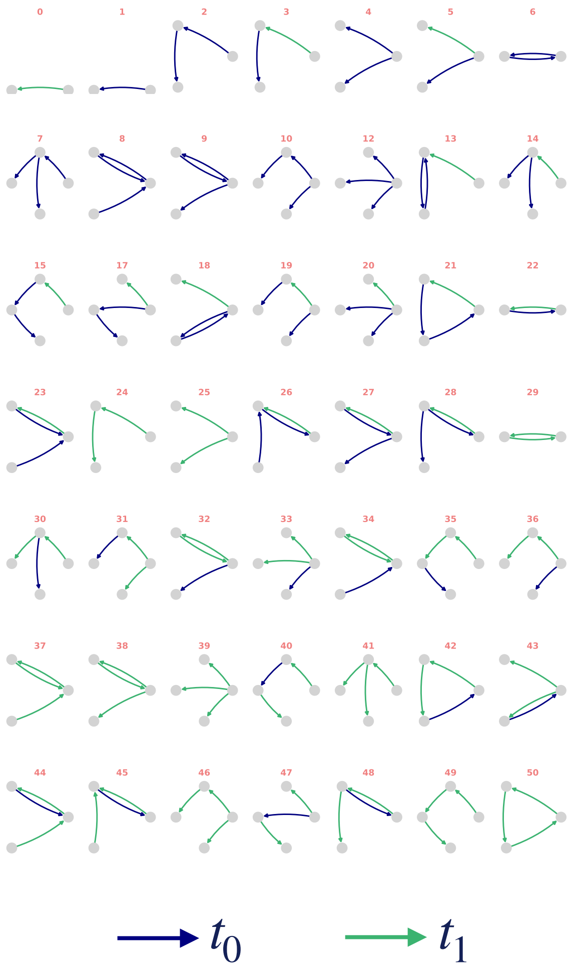


Figure 10.21: Graph evolution rules. *Set of all the graph evolution rules associated with their unique identifier (on top of each graph). Blue links correspond to the precondition, while green links indicate new links added to the postcondition.*

Part III

Graph Evolution rules

The third part of this thesis expands the scope of graph evolution rules, enriching the GERANIO framework through the introduction of the TULIP algorithm. We dedicate a different part of the thesis to this topic alone to stress the difference between stand alone rules and general graph evolution rules offered by the TULIP algorithm. While stand-alone rules offer valuable insights in specific scenarios, they often struggle to catch the intricate dynamics of network evolution in their entirety. The TULIP algorithm addresses this limitation by building upon the foundational terminology and concepts outlined in the background section (Chapter 4). This novel approach, to the best of our knowledge, represents a significant advancement in the field. Unlike conventional algorithms that focus on individual, stand-alone rules, our method, TULIP, can capture and quantify all possible evolutions of a subgraph, along with their associated probabilities. This comprehensive approach allows for a more nuanced and accurate representation of complex network dynamics, providing researchers with a powerful tool to analyze and predict the behavior of evolving graph structures across various domains. By considering the full spectrum of potential changes and their likelihoods, TULIP offers a more holistic view of graph evolution, potentially uncovering patterns and relationships that might be overlooked by more restrictive methodologies.

Chapter 11

TULIP

11.1 Introduction

This chapter presents TULIP (Temporal subgraphs for evolutionary profiling), a novel algorithm that represents a significant advancement in the field of temporal network analysis. TULIP builds upon our previously proposed framework for stand-alone rules, which addressed critical limitations such as the absence of a null model and a canonical form. However, TULIP goes beyond these improvements, offering a more comprehensive approach to understanding network dynamics. The key innovation of TULIP lies in its ability to capture all possible evolution patterns of subgraphs within a temporal network, along with their corresponding probabilities. This holistic approach provides a nuanced and complete picture of network behavior over time. The algorithm operates in two main phases: static pattern enumeration and temporal evolution mining. In the first phase, TULIP identifies and enumerates occurrences of predefined static patterns within the network. The second phase incorporates temporal information to mine the various ways these subgraphs evolve, revealing the dynamic nature of the network's structure. The outcome of this process is the construction of evolution profiles for each starting temporal subgraph. These profiles encapsulate all observed evolutions and their associated probability distributions, offering a multi-dimensional view of the network's temporal behavior. This comprehensive analysis enables researchers to gain deeper insights into the underlying mechanisms driving network evolution across various domains. To demonstrate the effectiveness and versatility of TULIP, we applied the algorithm to five diverse datasets, encompassing both Web2 and Web3 scenarios. These datasets include communication networks, financial networks, and trust networks. Notably, our results revealed intriguing patterns in evolutionary behavior across these different network types. Interestingly, we observed similar evolutionary behaviors in the Steemit and CryptoKitties datasets, both Web3-based networks, despite one being financial and the other representing social interactions. In contrast, the Sarafu dataset, another financial network from a Web3 platform, exhibited behavior more closely aligned with UC-Social and Bitcoin-Alpha, two traditional social networks. These findings highlight the potential of TULIP to uncover unexpected similarities and differences in network evolution across diverse domains and technological paradigms. In the following sections, we will delve into the technical details of the TULIP algorithm, discuss its implementation, and provide an in-depth analysis of the results obtained from these varied datasets. Through this exploration, we aim to demonstrate how TULIP opens new avenues for understanding complex systems, from social networks to financial interactions, in both traditional and emerging digital landscapes.

11.2 Methodology

In this work, we propose a novel algorithm to mine graph evolution rules that follows the pipeline depicted in Figure 11.1. The inputs are the temporal network (modeled as a growing projection, as described in the taxonomy of Chapter 3) and a starting set of static patterns. Using a subgraph isomorphism algorithm, the static occurrences of the patterns are enumerated. These occurrences are divided into regular-sized chunks and fed into the temporal pipeline. For each static occurrence, the temporal pipeline extracts the temporal edges and the evolutions. After the mining part, the temporal patterns are mapped into relative temporal patterns and classified using a canonical form algorithm. The categorized patterns with the associated frequency are used to create an vector-based profile that serves as evolutionary footprint of a network. The following sections will provide details on each part of the TULIP algorithm pipeline.

11.2.1 Enumeration of static patterns

The algorithm begins with a set of predefined starting subgraphs. This initial set can be defined in various ways, such as including all subgraphs with a maximum of three edges or selecting a smaller subset with specific significance in the field of application. For example, in a financial network analysis, one might focus on potential cycles by starting with open structures to evaluate how frequently they close. The selected static structures are then enumerated within

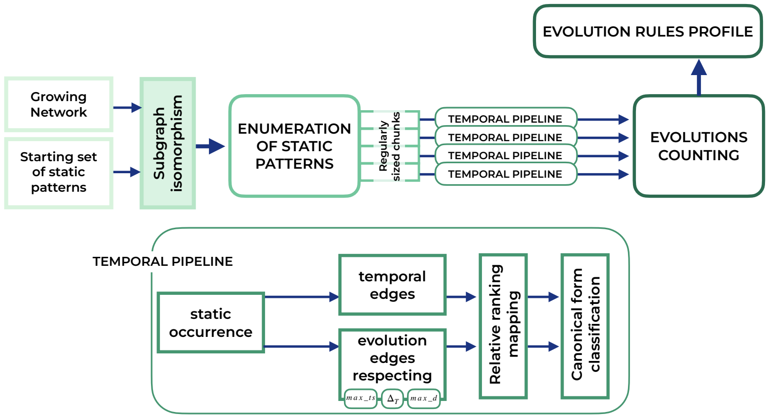


Figure 11.1: Pipeline of the Tulip algorithm*. The algorithm begins by enumerating all occurrences of a given set of static patterns. The results are then divided into chunks, and through the temporal pipeline, Tulip extracts pairs of evolutions in the form* $(pre, post)$*. These evolutions are subsequently counted, and the evolutionary profile is constructed*

the growing network using the VF2 subgraph isomorphism algorithm [41]. This algorithm, in its igraph [160] implementation, efficiently identifies all instances of the predefined subgraphs within the larger network structure. This part of the algorithm was parallelized on the starting subgraph, as it provided the highest potential for parallelization within the process.

11.2.2 Temporal pipeline

Once we obtain the list of edges forming the starting static patterns, we begin with the temporal part (Algorithm 4). This process is executed in parallel for each enumeration, regardless of the originating subgraph. The parallelization is applied to bunches of occurrences, striking a balance between time savings from parallelization and resource efficiency. While processing individual occurrences is quick, the sheer volume—millions of occurrences—necessitates this approach. Algorithm 4 is applied to each *chunk* of occurrences. For every static occurrence, we first extract the timestamps associated with the static edges (line 3 of Algorithm 4). We then reconstruct the temporal subgraph's evolution, this happens only if the starting temporal subgraph is not at the temporal span's end, which would preclude further evolution. Edges surrounding the starting subgraph are included in the evolution (post condition) if they satisfy three parameters constraints: max_{ts}, Δ_t, and max_d. The max_d parameter limits the influence of hubs in evolution counting, representing the maximum number of incoming and outgoing neighbors each node of the initial subgraph can have. If the number of incoming or outgoing edges from a node u in the subgraph exceeds max_d, only the first max_d are considered (lines 6 : 8 and 16 : 18 of Algorithm 5 for outgoing and ongoing edges respectively). Additionally, edges are included in the evolutions if they occur within the interval $[max_t s, max_t s + \Delta_t]$. The algorithm concludes by obtaining the canonical representation for both the precondition (starting subgraph) and the post-condition (starting subgraph with evolutions), based on the Bliss algorithm [36].

Algorithm 4 Temporal pipeline.

```
Input: $G, chunk, max\_d, \Delta_t, T$
Output: rules
1: $rules = ()$
2: for $edges \in chunk$ do
3:     $edges\_temporal = \{(u, v, G[u][v][ts]) \forall (u, v) \in edges\}$
4:     $max\_ts = max(\{x_2 \forall x \in edges\_temporal\})$
5:     if $max\_ts < T - \Delta_t x$ then ▷ Avoid occurrences at the end of the temporal span
6:         $edges\_temporal = relative\_ranking(edges\_temporal)$ ▷ Classify edges of pre-condition
7:         $label\_pre = get\_canonical(to\_layered(edges\_temporal))$
8:         $evolutions = get\_evolutions(edges\_temporal, max\_ts, max\_d, \Delta_t)$
9:         if $len(evolutions) > 0$ then ▷ Classify edges of post-condition, if any
10:            $evolutions = relative\_ranking(edges\_temporal + evolutions)$
11:            $label\_post = get\_canonical(to\_layered(evolutions))$
12:            $rules.append((label\_pre, label\_post))$
13:        else
14:            $rules.append(label\_pre)$
15:        end if
16:    end if
17: end for
18: return rules
```

Algorithm 5 Evolutions.

```
Input: $G, edges\_temporal, max\_d, max\_ts, \Delta_t$
Output: evolutions
1: $nodes = \{u \forall (u,v,t) \in edges\_temporal\} \cup \{v \forall (u,v,t) \in edges\_temporal\}$
2: $evolutions = list()$
3: for $n \in nodes$ do
4:     $out\_neighs = G.successors(n)$                ▷ get outgoing evolutions
5:     $out\_edges = \{(n, v, G[n][v][ts]) \forall v \in out\_neighs\}$
6:     if $len(out\_edges) > max\_d$ then
7:         $out\_edges = sorted(out\_edges)[: max\_d]$
8:     end if
9:     for $(u, v, ts) \in out\_edges$ do
10:        if $max\_ts < ts <= max\_ts + \Delta_t$ then
11:            $evolutions.append((u, v, ts))$
12:        end if
13:    end for
14:    $in\_neighs = G.predecessor(n)$                ▷ get ingoing evolutions
15:    $in\_edges = \{(v, n, G[v][n][ts]) \forall v \in in\_neighs\}$
16:    if $len(in\_edges) > max\_d$ then
17:        $in\_edges = sorted(in\_edges)[: max\_d]$
18:    end if
19:    for $(u, v, ts) \in in\_edges$ do
20:        if $max\_ts < ts <= max\_ts + \Delta_t$ then
21:            $evolutions.append((u, v, ts))$
22:        end if
23:    end for
24: end for
25: return evolutions
```

11.2.3 Canonical classification

The canonical classification process, which assigns a unique identifier to each isomorphic subgraph for universal applicability, involves several steps. First, we perform a relative-ranking mapping where UNIX timestamps are transformed into incremental integers starting from zero. This mapping is "relative" because timestamps always begin at zero and "ranking" because the temporal order, not the absolute time differences, is significant. Figure 11.2 shows an example of three different temporal subgraphs mapped into the same relative-ranking mapped subgraph because they all present the same temporal order.

We transform the temporal relative-ranking mapped subgraph into a multilayer graph without edge attributes to meet bliss algorithm requirements.

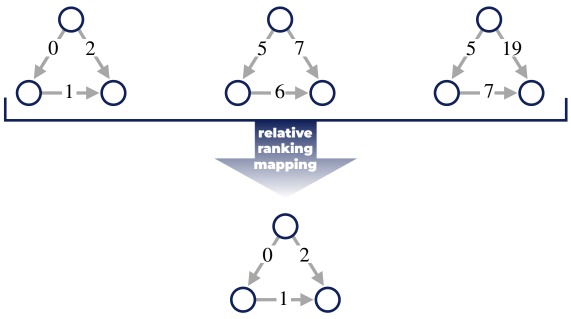


Figure 11.2: *Example of three temporal patterns that are mapped into the same relative-ranking mapped pattern.*

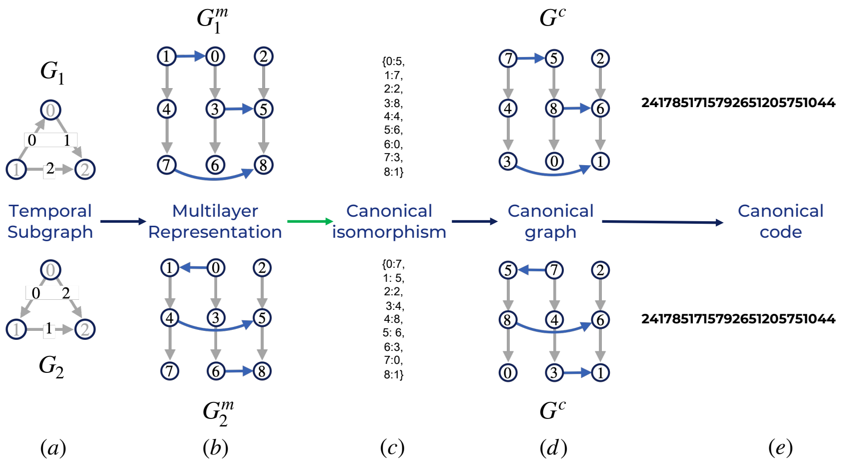


Figure 11.3: Example of the canonical coding process. *Temporal subgraphs in (a) are first transformed into their multilayer representations in (b). Then, the bliss algorithm is applied, obtaining the canonical isomorphism in (c) that is used to obtain the canonical graph in (d). From the canonical graphs we obtain the canonical code in (e).*

Each layer represents a timestamp, with nodes duplicated across layers and edges placed in their corresponding timestamp layer. Layers are connected by pillar edges linking nodes of the same entity. This process preserves temporal information while creating a bliss-compatible structure. The following paragraph will offer a detailed examples based on Figure 11.3. The temporal subgraphs in Figure 11.3a are transformed into a multilayer representation in Figure 11.3b. Nodes are duplicated $max_{ts} + 1$ times, creating $max_{ts} + 1$ layers. At each layer $l = 0, 1, \ldots, max_{ts}$, a node's id becomes $n + (n\_nodes * l)$. For example, in Figure 11.3b, graph G_1^m has 3 layers ($max_{ts} = 2$), and node 2 becomes node 8 in the third layer. Two types of edges exist in this representation: pillar edges (shown in gray) connect the same node across multiple layers, while actual temporal subgraph edges are assigned to layers corresponding to their timestamps. For instance, the temporal edge $(1, 2, ts = 2)$ of G_1 becomes edge $(7, 8)$ on the third layer in G_1^m. This multilayer representation is then processed by the bliss algorithm, producing the canonical isomorphism shown in Figure 11.3c. From this, we reconstruct the canonical graph G^c in Figure 11.3d. It's important to note that Figure 11.3 illustrates the example of G_1 and G_2 that are two isomorphic temporal subgraphs, however, the canonical isomorphism itself is not the same for the two isomorphic subgraphs. What corresponds is the canonical graph. Finally, we decode the obtained canonical graph. We take the adjacency matrix, concatenate the rows with a 1 as the prefix, and decode the resulting binary string into an integer. This method ensures we obtain the same integer code (which is more compact than storing the binary string) for every isomorphic temporal graph, as shown in Figure 11.3e.

11.2.4 Counting and profile

The temporal pipeline generates a list EV of tuples $(pre, post)$, representing the preconditions and their subsequent evolutions. To analyze the frequency of these evolutions, we employ two counting mechanisms. First, we count the occurrences of preconditions alone, establishing a function $counter_p : P \to \mathcal{N}$ that maps the set of preconditions P to their respective frequencies across the entire network evolution. This provides a baseline for understanding the temporal structure of the network. Second, we count the occurrences of complete evolutions using a function $counter_e : C \to \mathcal{N}$, which maps couples $(pre_i, post_j) \in C$ to their frequency of occurrence. These counting functions enable the construction of profiles for both preconditions and evolutions. The precondition profile vp is a vector where each element $vp_i = counter_p(pre_i)/|EV|$, representing the normalized frequency of each precondition. For evolutions, we create a collection of profiles, one for each precondition. Each evolution profile $ve(pre_i)$ corresponds to a specific precondition pre_i, with elements $ve(pre_i)_j = countere(pre_i, post_j)/|EV|$, representing the normalized frequency of each possible evolution from that precondition. This comprehensive profiling approach allows for a detailed analysis of both the temporal network structure and its dynamic evolution patterns.

11.2.5 Case Studies

We applied our methodology to five diverse datasets modeling different types of interactions, ranging from communication networks to financial networks on Web3 platforms. The first three datasets—UC-social, Bitcoin Alpha, and Sarafu—are described and modeled in detail in Chapter 6. While the Steemit and Cryptokitties datasets are also mentioned in Chapter 6, their modeling requires further elaboration, which we provide here. The Steemit dataset encompasses a subset of follow operations on the Steemit platform, specifically those occurring between December 6th, 2016, and March 6th, 2017, totaling 302,669 follow operations. We modeled this data as a growing projection $G_[(1,T)]^{GP} = (V, E, f)$, where f is an edge-labeling function assigning to each edge (i, j) the first time user i followed user j. This modeling resulted in a growing network comprising 23,493 nodes and 290,801 edges. The Cryptokitties dataset, derived from Nadini et al. [104], represents a subset of transactions within the Cryptokitties market/game. For our analysis, we considered all transactions recorded in this subset and modeled them into a growing projection $G_[(1,T)]^{GP} = (V, E, f)$, following the same definition as the other networks. The resulting network consists of 99,984 nodes and 481,540 edges. Table 11.1 presents the number of edges and nodes for each dataset in its initial columns. It's important to note that for the mining phase of our analysis, we initially considered timestamps in Unix time format, which measures time as the number of non-leap seconds elapsed since 00:00:00 UTC on January 1, 1970 (the Unix epoch). However, during the canonical representation phase, we transformed these Unix timestamps into relative-ranking timestamps, thereby losing the original Unix time information. This diverse set of datasets allows us to test our methodology across various network types and interaction patterns, providing a comprehensive view of its applicability and effectiveness in different contexts.

11.3 Results and Discussion

In this section, we showcase the results of the algorithm described in 11.2 applied to the five different networks, first highlighting the insights we can get from the pre-condition profiles and then from the evolutions too. In Table

11.1, we show the computation time of the algorithm — including the running time for the enumeration of static patterns as well as the temporal pipeline. The experiments were conducted on a Dell PowerEdge T620 server with dual Intel Xeon CPUs (2.10GHz, 16 cores, 32 threads) and 376GB RAM[1].

Dataset	$\|N\|$	$\|E\|$	max_d	Δ_t	**PartI**	**PartII**
UC-social	1,899	20,211	10	9h19m16s	17s	15m
Bitcoin Alpha	3683	22,650	6	5days	20s	14m30
Sarafu	40,343	143,239	3	1day, 5h25m42s	1m30	30m
Steemit	23,493	290,801	12	9s	1h37	45h
Cryptokitties	99,984	481,540	4	17m6s	30m	7h30

Table 11.1: *Overview of the parameters setted for the five datasets, with computation times and networks' size. max_d is chosen as an approximation of the graph degree, while Δ_t corresponds to the median of inter-link time.* ***PartI*** *column corresponds to the computation time of the static enumeration phase, while* ***PartII*** *stands for the temporal pipeline.*

11.3.1 Pre profile

The first outcome of this study is the generation of pre-condition profiles for each analyzed network, as detailed in Chapter 11.2. These profiles reveal a maximum of 325 pre-conditions, corresponding to all possible temporal subgraphs with 2 or 3 nodes. While larger networks exhibit all 325 pre-conditions, the Bitcoin network shows 323, and UC-social datasets display 287. Despite this wide range of identified pre-conditions, their frequency distribution is heavily skewed towards simpler structures. Figure 11.4 illustrates the profiles of the top 20 pre-conditions, ranked by their frequency in the UC-social profile. A striking observation is the distinct profiles of Cryptokitties and Steemit compared to other networks. This distinction is intriguing as both are Web3 platforms, setting them apart from traditional networks. Interestingly, Sarafu, despite also being a Web3 platform, shows similarities to conventional networks, possibly due to its humanitarian/social nature. The dominant patterns reveal further insights into network structures. Cryptokitties and Steemit are highly concentrated on pre-condition $id = 0$, suggesting network growth primarily through node expansion without reciprocation. In contrast, Sarafu is

[1] The algorithm has been implemented in Python.

dominated by pre-condition $id = 13321$, indicating frequent user interactions reflecting money exchanges. UC Social and Bitcoin exhibit similar profiles, with a higher concentration on the first 4 patterns (no reciprocal edges) and significant presence in the next 8 patterns (open triads with one reciprocal edge) The analysis applies normalization to the entire set of pre-conditions, as explained in Chapter 11.2. The visualization of Figure 11.4 is limited to the top 20 patterns (depicted in Figure 11.5) due to negligible percentages beyond this point. This initial analysis of pre-conditions provides insights into the structural differences between various network types, particularly highlighting the unique characteristics of Web3 platforms and traditional social networks. Moreover, by examining the preconditions alone, we can gain a glimpse into the temporal structure of each network, allowing us to characterize its dynamic behavior.

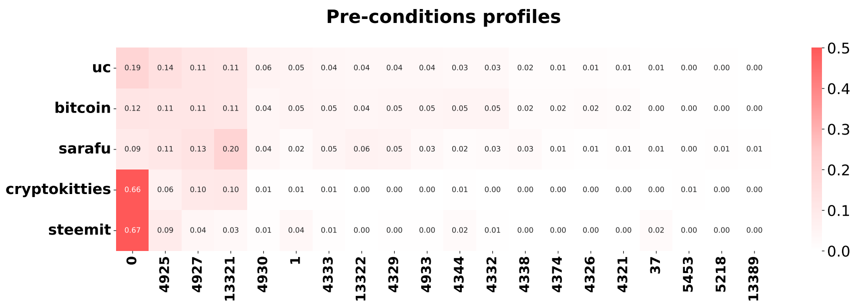


Figure 11.4: *Heatmap showing the profile of pre-conditions for the 5 datasets. The visualization is limited at the 20 most frequent patterns due to negligible percentages beyond this point. Each line represents a profile for a dataset, i.e. a probability distribution over the frequency of pre-conditions.*

11.3.2 Evolutionary profile

Our analysis now shifts to examining the actual evolution of the networks. We begin by considering some quantitative results presented in Table 11.2. This table displays, for each dataset, the number of preconditions, the number of rules (defined as couples of $(pre_i, post_j)$), and the median number of post-conditions for each precondition. The latter metric refers to the main feature of the TULIP algorithm's that offers an evolutionary profile, offering

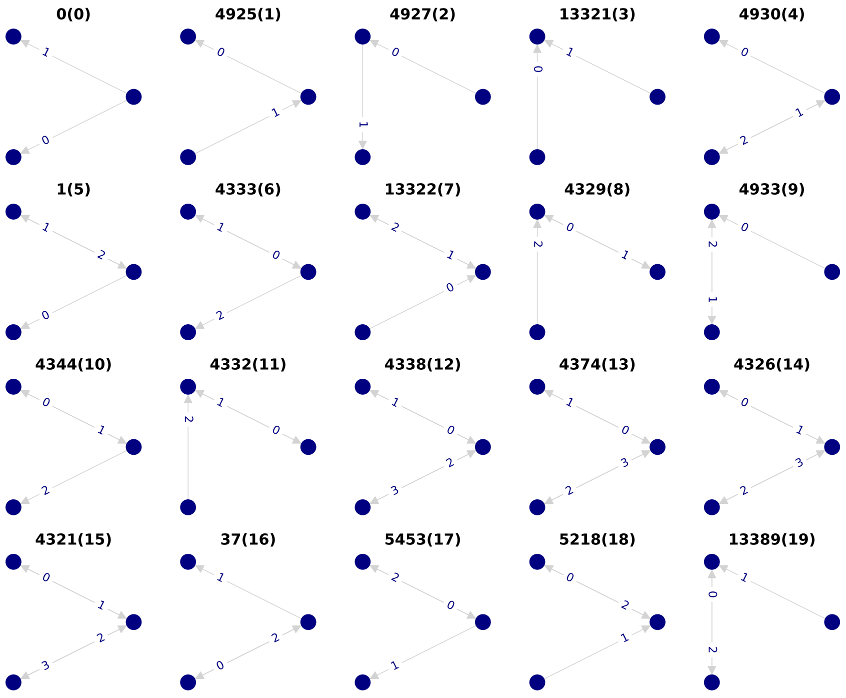


Figure 11.5: *Representation of the 20 most frequent pre-conditions*

an unprecedented perspective in the field of network evolution. It provides a profile of network evolution starting from each initial subgraph (precondition), as defined in Chapter 11.2. The datasets in Table 11.2 are ordered by the number of edges, revealing interesting insights. Notably, while Steemit is not the largest graph, it presents more than double the number of rules compared to Cryptokitties. This non-linear correlation between size and number of patterns is also evident in the first two datasets: despite comparable sizes of the Bitcoin and UC-social graphs, Bitcoin exhibits almost twice the number of rules. Figure 11.6 illustrates the relative size of these intersections, providing stronger confirmation that the Cryptokitties and Steemit graphs differ significantly from the other three, as evidenced by the distinct blocks in the figure. The precondition that most clearly demonstrates this two-cluster division is 4927 (shown in the first row of Figure 11.5), representing a path. For this precondition, Steemit and Cryptokitties present a high percentage of common

Dataset	pre	n. rules	median post
UC-social	287	2,885,731	5
Bitcoin Alpha	323	4,827,543	6
Sarafu	325	27,108,973	7
Steemit	325	484,574,431	6
Cryptokitties	325	231,124,737	1

Table 11.2: *Overview of tulip results.* ***pre*** *column counts the number of precondition found in each dataset.* ***n. rules*** *and* ***median post*** *columns refers to the evolution part, respectively counting the number of rules found and the median number of post condition for each pre condition.*

post-conditions, while the set of common post-conditions in the other three datasets remains considerable. Interestingly, for precondition 0, Steemit and Cryptokitties present a lower percentage of intersection, despite this precondition being very frequent in both graphs (as seen in Figure 11.5). This suggests that although both networks frequently exhibit this initial state, they evolve in different ways. This divergence makes sense given the different nature of Steemit compared to Cryptokitties, despite both being part of the Web3 landscape. Cryptokitties primarily represents financial transactions, while Steemit has a distinct social media focus (follow operations).

We can now delve deeper into the actual evolutionary profile, visualizing it for each precondition similarly to how we examined the precondition profiles in Figure 11.4. This involves first observing the distribution and then selecting noteworthy postconditions. Figures 11.7 and 11.9 present results for preconditions 0 and 4927 (discussed earlier), focusing on the first 20 postconditions to appear across all datasets out of 302 total postconditions. We denote with the symbol $\nexists$ the absence of evolution, meaning that the starting subgraph do not evolve in any way respecting the consraints given by the parameters max_{ts}, Δ_t, and max_d. A key observation is that precondition 0, highly present in both CryptoKitties and Steemit datasets, exhibits different common postconditions (only 14% overlap). Figure 11.7 provides more details, with subfigure (a) including postcondition $\nexists$ (representing absence of evolution edges within max_d and Δ_t parameters) and subfigure (b) excluding it for better visualization of other frequent evolutions. The $0 \to 0$ rule (corresponding to $\nexists$ postcondition) frequency approaches 1 in Steemit, while in CryptoKitties there is space to observe other postconditions because postconditions 275 and 272 have minor but noticeable frequency shares (shown in Figures 11.8a and b). The other three datasets exhibit distinct profiles: Bitcoin predominantly features postcondition 136 (Figure 11.8c), UC-social focuses on 275, and Sarafu shows a

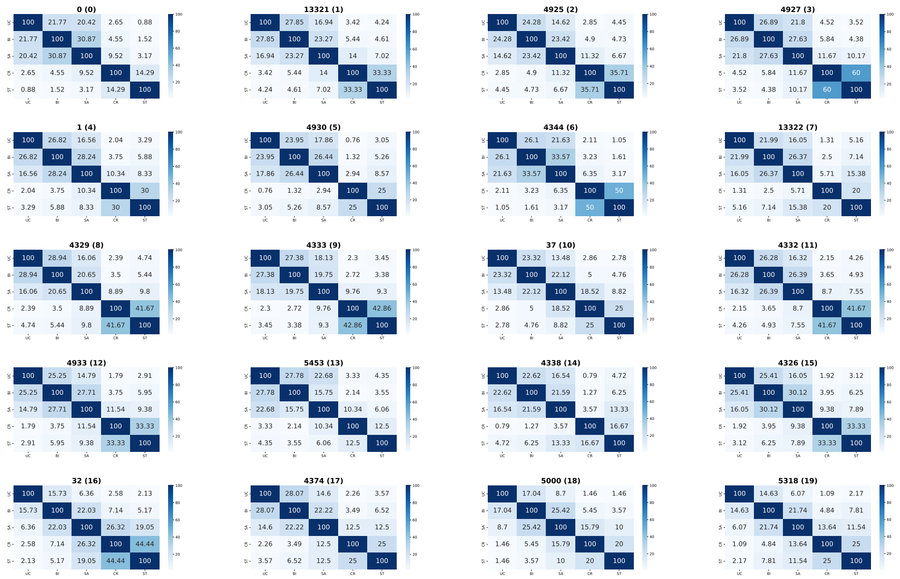
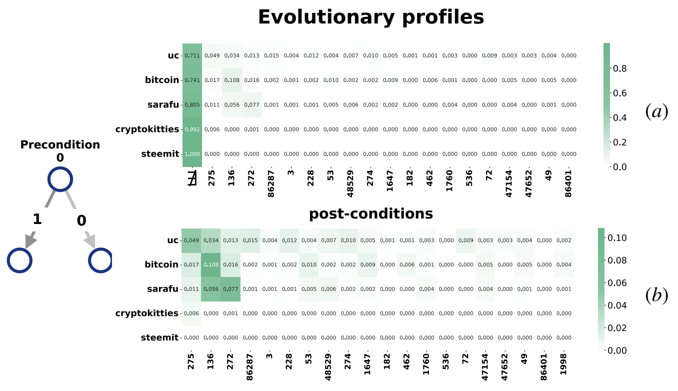


Figure 11.6: *Relative size of the intersections of post condition, for each of the most frequent pre-conditions.*

mix of both 136 and 272. Interestingly on the Web3 platforms, postcondition 136, representing reciprocal behavior, is absent in Steemit and CryptoKitties networks due to their lack of classical social behavior. Conversely, it features prominently in Sarafu, reflecting its strong humanitarian/collaborative component. Postcondition 136 exhibits its most significant presence in the Bitcoin network, this prevalence of reciprocal behavior aligns well with Bitcoin's underlying trust-based dynamics.

Figure 11.9 details the evolutions of precondition 4927, with subfigure (a) including and (b) excluding the absence of evolutions (postcondition $\nexists$). These profiles resemble those in Figure 11.7, despite the different intersection between postconditions shown in Figure 11.6. This similarity stems from the skewed distribution of profiles across few rules. The first three postconditions for precondition 4927 correspond to those of precondition 0, representing the same evolution but starting from a different subgraph. For instance postconditions 136 and 5822 (in Figure 11.8c and f, respectively) represent the reciprocal evolution of the most recently created edge ($ts = 1$). This distribution is reflected in the frequent occurrence of the reciprocal evolution (id=5822) in the

Bitcoin graph (a trust network), while being entirely absent in Steemit and CryptoKitties networks.



	∄	275	136	272	86287	3	228	53	48529	274	1647	182	462	1760	536	72	47154	47652	49	86401
uc	0.711	0.049	0.034	0.013	0.015	0.004	0.012	0.004	0.007	0.010	0.005	0.001	0.001	0.003	0.000	0.009	0.003	0.003	0.004	0.000
bitcoin	0.741	0.017	0.108	0.016	0.002	0.001	0.002	0.010	0.002	0.002	0.009	0.000	0.006	0.001	0.000	0.000	0.005	0.000	0.005	0.000
sarafu	0.805	0.011	0.056	0.077	0.001	0.001	0.001	0.005	0.006	0.002	0.002	0.000	0.000	0.004	0.000	0.000	0.004	0.000	0.001	0.000
cryptokitties	0.992	0.006	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
steemit	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

	275	136	272	86287	3	228	53	48529	274	1647	182	462	1760	536	72	47154	47652	49	86401	1998
uc	0.049	0.034	0.013	0.015	0.004	0.012	0.004	0.007	0.010	0.005	0.001	0.001	0.003	0.000	0.009	0.003	0.003	0.004	0.000	0.002
bitcoin	0.017	0.108	0.016	0.002	0.001	0.002	0.010	0.002	0.002	0.009	0.000	0.006	0.001	0.000	0.000	0.005	0.000	0.005	0.000	0.004
sarafu	0.011	0.056	0.077	0.001	0.001	0.001	0.005	0.006	0.002	0.002	0.000	0.000	0.004	0.000	0.000	0.004	0.000	0.001	0.000	0.001
cryptokitties	0.006	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
steemit	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Figure 11.7: *Profile of post-condition for pre-condition id* $= 0$*, both including in (a) or excluding in (b) the* $\nexists$ *postcondition.*

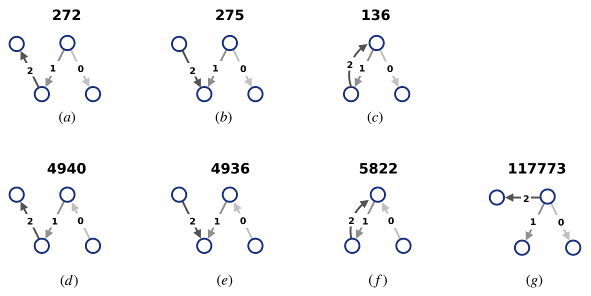


Figure 11.8: *Representation of the post-conditions of the pre-condition id* $= 0$ *(from (a) to (c)) and of the pre-condition id* $= 4927$ *(from (d) to (g)) cited in the discussion.*

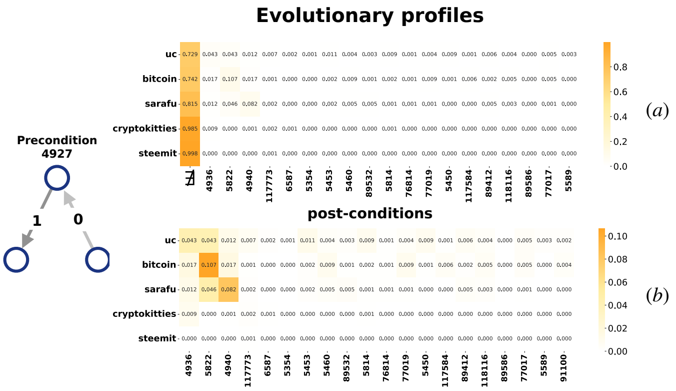


Figure 11.9: Profile of post-condition for pre-condition id = 4927*, both including in (a) or excluding in (b) the ∄ postcondition.*

11.3.3 Sensitivity to parameters

To assess the robustness of our algorithm, we conducted a sensitivity analysis focusing on two key parameters: max_d and Δ_T. This analysis was performed on the smallest graph in our dataset, UC social, to ensure computational feasibility while still providing meaningful insights. The max_d parameter controls the influence of hubs by limiting the number of outgoing and incoming edges for each node in the subgraph. We examined the algorithm's behavior across a range of max_d values: 5, 10, 20, and 42. The value 10 represents the average degree and was used as the default in our previous results. The upper bound of 42 corresponds to the 99th percentile of the degree distribution, capturing the behavior of highly connected nodes. The Δ_T parameter defines the time window for evolution events, effectively controlling the counting of evolutions that are temporally distant and thus potentially less causally related. In our results we based the choice on the inter-time distribution of events, choosing the 50^{th} percentile of this distribution as default. To test sensitivity, we also examined the effects of setting Δ_T to the 25^{th} and 75^{th} percentiles of this distribution. To isolate the effects of each parameter, we employed a one-at-a-time approach. For the Δ_T analysis, we fixed max_d at its default value of 10 and varied Δ_T across the 25^{th}, 50^{th} (default), and 75^{th} percentiles of the inter-time distribution. For the max_d analysis, we fixed Δ_T at the 50^{th} percentile and varied max_d across the values 5, 10, 20, and 42. In our analysis of parameter sensitivity, we first examined the computational time required

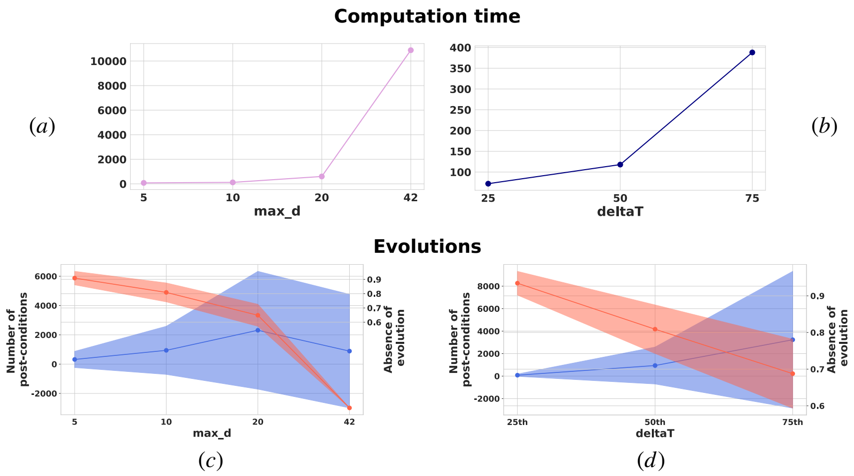


Figure 11.10: Sensitivity to parameters. *The computation time is evaluated when varying parameters max_d, in (a), and Δ_t, in (b). As for the evolution part,the plots show the mean number of post conditions (blue line with area indicating the standard deviation interval) and the percentage of frequency of the $\nexists$ post-condition (red line) when varying max_d, in (c), and Δ_t, in (d).*

when varying max_d and Δ_T. As illustrated in Figure 11.10a-b, the processing time remains relatively low and stable across most parameter configurations. However, there is a notable exception when max_d reaches 42, which results in a significant increase in computational time compared to other values. This spike in processing time is likely due to the substantially larger number of connections being considered in the subgraph, leading to more complex calculations, especially for what concerns the canonical form. Turning our attention to the evolution results, we focused on two key metrics: the percentage of $\nexists$ postconditions (indicating the absence of evolution meeting the given parameters) and the number of post-conditions. We observed that the percentage of $\nexists$ postconditions decreases gradually as the parameters increase. Interestingly, changes in Δ_T appear to have less impact on this decrease compared to changes in max_d. The most dramatic effect is seen when max_d reaches 42, at which point the mean percentage of $\nexists$ post-conditions drops to 0. This suggests that at this high max_d value, the algorithm captures all potential evolutions within the network, leaving no subgraph without an identified evolutionary path. Regarding the mean number of post-conditions, we noted an increasing trend as both parameters increase. This indicates that allowing for more connections (higher max_d) and longer time windows (higher Δ_T) enables the algorithm to identify more complex and varied evolutionary patterns. However, it's important to note that this increase in post-conditions is accompanied by a high degree of variance, as evidenced by the wide colored area surrounding the mean line in our visualizations. This variance suggests that while the algorithm generally identifies more post-conditions with higher parameter values, there is considerable variation in the complexity of evolutions detected across different parts of the network.

11.4 Conclusion

In conclusion, this chapter has presented TULIP, a novel algorithm that significantly advances the field of temporal network analysis. TULIP's key innovation lies in its ability to capture all possible evolution patterns of subgraphs within a temporal network, along with their corresponding probabilities, providing a comprehensive view of network dynamics over time. Furthermore, TULIP offers an evolutionary profile of the network, allowing researchers to track and analyze how the network's structure evolve, while also comparing the evolutionary behavior of different networks. We demonstrated TULIP's effectiveness and versatility by applying it to five diverse datasets: UC-Social (social network), Bitcoin Alpha (trust network), Sarafu (complementary currency financial network), CryptoKitties (NFT sales network), and Steemit Follow (Web3 platform follow operations). This diverse selection allowed us to explore TULIP's performance across various network types, including both Web2 and Web3 scenarios. Our analysis of these datasets yielded valuable insights from both pre-condition profiles and evolutionary patterns. The pre-condition profiles revealed similarities and unique feature for each network, highlighting the structural properties that serve as starting points for temporal evolution. The evolution analysis then uncovered fascinating commonalities and differences in post-conditions across the datasets, providing a nuanced understanding of how different network types evolve over time. Notably, we observed unexpected similarities in evolutionary behaviors between seemingly disparate networks, such as Steemit and CryptoKitties, both Web3-based but representing different interaction types. Conversely, we found that Sarafu, despite being a Web3 financial network, exhibited behavior more closely aligned with traditional social networks like UCSocial and Bitcoin Alpha. These findings underscore TULIP's potential to uncover non-obvious patterns and relationships in network evolution across diverse domains. Furthermore, our sensitivity analysis shows how TULIP's results change with respect to different parameters. By examining how computational time and other results changed with variations in key parameters, we provided insights into the algorithm's scalability and applicability to different network sizes and complexities. In summary, TULIP represents a significant step forward in our ability to analyze and understand temporal networks. Its comprehensive approach to capturing evolution patterns, combined with its demonstrated effectiveness across diverse datasets, opens new avenues for research in complex systems analysis.

Part IV

User Migration

User migration represents an evolutionary aspect of social networks that cannot be fully captured by graph evolution rules alone. This phenomenon is particularly relevant in the Web3 context, especially on blockchain-based online social networks (BOSNs), and presents unique challenges for network analysis.

Hard fork in Web3 online social networks. Web3 mainly relies on blockchain technologies to support a wide ecosystem of services. In the case of Web3 online social networks - Web3 OSNs - the underlying blockchain provides data storage and data validation for all the social operations. The validation process enables the production and exchange of cryptocurrencies, used in financial operations in the Web3 platform [119]. Among the proposed Web3 OSNs, Steemit [161], and the underlying blockchain Steem, has been the pioneer of the paradigm. Like other Web3 OSNs, it relies on a cryptocurrency, called STEEM, that can be exchanged for goods or services. Moreover, the cryptocurrency fuels a reward mechanism, which supports network growth by repaying users for their activity on the platform (more details about the Steemit platform can be found in Chapter 6). Web3 social platforms may offer data about a phenomenon that is peculiar to blockchain-based systems: a hard fork, i.e. an event that occur when miners do not consider as valid the blocks validated with a newly proposed consensus protocol so that two different branches are created if validators do not reach a consensus on the protocol to use. In this situation, the members of the original branch may decide to migrate to the platform based on the new branch, leading to a user migration. Such a split has happened on Steemit as well. After a dispute inside the network, a group of users on the 20th of March 2020 copied the blockchain data into a new blockchain called Hive. In this case, users are provided with the same username on both platforms, which means that they can still be active on both platforms.

User migration in OSNs. User migration is a “universal” process spanning online social media and networks but is not fully understood yet, especially in the Web3 world. Most of the studies are based on the most spread social platforms. For instance, by matching user accounts through external data, Kumar *et al.* [162] have analyzed user migration patterns. Newell *et al.* have conducted an analysis of user activity during a cross-platform migration with the goal of understanding the motivations behind migration. Other works have focused on users migrating across groups on the same platform, showing non-random migration patterns in Facebook groups [163]. A more in-depth analysis has been conducted by Davies *et al.* [164] in the case of user migration across COVID-19 subreddits. All previous studies are based on data collected from centralized social platforms and none of them has looked at user migration across Web3 platforms, especially as a consequence of a hard fork. Only recently Ba *et al.* [121] have focused on user migration in Web3 social platforms by evaluating the effects of user migration on the graph structure of the interactions and assessing the predictability of migrating. Finally, Ba *et. al.* [165] studied the Steemit user migration from a mesoscopic point of view, observing how communities are characterized by different migration behaviors. With respect to these latter works, here we focus on the behavior and role of hub nodes in leading the user migration process, and on the evaluation of the influence hubs may exert on their closest connections.

Evolutionary aspects of user migration In this thesis, we explore the unique challenges and implications of user migrations within the Steemit paltform, complementing the graph-based analyses of the previous chapters. Our first study examines two key aspects: the likelihood of hubs migrating to a new social platform after a shock event, and their influence on their neighbors' migration decisions. We find that different hub types employ varied migration strategies. For example, financial hubs tend to maintain a presence on both the original and new platforms. We also observe that users directly interacting with hubs are more likely to migrate. These findings highlight the importance of understanding hub activity and influence in managing user migration processes.

We then investigate the complex relationships between groups, discussions, and migration patterns. Using network-based analysis focused on community detection in multilayer networks, combined with text mining, we uncover several key insights:

- A group's network position correlates with its migration likelihood, with marginal groups more prone to leaving;
- Group network structure is significant: users in densely connected groups with strong financial interactions are more likely to stay;
- Users who migrate discuss different topics compared to those who remain;
- User groups involved in monetary transactions show increased interest in migration-related content if considering leaving

These results emphasize the crucial role of social and economic relationships during user migration triggered by fork events. In the broader context of online social media, this motivates a network-inspired approach to studying user migration, focusing on groups and specific subgraphs while leveraging user-generated content.

Finally, we introduce a novel machine learning pipeline using graph neural networks (GNNs) to predict user migration in BOSNs. We model the data as a directed temporal multilayer graph, capturing both social and monetary user interactions. To address class imbalance in node classification, we propose a data-level balancing technique based on undersampling. Our evaluation shows that GNNs are suitable for user migration prediction, and our undersampling approach significantly improves predictive power on severely imbalanced data.

Chapter 12

Influence of hubs

The current online social network landscape is characterized by competition to get larger audiences leading to massive user migrations which will determine the shape of the future Web. However, user migration phenomena have not been fully understood and their driving mechanisms are still not well identified; in particular, the behaviors of hubs and the influence they exert on their followers are unclear. In this work, we focus on these aspects by analyzing the propensity of hubs to migrate towards a new social platform as a consequence of a shocking event; and the influence they exert on the decision of their neighbors of migrating to a new platform or staying on the native one. We conducted analysis on the data about Steemit and Hive described in Chapter 6, due to the blockchain nature of these Web3 platforms, we got detailed data about social and financial interactions among the users, along with information that allowed a precise reconstruction of the context surrounding the migration. The main findings suggest that different types of hubs apply different strategies when choosing to migrate, e.g. financial hubs diversify their strategy by staying and migrating at the same time. As for hub influence, results suggest that users directly interacting with hubs tend to migrate. In general, findings on influence indicate that understanding the activity and the influence of hubs is crucial in monitoring and controlling the user migration process. In particular, our research questions are:

1. How do the hubs behave when faced with the choice to migrate to a new platform? Are they more likely to migrate or stay, so keeping their role and status?
2. Since in modern Web3 OSNs, accounts may gain importance or popularity through different types of interactions and strategies, even involving

financial transactions, do different definitions/types of hubs lead to different behaviors when faced with the choice of migrating or not?
3. Does the decision made by hubs whether to migrate or not influence their neighboring nodes? Do they exert of sort of social/financial pressure on their neighborhood?

12.1 Dataset

In this work, we consider all the users' financial and social operations of the Steemit platform (described in Chapter 6), from June 3, 2016, up to January 21, 2021. Specifically, for the Steem blockchain, the obtained data collection consists of $993,641,075$ social operations and $72,370,926$ financial operations; while Hive registers a total number of $206,224,132$ social operations and $4,041,060$ financial actions. The cited number of operations concerns more than 1.4 million users on the social layer and around 1.3 million on the financial layer.

12.2 Methodology

Despite the variety of operation types, the operation schema is unique, so we can model each transaction collected by APIs as a tuple $I = (u, v, t, r)$, which describes an interaction between users u and v of type r at time t. As mentioned before, we leveraged the transactions' classification introduced in [110] and focus only on operations between users such that $r \in \{social, financial\}$.

12.2.1 Graph modeling

Based on the set of tuple interactions Is, we built different network-based representations of the interactions among users, according to the time period we dealt with. Specifically, we adopt an incremental graph-based representation for data up to the hard fork moment, denoted by T_{fork}, and a snapshot-based graph representation for interactions that happened after the hard fork:

- **Growing projection graph:** Concerning operations that happened before T_{fork}, we build two different graphs - layers - that isolate the different types of operations. So, we obtain two directed weighted graphs G_s and G_f, that include edges with r equal to social s or financial f, respectively. Both G_s and G_f adopt the growing projection described in Chapter 3's taxonomy, building a graph $G_[1, T_{fork}]^{GP}(r) = (V, E, f)$ in which the edge (u, v, w) indicates that nodes u and v had $f(u, v) = w$ interactions of type r. In this modeling, the edge-labeling function indicates the number of interactions among the specific couple of nodes instead of the timestamp.

- **Sequence of graphs** The transactions occurring after the hard fork can happen on two different layers (social or financial), but they also involve one of the two blockchains (Hive or Steem). For these reasons, after the hard fork, we adopt the interval model (defined in the taxonomy described in Chapter 3) to get four different graphs that isolate the different kinds of operations and blockchains. Formally, we deal with four directed weighted temporal graph sequences $G^H_s[1\dots9], G^H_f[1\dots9]$ and $G^S_s[1\dots9], G^S_f[1\dots9]$, where s and f stand for social or financial, respectively; while H and S indicate on which blockchain the operations have been recorded, so Hive or Steem, respectively. Each graph G^P_r, representing transactions of type r happened on platform P, is defined as the sequence $< G^P_{r1}, .., G^P_{r9} >$, where each $G^P_{ri}, i = 1, \dots 9$ represents a 1-month window aligned to the day of the hard fork. Specifically, each graph (snapshot) G^P_{ri} covers transactions with a timestamp from the 21st of the $i-1$-th month to the 20th of the i-th month, starting from March 2020. This interval model allows us to observe the activity of nodes in each period, and consequently to study the migration choices with a monthly granularity.

12.2.2 Hub definition

We decided to investigate the role of two very different kinds of hubs based on *(a)* the degree and *(b)* their involvement in platform management. Concerning the degree, we select two sets of hubs: *(i)* **social in-degree hubs** are the 21 nodes with the highest in-degree on the social layer, and *(ii)* **financial degree hubs** are the 21 nodes with the highest degree on the financial layer. We choose to use the undirected version of degree in the financial layer in order to get the nodes that economically interact the most with other nodes, considering the in-degree or out-degree too restrictive in this case. The set of financial and social degree hubs only includes nodes that performed at least an operation in the last month before the fork, thus being active. This filter is needed to be sure to select hubs that could actually influence other nodes with their action at the hard fork time.

The other kind of hubs concerns a type of user that plays an essential role on Steem, called witness [166]. Basically, they are the set of people that can actually create and validate blocks on the blockchain, and they are voted by users of the platform according to a consensus mechanism called Delegated

Proof of Stake (DPoS). The witness role is assigned at every election round to the 21 most-voted users. In this work, we select as central witness nodes, the 21 accounts that performed the highest number of *feed publish* operations. This operation can only be performed by the top 21 witness nodes at each round. So, the ranking based on the number of feed publish operations performed is a good estimator for selecting the nodes that played more times the witness role. Based on this ranking, we obtain two sets of witness hubs: *(i)* **social witness hubs** are the top 21 nodes active in the last month before the fork on the social layer; *(ii)* **financial witness hubs** are the top 21 nodes active in the last month before the fork on the financial layer. Note that we filter on the active nodes in the last month before the fork and not those active in a longer period (3 months or one year before) because rumors about the hard fork began only one month before it happened. Therefore, we are only interested in hubs that are active in the actual period when they could have influenced their neighborhoods. After obtaining the four sets of hubs, we were able to study their role in the user migration process. The methodology is divided into two parts: in the first one we study the level of activity of hubs on the two platforms and their final decisions; then, we observe how the 1-hop neighbors of each hub behave with respect to the rest of the network.

12.2.3 Hubs activity

We first cope with the hubs and the dynamic of their activity. Specifically, we leverage the snapshot representation (interval model) to collect the number of operations on each platform for each hub. Formally, for each hub h, we compute the activity level separately on both platforms $P = \{S, H\}$, for each month i. The activity level of each hub is defined as follows:

$$p_i(h, P, r) = c_i(h, P, r)/c_i(h, r)$$

where:

$$c_i(h, P, r) = \sum_{\{(u,v) \in E_{G^P_{ri}} | u=h\}} w(u, v)$$

and:

$$c_i(h, r) = \sum_{\{(u,v) \in E_{G^S_{ri}} \bigcup E_{G^H_{ri}} | u=h\}} w(u, v)$$

In short, the activity level p_i for a hub h indicates the fraction of social or financial operations done on a specific blockchain.

Activity visualization. In order to get a direct view of the preferred platform for each hub, we process its activity on both platforms, for each period and operation type. We define $max\_p_i(h,r)$ as follows:

$$max\_p_i(h,r) = \begin{cases} +p_i(h,H,r) & \text{if } p_i(h,H,r) > p_i(h,S,r) \\ -p_i(h,S,r) & \text{otherwise} \end{cases}$$

Through this indicator, we are able to summarise through a heatmap the preferred platform for each hub, monthly, and by type of operation. The more each cell of the heatmap is red, the more $max\_p_i(h,r)$ is close to 1, meaning that the hub performed almost every operation on Hive. On the contrary, the more the cell is blue (-1), the more the hub remained on Steem. Softer colors correspond to a more balanced activity level on both platforms.

Migration decision. After observing the preferred platform for each hub, we assign a migration decision to each hub. Note that $max\_p_i(h,r)$ admits values in the interval $[-1,-0.5]$ or $[0.5,1]$, so we define *inactive* the periods where $max\_p_i(h,r) = 0$. Moreover, the migration decision depends on the value of $max\_p_i(h,r)$ in the most recent active month, (noted as $max\_lastive(h,r)$) and it is defined as follows:

$$\textbf{decision} = \begin{cases} \textit{migrant} & \text{if } max\_lastive(h,r) \geq 0.75 \\ \textit{resident} & \text{if } max\_lastive(h,r) \leq -0.75 \\ \textit{inactive} & \text{if } max\_lastive(h,r) = 0 \\ \textit{diversifier} & \text{otherwise} \end{cases}$$

12.2.4 Hubs' influence

Our main goal is to discover whether and to what extent hubs influence migration choices. Specifically, we investigate whether direct neighbors of hubs tend to make different decisions with respect to the other nodes in the network. To this aim, first, we compute the migration decision of all nodes in the neighborhoods of hubs through the same criteria described in the previous section. Then, we compare the distribution of decisions of all the nodes in the graph against the nodes that belong to the neighborhood of at least one hub. Note that, we consider as neighbors of a hub all nodes with an outgoing edge towards the hub from the beginning of the data collection (June 3, 2016) up to the fork date (March 20, 2020). Moreover, when we mention all nodes, we actually mean all the nodes in the layer we are considering (social or financial) that were active in the last month before the fork, i.e. all nodes that could make a migration choice. Finally, for each hub h we define the tuple $(m(h), r(h), d(h), i(h))$ that reports the percentage of its neighbors, computed in the incremental graph $G^{layer}_{T_{fork}}$, that are classified as migrant (m), resident (r), diversifier (d) or inactive (i), respectively.

12.3 Results

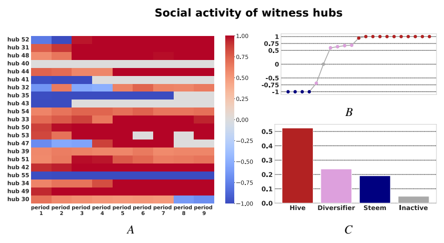
In this section, we introduce the main insights following the methodology described in Section 13.2. Indeed, we first analyze the activity of hubs, then we investigate the influence they exert on their neighborhood as for the decision of migrating or remaining.

12.3.1 Hubs activity and migration choice

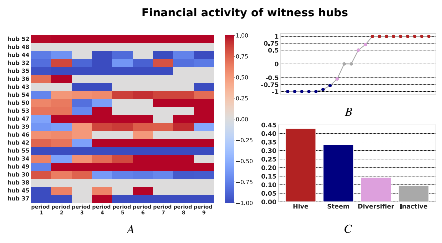
As previously mentioned, we identify hubs based on centrality criteria and the network layer. First, we focus on social and financial witness hubs. The two sets share the majority of nodes (16 common hubs over 21). However, their activity is significantly different in the two layers, so it is worth analyzing them separately. In the same way, we can distinguish two sets of degree hubs: the in-degree hubs in the social layer and the undirected-degree hubs in the financial layer. Due to the different definitions of central nodes, the sets of the two different layers share fewer nodes with respect to the witness case (12 over 21 hubs).

Social witness hubs. Figure 12.1a describes the social witness hubs' activity from three different points of view: Figure 12.1a-A shows the activity level $max\_p_i(h, r)$ for each hub h and period i for $r = social$; Figure 12.1a-B concerns the migration choice level of hubs ($max\_lastive(h, social)$); and Figure 12.1a-C reports the distribution of the migration decision. In this case, it highlights that hubs tend to migrate, but the decision is not immediate, as indicated by the heatmap. In fact, we can observe that the typical behavior of migrant hubs (7 over 11 migrants) is to be active on both platforms for some months and then prefer the new platform Hive. On the other hand, the resident hubs do not manifest a period of dual activeness. In general, both resident and migrant decisions are strong, meaning that one dedicates the entire activity to one platform only. In fact, as displayed in Figure 12.1a-B, most of the resident and migrant hubs are on the $+1$ or -1 lines, i.e. a full-time activity on a single platform.

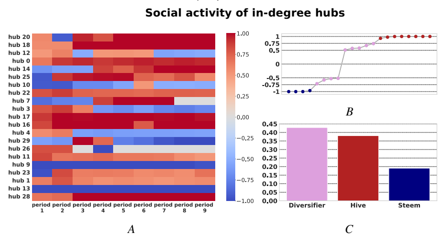
Financial witness hubs. Figure 12.1b reports the three viewpoints about witness hubs' financial activity, as in the previous case. It is clear that, even if



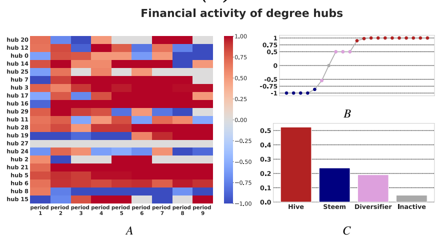
(a)



(b)



(c)



(d)

Figure 12.1: *From a) to d), properties of the activity level of social witness hubs (a), financial witness hubs (b), social in-degree hubs (c), and financial degree hubs (d). For each type of hub, we report the heatmap - A - displaying the monthly trend of* $max\_p_i(h,r)$ *for the 21 hubs; B) the hubs increasingly ordered by* $max\_lastive(h,r)$ *and colored according to their final decision; and C) the distribution of the decision (migrant, resident, diversifier and inactive) of the hubs.*

the set of financial witness hubs shares the 76% of users with the social witness one, the activity is really different. The more evident difference concerns the increase of inactivity (grey cells). Another difference concerns the lapse of time between the hard fork and the hubs' decision: there is a group of hubs that decide in the very first period (2 months); on the other hand another group is more hesitant in making a strong decision but tends to have a preferred platform instead of staying active on both ones. In fact, two hubs only stay active on both platforms at the end. Concerning the strength of the migrant or resident decision, in the financial case, we observe that when the decision is *migrant* the totality of operations are done on the new platform Hive, while a few *resident* hubs still perform some operations on Hive, resulting into a not completely symmetric scenario.

Social in-degree hubs. Figure 12.1c reports the study on the activity levels and migration choices of in-degree hubs in the social layer. The main feature that distinguishes this set of hubs from the other ones is the absence of inactivity. Note that only hubs with no activity for the entire post-fork period are considered as *inactive*, otherwise we consider the most recent activity for assigning a label/choice. Looking at the distribution of decisions in Figure 12.1c-C, it is clear that the prevalent decision is to stay active on both platforms. The indecision is still in the behavior of some hubs that decided to migrate since the migration choice is often preceded by an indecision period. Finally, as in the previous cases, once the hubs make a decision, they are fully committed to the preferred platform, as shown in Figure 12.1c-B.

Financial degree hubs. Figure 12.1d depicts the activity data about the financial degree hubs. It confirms the lower tendency for degree hubs to become inactive after the fork - only 1 over 21. In general, the financial degree hubs seem to be undecided about their choice because, even if in the last active period they perform operation only in one platform, it took them some months to decide. For instance, “hub 2” started by exploring Hive more, after two months he moved back to Steem, then it returned to Hive, and so on. An interesting feature of the final decisions can be observed in Figure 12.1d-B: when the decision is *diversifier*, it corresponds to an activity that is almost perfectly balanced on the two platforms. The *diversifier* decision that is more imbalanced is related to a percentage of 0.54 in favor of Steem. In general, financial degree hubs have played an expected diversification strategy where initially they started to explore the economical value of the new platform, then they diversified their actions between the two blockchains.

12.3.2 Influence of hubs

A further important element that may drive a user migration process is the influence hubs may exert on their direct neighborhood. Here we report the findings on this aspect from two viewpoints: *(a)* a comparison among the distribution of hubs’ choices, that of all nodes in the graph and that of nodes that are neighbors of at least one hub; and *(b)* a comparison between the distribution of choices within each hub neighborhood and the distribution of nodes’ decisions in the graph. Specifically, we only report the trends that are worthy of analyzing from a single-hub perspective.

Social in-degree hubs influence. Figure 12.2a reports the outcomes of our analysis on the influence of social in-degree hubs. The first row (from A to C in Figure 12.2a) visualizes the distributions of the decisions grouped by the three different cohorts detailed above. Here, we can observe that the choices of hubs follow a different distribution with respect to all the nodes active at T_{fork}: while the majority of hubs stay active on both platforms or migrate, in the entire network it is more common to be *resident*. Further, the key element to discovering whether hubs influence their neighbors is to compare the distributions of every node decision w.r.t neighbors' decisions (B and C in Figure 12.2a). Even if the ranking of labels is the same, the percentage of nodes in each class is different: in fact, in the neighborhood's distribution the *migrant* label gains 7.3%. This difference is confirmed by the distribution of $m(h)$ for the 21 social in-degree hubs shown in Figure 12.2a-D: for every hub, their neighborhood is characterized by a higher percentage of migrants with respect to the expected fraction of migrants. Moreover, the neighbors of hubs tend to be less inactive (shown in Figure 12.2a-E), maybe as a consequence of the absence of inactive in-degree social hubs. So, social in-degree hubs are never completely inactive and tend to prefer Hive, either in an exclusive way or in addiction to Steem. This tendency is reflected in their neighbors, where the percentage of users moving to Hive always increases together with a decrease in the inactive decision.

Social witness hubs influence. Figure 12.2b is structured in the same way as for the social in-degree hubs. From a global point of view, the distribution of hubs' migration choices differs completely from the one concerning all active nodes that could be influenced: the trend is the opposite because 52.4% of hubs are migrants, while 54.8% of active nodes are residents. In this case, the gain of the migrant decision in the hubs neighborhoods is 11.7%. In Figure 12.2b-D it is more evident because in some hub's neighborhoods the migrant fraction (red) is even higher than the resident one (blue). Moreover, the percentage of inactive is always lower than the one relative to *diversifier* decision, as shown in Figure 12.2b-E. So, the key feature of social witness hubs is their strong preference for migrating to Hive. This is reflected in the neighbor nodes, and it is particularly evident when looking at the neighborhood of every hub separately.

Financial witness hubs influence. Figure 12.2c reports findings on the financial witness hubs' influence. The first observation about this result concerns the distribution of migration choices of active nodes: in contrast with the previous ones, the distribution of labels here is almost homogeneous. As for the distribution of labels among hubs' neighborhoods (see Figure 12.2c-C), it is more similar to the distribution of hubs' choice (see Figure 12.2c-A) w.r.t. the active nodes one (see Figure 12.2c-B). As detailed in Figure 12.2c-D, hubs neighbors tend to migrate more and become inactive less, since the labels with the highest variations are migrant and inactive. In short, financial witness hubs play an influential role on their neighbors because the decision

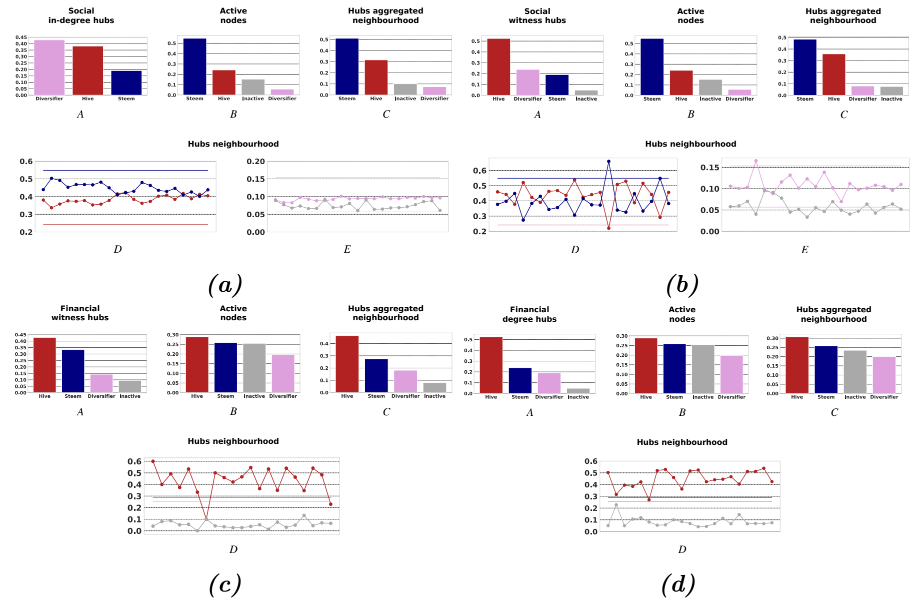


Figure 12.2: *From a) to d), properties of the influence exerted by social in-degree hubs (a), social witness hubs (b), financial witness hubs (c), and financial degree hubs (d). For each type of hub, we report A) the distribution of the decisions (migrant, resident, diversifier and inactive) of the hubs, B) the distribution of the decisions of all the active users, and C) the distribution of the decisions of the hubs' neighbors. Plots displayed in D) and E) report $m(h)$, $r(h)$, $d(h)$, and $i(h)$ values for the hubs. In these plots, horizontal lines represent the "expected" decision taking the distribution in B as "average" behavior.*

distribution differs a lot from the expected one. The difference is mainly driven by the strong increase in the decision of migrating towards Hive, in contrast with the decision of being inactive.

Financial degree hubs influence. Finally, Figure 12.2d shows how financial degree hubs influence their neighbors. In contrast with the previously described case of financial witness hubs, here the distribution of migration choice of hubs neighbors, shown in Figure 12.2d-C, is more similar to the general one, reported in Figure 12.2d-B. However, when observing the single $m(h)$ values plotted in Figure 12.2d-D, we can see an actual difference, similar to the one shown in the financial degree hubs case: the migrant choice fraction always (except 1) exceeds the percentage of expected migrants provided by the overall fraction of active nodes, while the percentage of inactive nodes always decreases. So, financial degree hubs present a dominant tendency to choose to migrate to Hive. Concerning the hubs' neighbors, despite the similar homogeneous distribution, the gain in the percentage of migrant decisions is evident.

After observing the influence of every type of hub, we can now highlight the main characteristics regarding the influence hubs have exerted on their neighbors when it came to deciding to migrate or remain. First, it is clear that hubs' neighborhood tends to migrate more frequently than "average" active users. Moreover, being a neighbor of a hub correlates with a lower probability of being inactive after the fork, i.e. neighbors of hubs are more likely to keep their activities in one of the two blockchains. In general, the influence that hubs exert on their neighborhood does not reflect in a complete change in the ranking of most frequent decisions. In fact, the most frequent decision in the overall graph is the same as the one of the hubs' neighbors, but the distribution change. Moreover, on the financial layer, in each hub's neighborhood, the fraction of migrant nodes particularly increases in contrast to the fraction of nodes that become inactive. This suggests that being close to financial hubs makes a node more motivated to be active even after a strong event like the hard fork. The same observation holds for the social layer, where there is also a tendency for hub neighbors to be active on both platforms - diversifiers - with a higher probability with respect to the "average" decision.

12.4 Conclusions

In conclusion, this work aims to observe the decisions of central nodes and the influence on their neighbors, in the context of a blockchain-based social network's split event. We focused on the fork event involving Steemit, leading to the birth of a new social network, Hive. Since the latter has maintained the same usernames as Steemit, we were able to track the user migration. We modeled the transactions [167] before the hard fork using an incremental weighted graph. On the other hand, we adopt a snapshot approach to model operations after the fork on both platforms, building a sequence of edge-labeled multigraphs, divided into two layers: social and financial ones. On this data source, we observe the variety of decisions of four types of hubs defined by degree and involvement in management operations, on both social and financial layers, highlighting that the most common decision for hubs is to migrate. Then, we focus on the decisions of hubs' neighbors, studying if they are influenced by the choice of their hub. Results suggest that when a node is a direct neighbor of a hub, it tends to migrate and not be inactive. Moreover, the influence behavior is more similar when observing hubs on the same layer instead of the same type of hubs. Future works in this context may concern the centrality transferability, i.e. the analysis of how the centrality of nodes is correlated across different layers. Another direction could be related to the influence of central nodes within a mesoscopic level.

Chapter 13

Influence of groups discussion

User migration, the large-scale movement of users between online social platforms, is a significant phenomenon in modern social networks, including blockchain-based social networks (BOSNs). In BOSNs, migration often occurs through hard forks, where the original blockchain splits, creating an alternative chain that users may choose to join. However, our understanding of user migration mechanisms remains limited, particularly regarding the role of tightly-knit user communities during these events. This study investigates the differences between users who stay and those who leave in terms of network structure and discussion topics. Through network-based analysis focusing on community identification in multilayer networks and text mining, we found that:

A group's position within the network of social and economic interactions correlates with its likelihood to migrate, with marginal groups more prone to leaving. Group network structure plays a crucial role, as users in densely connected groups with monetary interactions are more likely to remain. Users who migrate tend to discuss different topics compared to those who stay. User groups engaged in monetary transactions show increased interest in migration-related content if they intend to leave.

These findings emphasize the importance of social and economic relationships during user migration caused by fork events. In the broader context of online social media, this study underscores the need to examine user migration through a network-based approach that focuses on groups and specific subgraphs while also considering user-generated content.

13.1 Research questions

Network structure has been shown to play a crucial role in user migration processes. However, the specific impact of tightly connected user groups, or communities, during migration and fork events remains unclear. From a network perspective, we aim to explore the relationship between group network structure and migration patterns. Additionally, a unique aspect of migration in social networks is that users can discuss and coordinate their decisions on the same platform they may eventually leave. These discussions between those intending to stay and those planning to leave could indicate future intentions or influence decision-making. We seek to analyze user-generated content to better understand how pre-fork discussions within communities relate to their subsequent migration choices. These considerations lead us to two main research questions:

Research question RQ1: How does the network structure differ between user groups who remain on the original platform and those who migrate to a new one?

Research question RQ2: Is there a relationship between the topics communities discuss before a fork event and their ultimate migration decision?

13.2 Methodology

In this section, we present our proposed methodology. We begin by describing how to model the dataset for the task at hand, how to extract the network structure, and how to construct user migration-related labels before identifying communities. Finally, we provide the approach we intend to use to answer our research questions.

13.2.1 Modeling BOSN and user migration

Graph-based modeling

In BOSNs we have many different actions supported, with the traditional "social" interactions coexisting with economical or financial operations tied to the transfer of cryptocurrency tokens. Furthermore, each activity is timestamped. Each action in this scenario can be represented as a tuple (u, v, t, r), where u and v are users who interact through an action of type r at time t. We can create a multi-layer network [168] using the sequence of all the users' actions. We denote this network as $G_{T_{fork}} = (V, E, R)$, where:

- V is the set of users u who have participated in at least one interaction action in the set $I = \{(u, v, t, r)\}$ which has occurred before or at the timestamp T_{fork};
- E is the set of triple (u, v, r) with $u, v \in V$ and $r \in R$, representing a specific type of action among the ones in the set R of actions supported by the blockchain.

The resulting multi-layer network encodes the structure of the interactions among users prior to T_{fork}, i.e. the hard fork date; where the layers correspond to different types of actions available to support user interaction. In particular, here, we separate social and economic/financial interactions into two separate groups, thus decreasing the number of layers in $G_{T_{fork}}$ to two, the "social" layer and the "monetary" layer.

Given this setting, we are able to model a fork event, and the subsequent cross-platform migration, where users might migrate to another platform. Specifically, given two platforms, S and H, and a fork event at time T_{fork}, we consider *a)* **Migrant**: a user who performs at least one action on the new platform H after time T_{fork}; *b)* **Resident**: user staying on the original platform S, without performing any actions on the new platform H after time T_{fork}; and *c)* **Inactive**: users who are inactive or abandoned both platforms.

Community detection

In our research, we delve into the critical role of groups during user migration processes. To uncover these group structures within complex networks, we employ community detection techniques. After evaluating various state-of-the-art methods suitable for multilayer network analysis [169], we selected the Infomap algorithm [170], which utilizes random walks to identify communities. This choice was primarily motivated by Infomap's exceptional scalability [169]. Infomap's approach begins by assigning unique codewords to each node using prefix-free coding, such as Huffman coding. This allows random walks on the network to be represented as concatenations of these codes. The algorithm's core premise is that a random walker, upon entering a densely connected region of the graph (indicative of a community), will likely remain there for an extended period. This occurs because nodes within a community have stronger connections to each other than to distant nodes. To optimize this behavior, Infomap assigns distinct codebooks to different regions, termed modules, effectively shortening codewords for nodes within the same region. Consequently, community detection becomes an optimization problem of finding the partition that minimizes the overall code length. Given our multilayer network model, we employ the multilayer version of Infomap [171]. This version maintains the main ideas of the single-layer version while introducing inter-layer edges. These edges connect representations of the same user across different layers, allowing the random walker to transition between layers during its journey. It's important to note that this approach allows a user to belong to multiple communities depending on the layer, while still considering information from all layers during community assignments. In our work, which incorporates two layers representing social and economic interactions, we define two types of communities: social communities, derived from the social layer, and monetary communities, emerging from the economic layer. This nuanced approach enables us to capture the multifaceted nature of user interactions and group formations within the complex landscape of user migration.

13.2.2 Community structure and user migration

To address our research question concerning the interplay between group network structure and user migration (RQ1), we must first define the role of groups in the migration process. Our approach begins with the modeling of a community graph $G^C = (V^C, E^C)$, an attributed network that encapsulates the relationships among communities. In this graph, nodes represent individual communities, while edges denote edges between users belonging to different communities. Specifically, we establish a link between communities c_1 and c_2 if there exists a connection between a user in c_1 and a user in c_2, with the edge weight corresponding to the total number of links between nodes in c_1 and nodes in c_2. This community graph modeling can be applied to both social and monetary communities, yielding a *social community graph* and a *monetary community graph* respectively. These community graphs serve as powerful analytical tools, enabling various types of analyses. For each community c_i, we can derive attributes such as the number of inactive members, residents, and migrants. These attributes, in conjunction with the network structure, provide a rich source for extracting insights. The process of constructing community graphs and their utility in our analysis is visually summarized in Figure 13.1. This representation offers a clear overview of how we transform individual user interactions into a higher-level community structure, setting the stage for our investigation into the dynamics of user migration within and between these community structures. Figure 13.1.

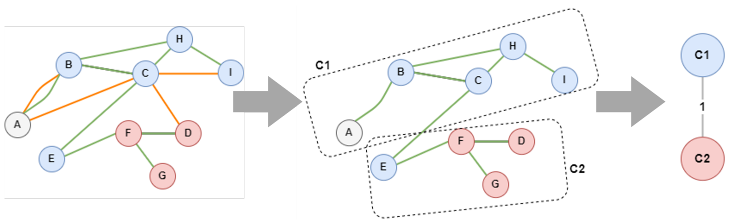


Figure 13.1: *An example of community graphs from a multilayer network. The initial network depicts nodes color-coded based on their migration status, interconnected via social (green) and monetary (orange) links. Following community detection, which identifies communities such as $C1$ and $C2$, we derive the corresponding community graph. In this graph, each node represents a distinct community, with inter-community edges reflecting the connections present in the original multilayer network. The color of each community node is determined by the predominant migration decision of its members: communities with a majority of residents trend towards sky blue, those with more migrants appear red, and those with a balanced composition are represented in white. The width of edges between community nodes is proportional to the strength of inter-community connections in the original network.*

Visualizing the interplay

To address RQ1, we begin by examining the community graphs, focusing on the inter-community connectivity as it relates to the migration status of community members. We then characterize the degree to which a community leans towards a particular user category (migrants or residents) by calculating the community entropy $H(c_i)$. This is defined as: $H(c_i) = \sum_{q=1}^{m} p_q(c_i) \log_2 p_q(c_i)$ where $p_q(c_i)$ represents the proportion of users in community c_i with label $q \in resident, migrant$. Furthermore, we analyze the subgraph induced by the nodes within a community c_i by computing its density $D(C_i)$, expressed as: $D(C_i) = \frac{|E|}{|V|*(|V|-1)}$ Here, $|E|$ denotes the number of edges in the subgraph, and $|V|$ the number of nodes. Subsequently, we investigate how both the density and entropy of communities correlate with the migration labels, providing insights into the relationship between community structure and migration patterns.

Measuring the interplay

To validate our visual observations, we perform an additional quantitative analysis. This involves examining several key metrics for each community: the count of inactive users, residents, and migrants, as well as the community's density and entropy. We assess the correlations between these community features, with a particular emphasis on how density and entropy relate to other characteristics. It's important to note that due to the nature of multilayer InfoMap, some communities may have an unusually small number of nodes or lack internal connections entirely. This can occur when nodes are grouped into the same community based on information from another layer, despite not having direct links within their current layer. Additionally, we encounter communities composed solely or predominantly of inactive nodes. For the purposes of our analysis, we exclude these atypical communities to ensure the robustness of our findings. This rigorous quantitative approach complements our visual examination, providing a more nuanced understanding of the relationship between community structure and migration patterns.

13.2.3 Community discussion and user migration

To address RQ2, we integrate user-generated content analysis with network structure examination. Our focus is on detecting and analyzing migration-related discussions in social networks, where users communicate through posts and comments. We concentrate on two key components derived from user posts: *hashtags* and *content topics*. Hashtags, words preceded by a hash mark (#), serve as user-defined content categorizations and facilitate content searches. By identifying migration-related hashtags, we can effectively pinpoint and analyze migration discussions. For a more direct analysis of text content and its relationship to migration, we propose an approach centered on content topics, which are extracted using topic modeling methods. Below, we outline our methodology for analyzing hashtags and text content topics:

Hashtags and communities

For hashtag analysis, we represent each user u with a user hashtag vector $\mathbf{v}_u = [v_1, v_2, \ldots, v_K]$, where $\mathbf{v}_u[k]$ denotes the count of the k-th hashtag. To compare multiple communities, we define the community hashtag vector of a community C as the average of its users' vectors: $\mathbf{v}C = \frac{1}{|C|}\sum u \in C\mathbf{v}u$. The community hashtag distribution $\mathbf{D}C$ is obtained by normalizing this vector: $\mathbf{D}C[k] = \frac{\mathbf{v}C[k]}{\sum_{i=1}^{|K|}\mathbf{v}_C[i]} \forall k \in 1 \ldots K$. In our analysis, we focus on a subset of hashtags, including both migration-related and unrelated ones. We then compare multiple communities using a heatmap visualization of these selected hashtags.

Topics and communities

For text analysis, we employ a topic-centered methodology. We first extract topics using Latent Dirichlet Allocation (LDA) [172]. Given a number of topics k, LDA groups articles into topics based on their content, allowing us to identify the most relevant topics. We apply this topic model to our entire collection of text documents (posts and comments) to visualize the discussion topics and their key words. Once an LDA model θ is trained on the document collection, it can provide a document topic distribution $\theta(d)$ for each document d. For each user u, we consider their set of posts $\mathcal{D}u$ and compute a user topic vector as the average of their document topic distributions: $\mathbf{v}_u = \frac{1}{|\mathcal{D}_u|} \sum d \in |\mathcal{D}_u| \theta(d)$. This vector represents the user's interest in each topic. To compare different user communities, we compute a community topic vector for a community C as $\mathbf{v}_C = \frac{1}{|C|} \sum u \in C \mathbf{v}_u$. We then visually compare these community topic vectors using a heatmap plot. This methodology allows us to analyze both hashtag usage and content topics across different communities, providing insights into how migration-related discussions vary among different user groups.

13.3 Dataset

In the following, we describe the data from the Steemit and Hive platform (described in Chapter 6) used for the analysis.

Interaction data

In this work, we focus on actions that represent an interaction between two users, either explicit or implicit. Specifically, we consider two main groups: *i)* financial and *ii)* social operations. Financial operations are those operations designated for the management of tokens, rewards, and asset transfer. In contrast, social operations are those that users are able to do on traditional social network platforms, like posting, rating, voting, sharing, and following. For the construction of the graph, we gathered operations from the very first block on the Steem blockchain, produced on 24th March 2016, up to the fork event, i.e. to block 41818752, with timestamp *2020-03-20T14:00:00*. While for migration status, we examine data after that timestamp, and up to January 2021. We recall that data between the two blockchains are identical up to the fork event, i.e. to block 41818752, with timestamp *2020-03-20T14:00:00*. From there, Hive and Steem have different data, as they have become two different blockchains. Overall, from the Steem blockchain, we extract $993,641,075$

operations describing social interactions and $72,370,926$ operations describing economic interactions; from the Hive blockchain, we get a total of $206,224,132$ social operations, and $4,041,060$ financial actions.

Text data

As we are interested in the discussion, we leverage the textual content produced by users before the fork. In Steem and Hive, users' posts and comments, are stored as *comment operations* on the underlying blockchains. The content of the post can be accessed as the *body*, and metadata information is also accessible including the *hashtags*. Please note that in Steemit, hashtags are called *tags* [1]. As a starting point, we consider the operations from [173], a total of $93,832,667$ *comment operations* that include both posts and comments. For this analysis, we are interested in fork-related discussion. Since everything started after the acquisition, we can focus on a limited period: for this work we focus on the period February 20, 2020 - March 20, 2020. Comment operations (both posts and comments) are in total $831,403$. We selected only i) posts not comments going down to $234,396$ posts, and ii) among them we consider only posts written in English, for a total of $140,638$ (the language is detected by the python library lang-detect). For the corpus of posts, pre-processing and cleaning are applied to the data to delete noisy, inconsistent, or incomplete items from the collection. Specifically, we applied the following operations: removal of HTML tags, URLs, punctuation, multiple whitespaces, numbers, stopwords, words shorter than 3 characters, and stemming. For this subset of documents, we find $284,932$ unique terms, while the average token length of posts is 104.5. Then, we consider the associated metadata information of the corpus of published posts, to derive the collection of hashtags. We perform some preprocessing on hashtag data as well: we filtered out the hashtag with less than 2 characters; then, we merged some hashtags of interest that share the same semantics. Specifically, we grouped all the hashtags that contain "hive" (e.g. hive-160196, hive-119845), "fork" (e.g. softfork, hardfork), "Justin", and "tron". We obtain $396,26$ unique hashtags, and on average, we observe 5.19 hashtags per post.

13.4 Results

We apply the methodology described in Section 13.2, on the Steem/Hive dataset, by first creating a multi-layer network with two layers: social and

[1] `https://steemit.com/faq.html#What_are_tags`

monetary, where nodes are labeled based on their activity after the fork. A summary of network statistics and labels is presented in Table 13.1.

Table 13.1: *Statistics for the multi-layer network $G_{T_{fork}}$, grouped by social and monetary layers.*

	Social layer	**Monetary layer**
Nodes	1352114	1247587
Edges	217926899	5056317
Inactive	1287321	1218535
Resident	43339	12757
Migrant	21454	16295

An analysis of our multilayer network reveals distinct characteristics between the social and monetary layers. The social layer exhibits a higher number of active users and connections, which aligns with the prevalence of social operations compared to financial transactions in online platforms. Interestingly, despite the monetary layer having fewer links, it involves a comparable number of users to the social layer. This suggests that while financial interactions may be less frequent, they engage a similar user base. A notable divergence emerges when examining the proportion of resident and migrant users across these layers. In the social layer, residents form the majority, with approximately two-thirds of active users remaining on the original platform and only one-third migrating to Hive. Conversely, the monetary layer displays an inverse trend, where the migration to Hive has had a more profound impact. Here, the majority of users have chosen to transfer their financial activities to the Hive blockchain. This disparity in migration patterns between social and financial interactions provides valuable insights into user behavior and preferences in the context of blockchain platform transitions. It suggests that users may have different motivations and considerations when deciding whether to migrate their social connections versus their financial transactions.

13.4.1 The interplay of community structure and migration

In this section, we answer RQ1 on the interplay between group network structure and user migration. We created the monetary community graph and social community graph using the methodology described in Section 13.2. The monetary layer's community graph has 76 communities and 252 inter-community edges, while the social layer's community graph has 105 communities and 205 inter-community edges. We visualize the obtained community graphs, with nodes - communities - colored according to the proportion of migrants and residents among their members in Figure 13.2.

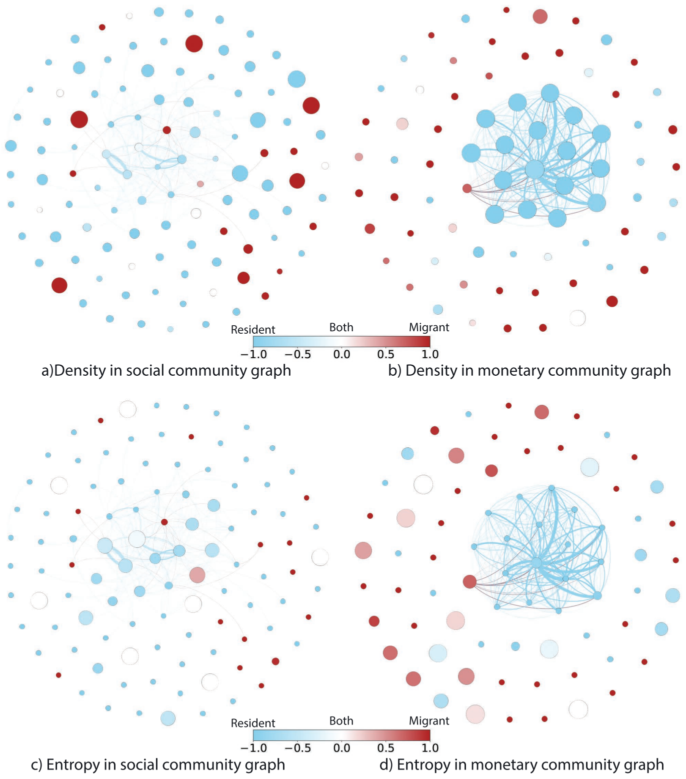

Resident
Both
Migrant
−1.0 −0.5 0.0 0.5 1.0
a)Density in social community graph
b) Density in monetary community graph
Resident
Both
Migrant
−1.0 −0.5 0.0 0.5 1.0
c) Entropy in social community graph
d) Entropy in monetary community graph

Figure 13.2: *Community graphs, for social layer - left - (105 communities, 205 inter-community edges) and monetary layer - right - (76 communities, 252 inter-community edges). In a) and b) community node size is proportional to its density. In c) and d) node size is proportional to its community entropy. We use colors to represent the majority between migrant and resident nodes: communities with more residents will go towards sky blue, while more migrants will lead to red nodes, and white is for nodes with a balanced mix of both. Edge width is proportional to the weight of the inter-community edge. The position of the nodes is determined by the visualization library Gephi [174] by leveraging connectivity in a force-based layout.*

Visualizing the interplay

Taking into account only node coloring and their *connectivity*, we see that migrant communities tend to be on the periphery of the community graph, with few or no inter-community links. This characteristic can be found in both monetary and social layers. We can also see that the community graphs have a more central part, which is made up of very connected communities with the majority of members being residents. In contrast, only a few communities with a majority of migrants are linked to the central core of the community graphs. The isolation of migrant nodes and communities is a first important indication of the significance of network structure: for a community, being marginal may be a trait that leads to the majority of its members migrating. We proceed in our analysis of group structure by taking *density* and *entropy* into account as community features. For the evaluation of the impact of community density on migration, we focus on the size of the community nodes, for the social community graph in Figure 13.2a and for the monetary community graph in Figure 13.2b. A visual examination of the network representation of the social community graph reveals that there is no clear distinction in terms of density among resident and migrant communities: we find both migrant and resident communities among the densest communities. On the other hand, in the monetary layer, we can see that the resident communities —more specifically, those in the network's central region— have the highest density values. For visual analysis of entropy values, we vary the size of communities of the social community graph in Figure 13.2c and at the monetary community graph in Figure 13.2d, according to their entropy. We observe that entropy values are pretty similar in the social layer. Entropy values are high across all communities in the social layer, and we are unable to distinguish any specific differences. On the other hand, we see a more varied situation on the monetary layer. In this layer, we can observe that the communities in the central part are characterized by low entropy values. Moreover, they tend to be composed of a majority of resident users and are more likely to connect with other resident communities. In addition, we observe high entropy values in isolated communities, but there is no distinction between resident and migrant communities. So while there are some differences in network structure between the considered layers, overall, entropy does not help characterize the two groups.

Measuring the interplay

We then move on to the quantitative analysis of the interplay between the network structure (density and entropy) and the migration decision (inactive, resident, migrant). We computed correlation statistics between the selected community features, taking into account the communities on the social and monetary layers. In Table 13.2, on the left side, we report correlation measures for the social communities.

Table 13.2: *Correlations on community properties in the social layer on the left and on the monetary layer on the right. p-values are reported in parenthesis.*

	density	entropy
inactive	-0.187 (0.057)	0.176 (0.073)
resident	-0.123 (0.211)	0.025 (0.797)
migrant	-0.075 (0.448)	0.357 (0.005)

	density	entropy
inactive	-0.296 (0.009)	0.164 (0.157)
resident	0.583 (0.0)	-0.209 (0.07)
migrant	-0.275 (0.016)	-0.060 (0.608)

We can observe that for the social communities, density has a slightly negative correlation with the number of resident and inactive users, while there is no correlation with the number of migrant users, which is consistent with the earlier network-based visual inspection. In terms of entropy, we observe a significant positive correlation (p-value ≤ 0.005) with the volume of migrants. On the right side of Table 13.2 we show the measurements computed with the communities in the monetary layer. Density has a moderately positive correlation with the number of residents, but a negative correlation with the presence of migrant nodes. These observations are in line with the network-based visual analysis, which revealed that density characterized monetary communities made up of residents, while migrants tend to be more loosely connected. Similarly, entropy shows a slight negative correlation with the number of resident nodes. So even at the quantitative level, we can confirm that group density can characterize users at a mesoscopic level. On the contrary, entropy does not seem to be helpful in the characterization of the groups.

Therefore, regarding the interplay of group network structure and user migration (RQ1), we can conclude that i) the "position" of a group within the network of social and economic interactions is related to the likelihood of a group to migrate, i.e. marginal groups are more likely to leave; ii) users in densely connected groups interacting through monetary transactions are more likely to stay, and iii) user migration differently affects on the network built on social interactions and the network based on monetary transactions.

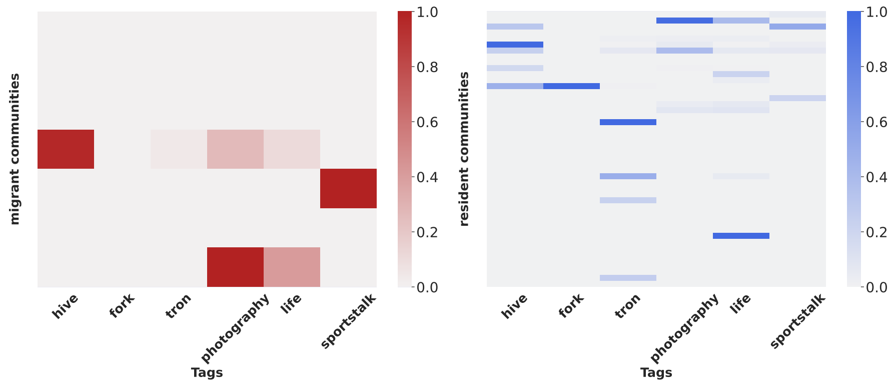
13.4.2 The interplay of community discussion and migration

In this section, we answer our research question on the interplay of group discussions and user migration (*RQ2*). We first present the results obtained by the analysis of hashtags, followed by the analysis of topics.

Hashtags and communities

To study the interplay with community structure, we compare hashtag usage across communities. We rely on communities obtained considering either social interactions or financial interactions to compute the community hashtags distributions and compare them through a heatmap plot, as described in Section 13.2. For easier comparison, we group communities into those with a majority of resident users and the ones with a majority of migrant users. We first consider the communities on the social layer in Figure 13.3, where each row represents a community on the social layer and its community hashtag distribution. Overall, there is a difference in hashtag distribution between migrant and resident users, but there is not clear distinctive trait. Moreover, migration-related hashtags are not limited to migrant communities, discussion involves resident users as well: on the social migrants' side (13.3a), we can see how some of the communities barely use the selected migration-related tags. We can observe a community discussing using *hive* hashtag much more than other ones, while other communities seem to be focused on other hashtags as often. Moreover, the usage of migration-related hashtags is not limited to migrants: when we consider communities with a majority of residents (13.3b), we can observe a few communities using migration-related hashtags a lot, especially *hive* and *Tron*.

We then analyze hashtag distribution, focusing on communities on the monetary layer. We report the hashtag distributions in Figure 13.4. Overall, here we find more differences between migrant and resident users, as we see higher usage of migration-related hashtags on the migrant side. In fact, when we consider the migrant communities (13.4a), we observe quite a few communities using migration-related hashtags especially *hive* and *tron* often. On the other hand, resident monetary communities (13.4b), tend to use migration-related hashtags rarely, except for one community. Overall the communities exhibit very different hashtag distributions, but there is not a clear trend distinguishing migrant communities from resident ones.



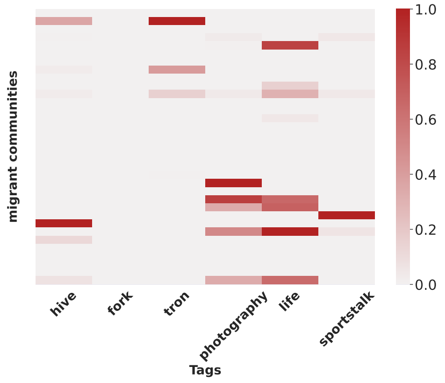
(a) Migrant communities *(b) Resident communities*

Figure 13.3: *Social community hashtag distribution. Heatmaps represent communities on the social layer and their most used hashtags. On the X-axis a selection of hashtags and on the Y-axis the communities, in figure a) migrant communities and in b) resident communities. Values in each cell correspond to the frequency (count) of a hashtag in a community, min-max normalized by hashtag.*

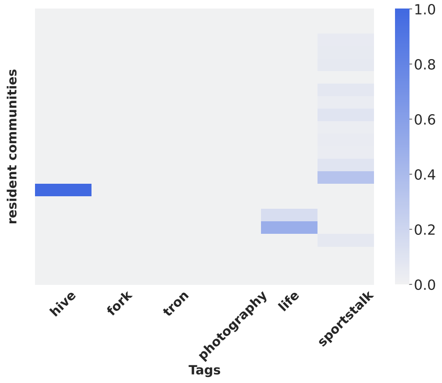
Topics and communities

In this section, we present the results obtained by applying the methodology proposed in Section 13.2 for the analysis revolving on *content topics*. We first observe the obtained topics and their most important words in Table 13.3.

We can see how topics are varied, from topics of general interest such as food, nature, and so on, while other topics are more focused on the economical and technical aspects of the platform and the blockchain world. For easier comprehension, for each topic, we assigned a label based on its most important words. Most labels are self-explanatory, but we briefly go over each label for a better understanding of the following analyses. Topic *Platform* is characterized by terms related to the platform and others related to migration. Topic *Monetary* is characterized by cryptocurrency-related terms; similarly, the *Investments* topic is characterized by keywords related to finance (price, forecast, open). Topics like *Food*, *Nature*, and *Positive* tend to have terms of general interest. *DApps* stands for Decentralized APPlications, i.e. applications that run on top of the hosting blockchain and the corresponding *DApps* topic re-



(a) Migrant communities



(b) Resident communities

Figure 13.4: *Monetary community hashtag distribution. Heatmaps represent communities on the social layer and their most used hashtags. On the X-axis a selection of hashtags and on the Y-axis the communities, in figure a) migrant communities and in b) resident communities. Values in each cell correspond to the frequency (count) of a hashtag in a community, min-max normalized by hashtag.*

unites discussion over some of them. For instance, Dtube [2]is a video-sharing platform with a cryptocurrency-based reward system; while Actifit is another Dapp for fitness enthusiasts [3]. *Appics* is another DApp, similar to Steemit, [4] that relies on the Steem blockchain. Some gaming content is reunited in the *Games* topic. Finally, it seems that while non-English posts are removed, there is a significant community discussing china-related topics. Overall, the choice of 10 topics produced coherent topics. Hence we can proceed with the analysis of the interplay between topics and user migration.

We now consider the topic distributions that characterize the communities. We apply the methodology to compute *community topic vectors* (see section 13.2). We first start with communities on the social layer: the obtained *social community topic vectors* are shown in Figure 13.5. As a general observation, we can see that the migrant social communities detected tend to have lower values overall, while on the resident side, we find more communities and we can observe more interest peaks in the values. When we focus on topics, we

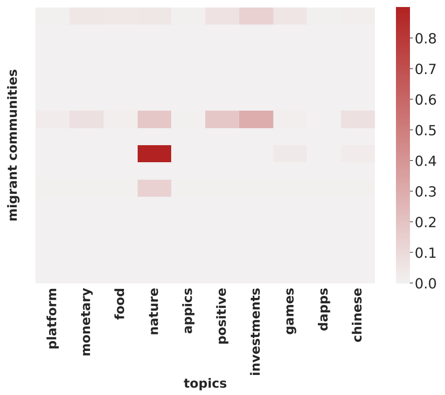
[2] `https://d.tube/`
[3] `https://actifit.io/`
[4] `https://www.appics.com/`

Table 13.3: *Top 10 stemmed keywords for each topic detected with LDA topic model.*

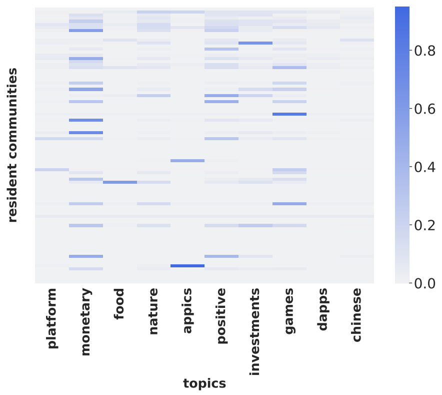
Label	**Top 10 Keywords (stemmed)**
Platform	’steem’, ’commun’, ’steemit’, ’vote’, ’power’, ’post’, ’blockchain’, ’tron’, ’hive’, ’justin’
Monetary	’token’, ’crypto’, ’blockchain’, ’user’, ’invest’, ’bitcoin’, ’platform’, ’coin’, ’account’, ’cryptocurr’
Food	’jpeg’, ’food’, ’coffe’, ’cook’, ’restaur’, ’fresh’, ’tast’, ’fruit’, ’weight’, ’rice’
Nature	’walk’, ’beauti’, ’time’, ’flower’, ’like’, ’todai’, ’activ’, ’home’, ’natur’, ’place’
Appics	’appic’, ’amazonaw’, ’content’, ’east’, ’categori’, ’author’, ’hashtag’, ’caption’, ’permlink’, ’profileimageurl’
Positive	’like’, ’peopl’, ’time’, ’know’, ’thing’, ’want’, ’life’, ’think’, ’love’, ’feel’
Investments	’open’, ’deal’, ’forecast’, ’market’, ’rate’, ’price’, ’expect’, ’coronaviru’, ’post’, ’year’
Games	’game’, ’video’, ’plai’, ’imag’, ’link’, ’steemhunt’, ’post’, ’view’, ’youtub’, ’screenshot’
Dapps	’post’, ’upvot’, ’photo’, ’themarkymark’, ’actifit’, ’dtube’, ’steem’, ’contest’, ’vote’, ’follow’
Chinese	’chines’, ’center’, ’mandarin’, ’btdx’, ’ccenter’, ’dtube’, ’http’, ’jesu’, ’class’, ’muslim’

can see that on both sides, we do not find communities mainly interested in the platform and migration-related topics. Among the topics of interest for the resident groups, the *Monetary* topic emerges, as it seems the focus in many communities. Also, we can see that other topics tend to be of interest across multiple communities such as *Nature*, *Positive*, *Investments*, *Games*, while interest in the remaining topics seem to be limited to only a few communities.

Finally, we consider the *monetary community topics vectors* in Figure 13.6. The first observation is that overall, there is a difference in topics of interest between migrant and resident users: on the migrant side, we can observe more often peaks in the values, while on the resident side, values are generally more distributed across all topics. When we focus on topics, we can see that there is a strong difference concerning the *Platform* topic: communities on the migrant side often have high values in this topic. There is a greater interest by migrant users on the platform and migration-related discussions. On the contrary, we can see how *Monetary* topic peaks are actually more frequent on the migrants' side as well; a similar observation can be made for *Nature*. When it comes down

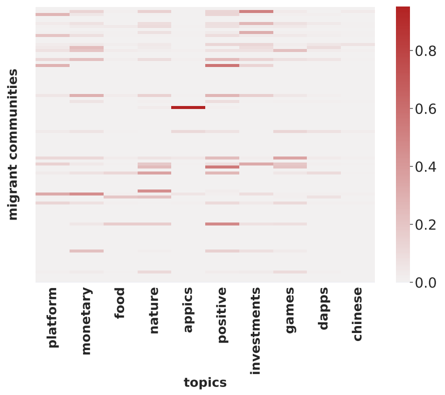


(a) Migrant communities

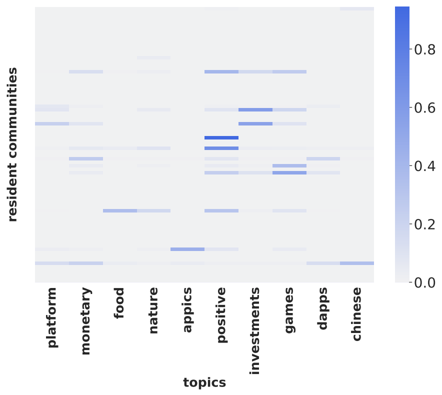


(b) Resident communities

Figure 13.5: *Social community topics vectors. Heatmaps represent communities on the social layer and their topics of interest. On the X-axis topics and on the Y-axis the communities, in figure a) migrant communities and b) resident communities. Values represent the interest in each topic.*



(a) Migrant communities



(b) Resident communities

Figure 13.6: *Monetary community topics vectors Heatmaps represent communities on the social layer and their topics of interest. On the X-axis topics and on the Y-axis the communities, in figure a) migrant communities and b) resident communities. Values represent the interest in each topic.*

to other topics the difference is less evident: on both sides, we can see that *Food*, *Dapps*, *Chinese*, *Appics* are actually limited to only a few communities. While *Positive*, *Investments*, *Games* tend to be more spread out and of interest to more communities, on both sides.

Therefore, regarding the interplay of group discussions and user migration (RQ2), we can conclude the following: i) between migrant and resident users there is a difference in hashtag distributions as well as topics of interest; ii) social communities migration-related hashtags and migration-related topics involve both migrant and resident users; and iii) vice-versa on the monetary layer, we see a clearer interest by migrant users in migration-related hashtags and topics.

13.5 Conclusions

In this work, we addressed the open problem of user migration due to hard fork events occurring in BOSN. Specifically, we investigate user decision-making to stay (resident) or leave (migrant) the platform by leveraging network structure and user-generated text content. Our findings on the impact of network structure, such as the crucial of density, show that structural information, derived from user interactions, should be considered for analysis and prediction tasks. Our findings on differences related to text content and user discussion show how the groups could be also characterized by the content they post and share, something that can be useful not just for predicting user migration but also for the analysis and understanding of causes and dynamics during conflict or turning-point events in a social platform. In general, understanding user migration is of high importance for both traditional social platforms and blockchain online social network platforms that are trying to retain their users as they grow, but also for new alternative platforms trying to emerge. These findings could be useful to other blockchains, as they show the importance of designing proper consensus protocols to handle turning-point events. Besides user migration, the representation for the blockchain data modeling might be applied to a few phenomena characterizing the Web3, for instance, the trading networks generated by NFT (not-fungible token) exchanges or other kinds of social and financial interaction mediated or fueled by DApps, such as games or thematic social networks.

Chapter 14

Migration prediction

Recent studies [162, 175, 163, 164, 121] highlight user migration in online social media (BOSM) platforms as an understudied research area with significant methodological gaps. The primary challenge lies in predicting user migration accurately, especially when limited user information is available. While interaction graph-based approaches show potential, graph neural networks (GNNs) remain unexplored in this domain, despite their proven effectiveness in other machine learning tasks. Graph neural networks offer unique advantages such as eliminating the need for feature engineering and demonstrating predictive power without extensive contextual user information. They can effectively handle complex interaction graph structures and provide nuanced insights into user migration patterns. A critical issue in user migration prediction is class imbalance, where dataset class sizes vary substantially [176]. Current techniques primarily focus on data-level modifications through sampling strategies, particularly oversampling. However, these methods can introduce bias, especially when sufficient class samples exist. Existing research notably lacks comprehensive undersampling approaches. This research gap leads to two key research questions:
RQ1: Are graph neural networks a suitable method for user migration prediction?
RQ2: Can we improve performance in cases of severe class imbalance with a balancing method following an undersampling approach?

To address the identified research gaps, we developed a comprehensive machine learning pipeline specifically designed to investigate user migration in Blockchain Online Social Media (BOSM) platforms. Our approach models user interactions as a directed temporal multilayer graph, capturing both social and monetary interactions to predict user behavior through a sophisticated classification task. We employed graph neural networks to verify their effectiveness in user migration prediction, leveraging data from Steemit and Hive platforms as detailed in Chapter 6. Simultaneously, we introduced an innovative data-level balancing technique following an undersampling approach, which we integrated and compared within the same predictive pipeline. Our approach for the selection of the best model and the proposed balancing approach have highlighted some interesting insights. Graph neural networks are an effective method to predict user migration in blockchain-based online social networks: the GNN model is able to use graph structure on the graph of monetary interactions, even with moderate data unbalance; however, the GNN model struggles on the graph of social interactions that is characterized by severe data imbalance (*RQ1*). However, after applying our proposed data-level balancing approach that produces a more balanced training set, graph neural networks show good predictive power even on severely imbalanced data (*RQ2*).

14.1 Related work

Machine learning on graphs

Over the past decade, machine learning techniques for graph-based tasks have undergone significant evolution. Initially, researchers relied on manual feature generation, creating statistical vectors for each node that could be input into traditional learning models. These early approaches were characterized by their time-consuming nature and lack of adaptability to the learning process. The emergence of graph representation learning marked a pivotal shift in this field. This approach focuses on encoding structural graph information into low-dimensional latent spaces, allowing for more dynamic and flexible analysis. Graph neural networks (GNNs) have rapidly become the state-of-the-art methodology, demonstrating exceptional performance across various complex tasks, such as node classification [177], link prediction [178], community detection [179] and graph classification [180]. GNNs were designed to perform predictions by leveraging both topology and graph attributes by redefining basic deep learning operations, such as convolution, for graph-structured data. The concept has been formalized as the *message passing framework* [181]: the convolution on graphs can be performed by aggregating the values of each node's features along with its neighboring nodes' features. One of the earliest examples is the Graph Convolutional Network (GCN) model proposed by [182]. Given a graph $G = (V, A, X)$ such that V is the set of vertexes, X

is the node feature matrix, and A the adjacency matrix, at each layer k the embedding h of a node i is updated with the following computation:

$$h_i^{(k+1)} = \sigma\left(\sum_{j\in N(i)} \frac{1}{\sqrt{\widetilde{D}_{ii}\widetilde{D}_{jj}}} h_j^{(k)} W^{(k+1)}\right) \tag{14.1}$$

where $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$ corresponds to the degree of i, computed on A_{ij} the adjacency matrix with self-loops added. The aggregation is order-invariant, (examples of such functions are average or summation). The number of layers of a GNN defines the number of hops up to which a node will receive information. Starting from these, we have seen the proposal of many architectures such as GAT [183], graph autoencoders [184], GraphSAGE [177], and many more, to cover different tasks and types of graph data.

Class imbalanced learning on graphs

Class imbalance represents a significant challenge in machine learning, occurring when target class sizes in a dataset differ substantially [176]. This phenomenon affects critical domains such as fraud detection, disease diagnosis, anomaly detection, and sentiment analysis. Imbalanced data samples can severely compromise model performance, particularly for minority classes. Models struggle to learn minority class characteristics, leading to poor generalization and a tendency to predict majority class instances. While class imbalance remains a complex problem, researchers have developed mitigation strategies categorized into data-level and algorithm-level methods [185]. Data-level methods, which modify training data distribution, offer greater flexibility by allowing use of existing models. These approaches typically employ sampling techniques like under-sampling majority classes, over-sampling minority classes, or hybrid approaches. However, traditional sampling methods face significant limitations when applied to graph-based data [186]. Graph learning introduces unique complexities: removing nodes or edges fundamentally alters graph structure, potentially disrupting message-passing processes in graph neural networks. Similarly, adding nodes requires careful management of both attributes and connectivity. Current research predominantly focuses on over-sampling approaches for generating synthetic minority data instances [185]. However, generating artificial data when sufficient samples exists risks introducing dataset bias. Critically, existing works notably lack comprehensive undersampling methodologies for addressing class imbalance in graph-based learning environments.

14.2 Research questions

This chapter addresses the underexplored challenge of user migration prediction in Web3 platforms, developing a machine learning pipeline to forecast user behavior across different platform scenarios: migration, platform retention, dual platform engagement, or user inactivity. Through this investigation, we aim to answer two critical research questions:
Research question 1 (RQ1): Are graph neural networks a suitable method for user migration prediction?
Research question 2 (RQ2): Can we improve performance in cases of severe class imbalance with a balancing method following an undersampling approach?
Our research seeks to advance understanding of user migration in Web3 platforms, offering insights that can significantly improve platform design, user experience, and strategic management of digital ecosystems.

14.3 Methodology

Our research leverages user interaction data to predict migration decisions, adopting a graph-based machine learning approach similar to previous work [121]. By treating user migration as a multiclass node classification problem, we encode user behavior into distinct classes and utilize network structure for predictions. In this section, we define the machine learning pipeline (depicted in Figure 14.1), that will be used to perform the user migration prediction task.

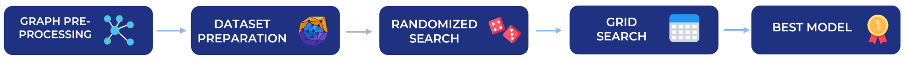


Figure 14.1: *The proposed methodology to solve node classification tasks.*

In the following, we describe the methodology adopted in each step, which will allow us to leverage interaction data as input for machine learning models, to verify the effectiveness of graph neural networks in the setting of a user migration prediction task, as well as to address the class imbalance in datasets.

Modeling user interactions and user decisions: graphs and labels

User interactions can be modeled as a set of tuples $I = (u, v, t, r)$, where u and v are users, who explicitly or implicitly interact at time t through an action of type r. We are interested in the graph structure before the fork, so we can consider the links before t_{Fork} (March 20th, 2020, 2:00 PM), which we denote as $I_{t_{Fork}}$. From this subset of interactions, we are able to build a temporal directed multilayer graph [187], that we denote as $\mathcal{G} = \{\mathcal{G}^r_{t_{Fork}} \forall r\}$, where each element $\mathcal{G}^r_{t_{Fork}}$ is a layer of the multilayer graph. More precisely for each interaction type r, a layer of the graph can be seen as a temporal weighted graph $\mathcal{G}^r_{t_{Fork}} = (V^r_{t_{Fork}}, E^r_{t_{Fork}})$ that stores the interactions of type r that happened up to t_{Fork}. Each edge $(u, v, t, c) \in E^r_{t_{Fork}}$ encodes the operations from node u to node v, described by the counter c and timestamp t. Specifically, the counter c keeps track of the number of operations within the directed pair of nodes, while the timestamp t corresponds to the time of the first operation from u to v. While the obtained graphs could be used to perform prediction on all users, they may have not been active before the fork, therefore it is important to filter users that stopped using the platform before the fork event. We define a set U of users of interest, in which we consider only users active before the fork while including new users that would appear in the following time period. Similarly to what has been done in [121], a user u belongs to the set U (therefore *active*) if it performed at least one operation in the 3 months before the fork event. In this way, we are able to extract $G_{t_{fork}}$, i.e. the subgraph of $\mathcal{G}_{t_{fork}}$ induced by the set U of active nodes. If we consider the set of $r \in \{monetary(m), social(s)\}$, we can denote the layer graphs $G^m_{t_{Fork}}$ and $G^s_{t_{Fork}}$, representing monetary interactions and social interactions respectively, that will be leveraged to predict behavior after the fork. We then need to process interaction data to encode user behavior after the fork, in a way that can be learned by machine learning models. This means defining labels for each node based on the user activity after the fork. If we observe the interactions that happened after the fork event involving a user u, we can consider 4 possible cases:

- *resident*: a user active only on the original platform (Steemit)
- *migrant*: a user active only on the new platform (Hive Blog)
- *co-active*: a user that performs actions on both platforms
- *inactive*: a user that stops using both platforms

These cases are defined at the end of an observation period, after the fork event, considering the activity up to the last interaction in the available data. So each user a is assigned to one of the four labels after observing the interactions $I = (u, v, t, r)$ where $u = a$ and with $t > t_{Fork}$. The assigned label (*resident*, *migration*, *co-active*, or inactive) is defined as the *migration decision* l of user u.

Leveraging graph neural networks: model training and best model selection

The first step is the selection of the architecture for our machine learning model: we selected the GCN architecture from [182]. We implemented a similar version, as represented in Figure 14.2. First, the input features and the adjacency matrix are then leveraged by two graph convolutional layers that create node embeddings. Finally, a linear transformation layer uses the embeddings generated by the GNN, to return a vector with a dimension equal to the number of target classes of the task. Then we obtain a vector representing a probability distribution on the target classes by applying to the output of the previous layer z a softmax function $\sigma(z)$.

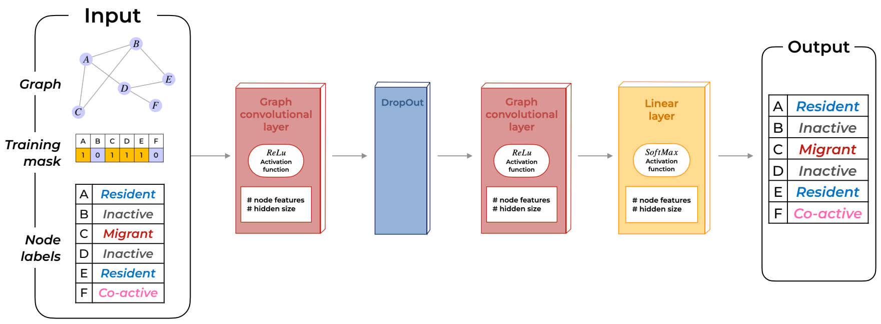


Figure 14.2: *Representation of selected graph neural network architecture. The selected architecture is inspired by the classical GCN architecture by Kipf [182].*

The selected graph neural network model needs to be trained, i.e. its weights need to be adjusted so that it can learn to predict the right classes. When the ground truth is available, GNNs can be trained in a supervised setting. For node classification tasks, supervised learning requires the so-called train-test split [188]. While in traditional machine learning tasks, the split requires the separation into two sets of training samples, when dealing with graphs, the split is not as straightforward: for graph neural networks, the training and test sets are defined as the creation of masks $M_1 \in \mathbb{R}^n$, like in Figure 14.3. The masks indicate which labels should be visible for the GNN model during training.

As in traditional supervised learning frameworks, the objective is to make the model output as close as possible to the ground truth values. This is done

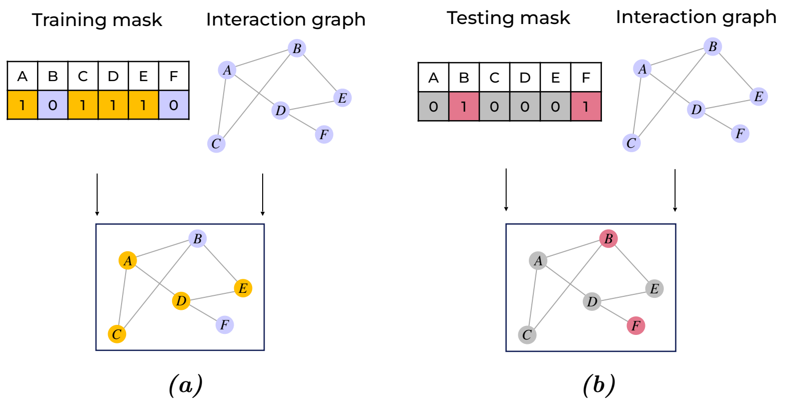


Figure 14.3: *Supervised training example. On the left side, the training mask is defined, while on the right side, an example of the corresponding test mask. Training and message passing are performed using the complete graph structure, but the loss function is computed only for training nodes. During testing, message passing is performed over the entire graph, but evaluation is conducted on test nodes.*

by adjusting model parameters through the data learning process to minimize a loss function. For classification problems, a commonly used loss function is cross-entropy loss [182]. In addition to tuning model parameters, selecting the optimal model configuration for a task involves optimizing hyperparameters, i.e., parameters that cannot be estimated during the learning process and must be predefined as they determine the model architecture [189]. Exploring all possible combinations of hyperparameters within a predefined grid — a process known as *grid search* — can be computationally expensive and time-consuming, especially for Graph Neural Network (GNN) models, which often have numerous hyperparameters, leading to an extensive search space. To mitigate this, a widely adopted approach is *random search* [190]:where only a randomly selected subset of hyperparameter combinations is evaluated.In this Chapter, An initial exploratory phase employs random search to identify promising configurations. These configurations are then refined to narrow down the candidate set, thereby reducing the number of combinations to evaluate during a subsequent full grid search.

Dealing with class imbalance: a new undersampling based approach.

Formally, in a multiclass supervised learning task, there are m classes in total, $\{C_1, ...C_m\}$, and $|C_i|$ is the size of the i-th class, referring to the number of samples belonging to that class.

Here, we introduce an under-sampling technique to balance the distribution of the target variable at the data level. Formally we balance the target variable as follows: we choose a percentage p, and compute the number of samples $n = min_i|C_i| * p$ to get the number of samples per class to include in the training set. To build a balanced training set, we perform under-sampling of each class C_i: we consider a random subset of cardinality n of samples, creating a uniform distribution. This leads to a reduced training set size, but each target class is equally represented. In Figure 14.4, we report a toy example with two classes. The selected method can be applied seamlessly in the pipeline we described previously in Figure 14.1.

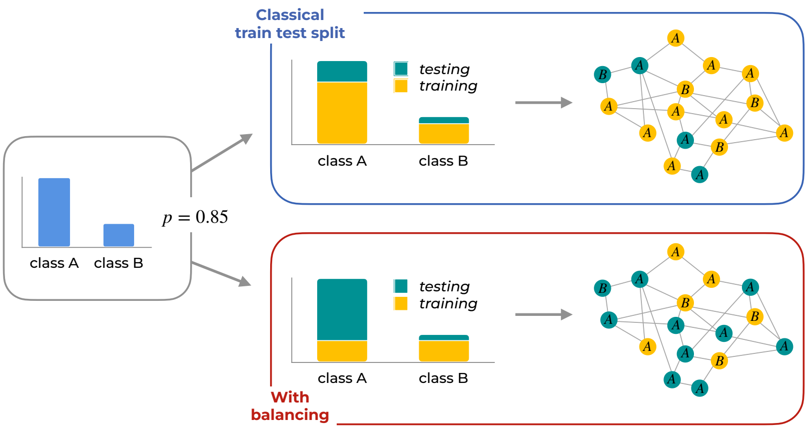


Figure 14.4: Train-test split with unbalanced classed. A visual example of an imbalanced dataset with 2 classes (A, B). On the top half, a representation of a classical 85/15 train-test split: in this case, the training set presents more examples of class A (9) than class B (3). On the lower half, we illustrate our proposed approach: we select 85% of the minority class B as training data, and the same number of examples is kept for the other classes. The obtained training set will present the same number of training nodes for each class (3).

Experimental setting

In this chapter, we evaluate graph neural network performance for user migration prediction using standard multiclass classification metrics. We focus on accuracy and weighted F1 score [185], computed through true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Specifically true positives (TP) and true negatives (TN) represent the number of accurate classifications of positive and negative samples, while false positives (FP) and false negatives (FN) indicate the number of incorrect classifications of positive and negative samples. Accuracy measures the proportion of correctly predicted observations, calculated as $accuracy = \frac{TP+TN}{TP+TN+FP+FN}$. While the $F1 = \frac{2*TP}{2*TP+FP+FN}$, represents the average of $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$. However, in multiclass classification, with C classes and N samples, F1 can be adjusted to account for each class size, leading to the *weighted F1* $= \sum_{i=1}^{C} w_i * F1(C_i)$, as the weighted average of class-wise F1 scores, where for each class C_i, we have a weight $w_i = \frac{|C_i|}{N}$. The metrics are evaluated both on training and test sets: to obtain more robust results. It is common in the literature to consider the average performance over multiple random seeds for each combination, therefore we report the average over 3 random seeds as done in [191]. Through the selected metrics, we compare the predictive performance of graph neural networks to two baseline classifiers: the *Uniform Baseline classifier* that generates predictions uniformly at random (hence it will make a correct prediction in around 1/4 of the cases) and the *Most Frequent Baseline classifier*, which predicts always the most frequent class observed in the training set.

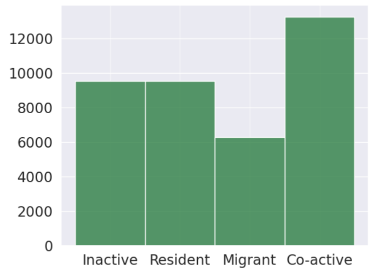
For RQ1, data is separated into training and test sets employingy a $70-30$ random train-test split. Research question two involves applying an undersampling technique with varying training and test set configurations. In this Chapter, for both RQ1 and RQ2, we focus exclusively on network structure. Therefore, node attributes from the dataset are not considered in the prediction: a constant attribute (equal to 1) is associated with each node. The weight update over the training in this Chapter is done by Adam optimizer [192].

14.4 Results

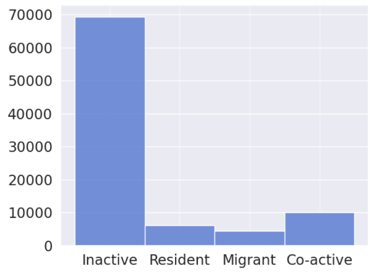
In this section, we present the preprocessed graph and associated labels, demonstrating how our methodology addresses the research questions through detailed graph representation and analysis.

Graph and labels

Applying the proposed methodology on the Steem-Hive dataset , we obtain a multilayer graph $G_{t_{fork}}$. Note that $G_{t_{fork}}$ is the active users' subgraph, i.e. the subgraph induced by the set of active users on the two layers $r \in \{monetary(m), social(s)\}$. The monetary layer graphs $G^m_{t_{Fork}}$ contains $38,566$ nodes connected by $949,046$ edges. While the social layer graph $G^s_{t_{Fork}}$ has $90,055$ nodes and $42,556,877$ edges, Overall, the social layer has more active users and links: this is consistent with the selected operations; in fact, social operations are far more common than monetary transactions. For these users, we encoded their behavior in the 4 possible classes whose frequencies are shown in Figure 14.5a for the monetary interactions and in Figure 14.5b for social interactions. We can observe how the distribution of labels is not balanced: in the monetary layer, there is a slight skew in the number of *co-active* users, and the minority class is composed of *migrant* users. Whereas the social layer is severely imbalanced as the majority of users become *inactive* after the fork event.



***(a)** Decisions on monetary layer*



***(b)** Decisions on social layer*

***Figure 14.5:** The distribution of the generated labels encoding the user migration decision, in the two layers a) monetary and b) social respectively.*

Predicting user migration

We now investigate whether graph neural networks are a suitable method for user migration prediction (RQ1) by applying the methodology presented in Section 14.3 on our dataset. We first train our models for prediction on the graph $G^m_{t_{Fork}}$ representing monetary interactions before the fork. In Table 14.1 we show the obtained results on the monetary layer. The trained GNN model outperforms both Baseline classifiers in accuracy and weighted F1 on the monetary layer, demonstrating its ability to learn from monetary interaction topology. We then perform the prediction task on the *social* layer $G^s_{t_{Fork}}$ that represents social interactions before the fork. In Table 14.2, we show the evaluation results. The gap between the trained model and the baseline classifiers is not as large. The most frequent baseline classifier (the one that predicts the most frequent class observed in training) obtains an accuracy score similar to the best GNN model, while the Uniform Baseline lags severely behind. When we consider the weighted F1 scores we observe a similar trend: the Baseline performs similarly to the GNN model. Further investigation revealed that after few epochs, the model defaults to predicting the most frequent class, indicating significant challenges with severely imbalanced datasets. In the case of a severely imbalanced dataset, the graph neural network model struggles in the prediction of less frequent classes, it acts similarly to the baseline classifier. In general, we can say that the GNN has learned from the input data, making it a suitable model for solving the problem on the monetary layer. While prediction in more imbalanced settings, like in the social graph requires addressing the class imbalance problem.

Table 14.1: *Accuracy and weighted F1 (mean and standard deviation over 3 random seeds [191]) obtained by the Baseline classifiers and the best GNN model on the monetary graph* $G^m_{t_{Fork}}$.

Model	Train accuracy	Test accuracy	Train weighted F1	Test weighted F1
Most freq.	0.346 ± 0.001	0.336 ± 0.003	0.178 ± 0.0011	0.169 ± 0.002
Uniform	0.249 ± 0.001	0.249 ± 0.004	0.253 ± 0.001	0.2515 ± 0.001
Best	$\mathbf{0.426 \pm 0.003}$	$\mathbf{0.424 \pm 0.006}$	$\mathbf{0.381 \pm 0.002}$	$\mathbf{0.379 \pm 0.003}$

Table 14.2: *Accuracy and weighted F1 (mean and standard deviation over 3 random seeds [191]) obtained by the Baseline classifiers and the best GNN model on the social graph* $G^s_{t_{Fork}}$.

Model	Train accuracy	Test accuracy	Train weighted F1	Test weighted F1
Most freq.	$\mathbf{0.770 \pm 0.001}$	$\mathbf{0.770 \pm 0.002}$	$\mathbf{0.671 \pm 0.001}$	$\mathbf{0.670 \pm 0.004}$
Uniform	0.250 ± 0.001	0.250 ± 0.001	0.319 ± 0.001	0.318 ± 0.001
Best	$\mathbf{0.770 \pm 0.001}$	$\mathbf{0.770 \pm 0.002}$	$\mathbf{0.671 \pm 0.001}$	$\mathbf{0.670 \pm 0.004}$

Dealing with class imbalance

In the following, we now analyze how to address class imbalance (RQ2) by applying the methodology presented in Section 14.3. Specifically, we compare the best GNN model obtained for the two layers and evaluate the impact of the balancing approach on performance.

We begin with the monetary layer: Table 14.3 presents the evaluation metrics for both approaches. The models trained on the balanced graph and those trained on the original graph exhibit comparable performance. This is expected, as the target variable in the monetary layer is not significantly imbalanced. Notably, the similarity in performance is a positive outcome since the model trained on the balanced dataset learns from fewer examples but still maintains performance. In fact, we observe slight overall improvements, further underscoring the efficacy of the approach. Next, we turn to the social layer, where the target labels are more imbalanced. Table 14.4 shows the evaluation results, highlighting the impact of the proposed balancing technique. First, we confirm that the model trained on the balanced dataset predicts not only the most frequent class but also other classes. While training on the balanced dataset leads to a drop in both accuracy and weighted F1 on the training sets, the test set performance remains strong, particularly in terms of weighted F1.

Table 14.3: *Accuracy and weighted F1 (mean and standard deviation over 3 random seeds [191]) obtained by the best GNN model trained on the imbalanced training set and the best model trained on the balanced training set, on the monetary graph* $G^m_{t_{Fork}}$ *.*

Model	**Train accuracy**	**Test accuracy**	**Train weighted F1**	**Test weighted F1**
Best imbal.	0.426 ± 0.003	0.424 ± 0.006	0.381 ± 0.002	0.379 ± 0.003
Best bal.	**0.427 ± 0.001**	**0.424 ± 0.001**	**0.386 ± 0.007**	**0.382 ± 0.007**

Table 14.4: *Accuracy and weighted F1 (mean and standard deviation over 3 random seeds [191]) obtained by the best GNN model trained on the imbalanced training set and the best model trained on the balanced training set, on the social graph* $G^s_{t_{Fork}}$*.*

Model	**Train accuracy**	**Test accuracy**	**Train weighted F1**	**Test weighted F1**
Best imbal.	**0.770 ± 0.001**	**0.770 ± 0.002**	**0.671 ± 0.001**	0.670 ± 0.004
Best bal.	0.403 ± 0.002	0.725 ± 0.006	0.359 ± 0.003	**0.788 ± 0.004**

The balancing technique significantly enhances the performance of the GNN model. In datasets with a more balanced class distribution, it enables the model to achieve strong performance while requiring less training data. Conversely, in highly imbalanced datasets, the technique facilitates a more effective learning process by allowing the model to better capture patterns associated with minority classes, thereby improving its predictive capabilities for these underrepresented groups.

14.5 Conclusion

In this Chapter, we tackled the challenge of predicting user migration, emphasizing two key aspects that have received limited attention: the effectiveness of graph neural networks (GNNs) as a predictive framework and the issue of class imbalance, which is particularly pronounced in classification tasks and blockchain-based systems. Our findings demonstrate that GNNs are a powerful tool for forecasting user migration within blockchain-based online social networks. By transforming user interaction data into multilayer temporal graphs tailored for GNN modeling, we developed a methodology that excels at leveraging monetary interaction graphs but faces challenges when dealing with the highly imbalanced social layer. However, by applying our proposed data-level balancing technique to create a more balanced training set, we significantly enhanced the model's predictive performance, even on severely imbalanced data. These results are particularly noteworthy as they underscore the predictive strength of graph-based structures, eliminating the need for manual feature engineering. Additionally, the models exhibit strong performance despite the absence of node features, a common limitation in blockchain-based systems. Looking ahead, future research will explore broader applications of our methodology, as user migration extends beyond online social networks to include transitions between cryptocurrencies, decentralized applications (DApps), and other blockchain ecosystems. The proposed approach holds promise for improving performance in related predictive tasks, such as fraud detection and bot identification, which are critical to both social networks and blockchain environments. Further investigations could also focus on developing alternative balancing strategies to further enhance model effectiveness.

Part V

Conclusions and future works

Chapter 15

Conclusions

In this thesis, we introduce GERANIO, a comprehensive and versatile framework for modeling, mining, analyzing, and representing the evolutionary rules that govern network dynamics. Our primary objective was to develop a robust methodology capable of generating an evolutionary profile for a wide spectrum of network types, offering unprecedented insights into their growth patterns and structural changes over time. As we conclude this research, we can confidently assert that the GERANIO framework represents a significant advancement in the field of network science. Unlike previous approaches that often focused on specific aspects or mechanisms of network evolution, our methodology offers a comprehensive and adaptable solution applicable across diverse domains, from social networks to biological systems and blockchain-based platforms. Our work has yielded several key contributions that collectively advance the field of network science and provide valuable tools for researchers:

- **Framework for Evolving Networks**: We introduce a comprehensive taxonomy for modeling evolving networks, designed to be universally applicable across various domains. This framework unifies diverse modeling approaches from existing stand-alone algorithms and extends to related topics such as user migration, providing a cohesive foundation for network evolution studies.
- **Canonical Coding**: To enable effective comparison of graph evolution across different networks, we develop an efficient categorization method for isomorphic subgraphs. This system ensures the universal applicability of results, facilitating standardized analysis and interpretation of network dynamics.

- **Custom Null Models**: We propose tailored null models specifically designed for graph evolution rule algorithms. These models allow for the extraction of significant rules by assessing their statistical significance beyond simple frequency metrics, enabling researchers to differentiate meaningful patterns from random fluctuations in network evolution.
- **TULIP Algorithm**: Our novel TULIP (Temporal subgraphs for evolutionary profiling) algorithm offers a more comprehensive approach to mining graph evolution rules compared to stand-alone methods. It captures all possible subgraph evolutions along with their associated probabilities, providing a detailed evolutionary footprint of a network.
- **Evolutionary Profile Concept**: To derive insights from mining results, we introduce the evolutionary profile—a probability distribution of rule frequencies. For general graph evolution rules, this concept extends to a multidimensional profile describing the likelihood of various subgraph transformations. These profiles can be aggregated across different networks, offering immediate visual insights into the evolving behavior of graphs, nodes, or groups.

The GERANIO framework's ability to generate detailed evolutionary profiles and identify significant patterns promises to enhance our understanding of complex systems and inform more effective strategies for network management and optimization.

Chapter 16

Future Works

This thesis opens up several promising paths for future research, the following paragraph will explore the different perspective separately.

GER and AI. The first direction concerns the integration of graph evolution rules for and with Artificial Intelligence (AI). One promising direction involves leveraging AI to accelerate the extraction of graph evolution rules through approximate counting methods. This approach could significantly reduce computational costs by avoiding the most expensive part of the algorithm—exact counting—while still maintaining a high degree of accuracy. Conversely, incorporating GER into AI methods offers potential for enhancing network and subnetwork representations. By utilizing evolutionary profiles as features in machine learning tasks, we can provide AI models with rich, dynamic information about network structures and their changes over time. This bidirectional synergy between GER and AI not only promises to accelerate the discovery of meaningful patterns in evolving networks but also to improve the performance and interpretability of AI models working with graph-structured data.

User Strategies Evolutionary profiles offer a promising avenue in the context of computational social science, for analyzing and understanding user strategies in social media environments. By examining the correlation between specific evolutionary patterns and user performance or reputation, we can gain valuable insights into effective growth strategies. For instance, we could investigate whether users whose profiles show a tendency towards expansive evolution—characterized by rapidly forming new connections across diverse network segments—achieve higher levels of popularity or influence compared to those with profiles dominated by reciprocal interactions within a more constrained network neighborhood. This analysis could reveal whether aggressive network expansion pays off more than cultivating deeper, reciprocal relationships in terms of gaining social capital or visibility. Furthermore, by segmenting users based on their evolutionary profiles and comparing these segments against various performance metrics (such as follower count, engagement rates, or virality of content), we can identify which evolutionary strategies are most effective in different contexts or platforms. This approach could also uncover temporal aspects of successful strategies, such as whether users who alternate between periods of rapid expansion and consolidation outperform those with more consistent behavioral patterns. Ultimately, these insights could not only enhance our understanding of social media dynamics but also inform platform design and user growth strategies in digital social environments.

Shocking events Evolutionary profiles offer a powerful tool for detecting shocking events or anomalies in network evolution, both from a node-centric and a broader network-level perspective. By comparing the evolutionary profiles of different time periods, researchers can identify significant deviations that may indicate unusual events or structural changes in the network. This approach allows for a multi-scale analysis, from examining sudden changes in a node's evolutionary profile that could signal shifts in user behavior, to comparing aggregate evolutionary profiles of entire networks across different time periods to reveal large-scale shifts in interaction patterns or structural dynamics. This methodology can be applied bidirectionally. For anomaly detection, establishing a baseline evolutionary profile for "normal" network behavior allows deviations to be flagged as potential anomalies or events of interest. Conversely, for known shocking events, researchers can analyze how the evolutionary profiles of the network changed before, during, and after the event, providing insights into the event's impact on network dynamics.

The user migration scenario presents a possible application of this approach. Here, the event causing the migration can be considered the shocking event, and its effects can be traced through changes in evolutionary profiles. Researchers could analyze how the evolutionary profiles of individual users and the overall network change throughout the migration process. This analysis would cover the periods leading up to, during, and after a migration event. By examining shifts in the network's evolutionary profile as users migrate, researchers may uncover cascading effects or structural reorganizations. Additionally, comparing the evolutionary profiles of users who migrate versus those who don't could reveal patterns that predict migration behavior. These insights could prove valuable for understanding and potentially forecasting user migration dynamics.

Network generation Graph evolution rules offer a promising approach for generating synthetic networks with specific characteristics. By leveraging these rules, researchers can create artificial networks that closely mimic the evolutionary patterns observed in real-world systems. This method allows for fine-grained control over the network generation process. For instance, one could specify a particular evolutionary profile to reproduce, ensuring that the synthetic network exhibits similar growth patterns and structural changes as observed in empirical data. Even more precisely, researchers can define specific rules to govern the network's evolution, allowing for the creation of networks with exact desired properties or behaviors. This capability is particularly valuable for testing hypotheses about network formation, studying the impact of specific evolutionary mechanisms, or generating benchmark datasets for evaluating network analysis algorithms. Furthermore, by adjusting the rules or profiles, researchers can explore "what-if" scenarios, examining how slight changes in evolutionary dynamics might lead to significantly different network structures over time. This approach to synthetic network generation provides a powerful tool for advancing our understanding of complex network dynamics and for developing more robust network analysis methodologies.

By addressing these research directions, we can continue to deepen our understanding of network evolution, enhancing our ability to model, predict, and interpret the dynamics of complex systems across various domains.

References

1. Worldwide Spending on Digital Transformation, `https://cdn.idc.com/getdoc.jsp?containerId=prUS52305724`, [Accessed 08-08-2024].
2. Gartner, Digital Transformation: How to Scope and Execute Strategy, `https://www.gartner.com/en/information-technology/topics/digital-transformation`, [Accessed 12-08-2024].
3. J. Sultan, 34 Digital Transformation Statistics For 2024 — digital-adoption.com, `https://www.digital-adoption.com/digital-transformation-statistics/`, [Accessed 12-08-2024].
4. A.-L. Barabási, Network science, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 371 (1987) (2013) 20120375.
5. J. L. Moreno, Group method and group psychotherapy, no. 5, Beacon House, 1932.
6. M. Coscia, The atlas for the aspiring network scientist (2021). `arXiv:2101.00863`.
7. M. Falkenberg, A. Galeazzi, M. Torricelli, N. Di Marco, F. Larosa, M. Sas, A. Mekacher, W. Pearce, F. Zollo, W. Quattrociocchi, et al., Growing polarization around climate change on social media, Nature Climate Change 12 (12) (2022) 1114–1121.
8. L. L. Cava, L. M. Aiello, A. Tagarelli, Drivers of social influence in the twitter migration to mastodon, Scientific Reports 13 (1) (2023) 21626.
9. E. Ferrara, R. Interdonato, A. Tagarelli, Online popularity and topical interests through the lens of instagram, in: Proceedings of the 25th ACM conference on Hypertext and social media, 2014, pp. 24–34.
10. L. M. Aiello, A. Barrat, R. Schifanella, C. Cattuto, B. Markines, F. Menczer, Friendship prediction and homophily in social media, ACM Transactions on the Web (TWEB) 6 (2) (2012) 1–33.
11. E. Loria, A. Antelmi, J. Pirker, Comparing the structures and characteristics of different game social networks-the steam case, in: 2021 IEEE Conference on Games (CoG), IEEE, 2021, pp. 1–8.

12. A. Tagarelli, R. Interdonato, " who's out there?" identifying and ranking lurkers in social networks, in: Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining, 2013, pp. 215–222.
13. E. Nier, J. Yang, T. Yorulmazer, A. Alentorn, Network models and financial stability, Journal of Economic Dynamics and Control 31 (6) (2007) 2033–2060.
14. D. Savage, Q. Wang, X. Zhang, P. Chou, X. Yu, Detection of money laundering groups: Supervised learning on small networks, in: Workshops at the Thirty-First AAAI Conference on artificial intelligence, 2017.
15. F. Braun, O. Caelen, E. N. Smirnov, S. Kelk, B. Lebichot, Improving card fraud detection through suspicious pattern discovery, in: Advances in Artificial Intelligence: From Theory to Practice: 30th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2017, Arras, France, June 27-30, 2017, Proceedings, Part II 30, Springer, 2017, pp. 181–190.
16. B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, C. Faloutsos, Fraudar: Bounding graph fraud in the face of camouflage, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 895–904.
17. I. A. Kovács, K. Luck, K. Spirohn, Y. Wang, C. Pollis, S. Schlabach, W. Bian, D.-K. Kim, N. Kishore, T. Hao, et al., Network-based prediction of protein interactions, Nature communications 10 (1) (2019) 1240.
18. F. Cheng, I. A. Kovács, A.-L. Barabási, Network-based prediction of drug combinations, Nature communications 10 (1) (2019) 1197.
19. R. Gallotti, M. Barthelemy, The multilayer temporal network of public transport in great britain, Scientific data 2 (1) (2015) 1–8.
20. W. H. Thompson, P. Brantefors, P. Fransson, From static to temporal network theory: Applications to functional brain connectivity, Network Neuroscience 1 (2) (2017) 69–99.
21. P. Holme, J. Saramäki, Temporal network theory, Vol. 2, Springer, New York City, NY, 2019.
22. X. Xue, L. Pan, M. Zheng, W. Wang, Network temporality can promote and suppress information spreading, Chaos: An Interdisciplinary Journal of Nonlinear Science 30 (11) (2020).
23. C. Liu, Z.-K. Zhang, Information spreading on dynamic social networks, Communications in Nonlinear Science and Numerical Simulation 19 (4) (2014) 896–904.
24. C. T. Ba, R. G. Clegg, B. A. Steer, M. Zignani, Investigating shocking events in the ethereum stablecoin ecosystem through temporal multilayer graph structure, arXiv preprint arXiv:2407.10614 (2024).
25. R. Pastor-Satorras, C. Castellano, P. Van Mieghem, A. Vespignani, Epidemic processes in complex networks, Reviews of modern physics 87 (3) (2015) 925.

26. J. Leskovec, J. Kleinberg, C. Faloutsos, Graph evolution: Densification and shrinking diameters, ACM transactions on Knowledge Discovery from Data (TKDD) 1 (1) (2007) 2–es.
27. M. S. Granovetter, The strength of weak ties, in: Social networks, Elsevier, 1977, pp. 347–367.
28. D. Easley, J. Kleinberg, et al., Networks, crowds, and markets, Vol. 8, Cambridge university press Cambridge, 2010.
29. M. Berlingerio, F. Bonchi, B. Bringmann, A. Gionis, Mining graph evolution rules, in: joint European conference on machine learning and knowledge discovery in databases, Springer, 2009, pp. 115–130.
30. X. Yan, J. Han, gspan: Graph-based substructure pattern mining, in: 2002 IEEE International Conference on Data Mining, 2002. Proceedings., IEEE, 2002, pp. 721–724.
31. C. Leung, E.-P. Lim, D. Lo, J. Weng, Mining interesting link formation rules in social networks, 2010, pp. 209–218. `doi:10.1145/1871437.1871468`.
32. M. Yuuki, T. Ozaki, O. Takenao, Mining interesting patterns and rules in a time-evolving graph, Lecture Notes in Engineering and Computer Science 2188 (03 2011).
33. B. Bringmann, S. Nijssen, What is frequent in a single graph?, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2008, pp. 858–863.
34. K. Vaculík, A versatile algorithm for predictive graph rule mining., in: ITAT, 2015, pp. 51–58.
35. E. Scharwächter, E. Müller, J. Donges, M. Hassani, T. Seidl, Detecting change processes in dynamic networks by frequent graph evolution rule mining, in: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, 2016, pp. 1191–1196.
36. T. Junttila, P. Kaski, Engineering an efficient canonical labeling tool for large and sparse graphs, in: 2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX), SIAM, 2007, pp. 135–149.
37. L. Gauvin, M. Génois, M. Karsai, M. Kivelä, T. Takaguchi, E. Valdano, C. L. Vestergaard, Randomized reference models for temporal networks, arXiv preprint arXiv:1806.04032 (2018).
38. S. Lozano, A. Arenas, A. Sanchez, Mesoscopic structure conditions the emergence of cooperation on social networks, PLoS one 3 (4) (2008) e1892.
39. G. Tibely, L. Kovanen, M. Karsai, K. Kaski, J. Kertesz, J. Saramäki, Communities and beyond: mesoscopic analysis of a large social network with complementary methods, Physical Review E—Statistical, Nonlinear, and Soft Matter Physics 83 (5) (2011) 056125.
40. A. Galdeman, M. Zignani, S. Gaito, Disentangling the growth of blockchain-based networks by graph evolution rule mining, in: 2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA), IEEE, 2022, pp. 1–10.

41. L. P. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub) graph isomorphism algorithm for matching large graphs, IEEE transactions on pattern analysis and machine intelligence 26 (10) (2004) 1367–1372.
42. M. Yuuki, T. Ozaki, O. Takenao, Mining interesting patterns and rules in a time-evolving graph, Lecture Notes in Engineering and Computer Science 2188 (03 2011).
43. G. Bianconi, R. K. Darst, J. Iacovacci, S. Fortunato, Triadic closure as a basic generating mechanism of communities in complex networks, Physical Review E 90 (4) (2014) 042806.
44. D. Romero, J. Kleinberg, The directed closure process in hybrid social-information networks, with an analysis of link formation on twitter, Proceedings of the International AAAI Conference on Web and Social Media 4 (1) (2010) 138–145. `doi:10.1609/icwsm.v4i1.14015`.
URL `https://ojs.aaai.org/index.php/ICWSM/article/view/14015`
45. O. Frank, Triad count statistics, in: Annals of Discrete Mathematics, Vol. 38, Elsevier, 1988, pp. 141–149.
46. P. Ribeiro, P. Paredes, M. E. P. Silva, D. Aparicio, F. Silva, A survey on subgraph counting: Concepts, algorithms, and applications to network motifs and graphlets, ACM Comput. Surv. 54 (2) (mar 2021). `doi:10.1145/3433652`.
URL `https://doi.org/10.1145/3433652`
47. V. Batagelj, A. Mrvar, A subquadratic triad census algorithm for large sparse networks with small maximum degree, Social networks 23 (3) (2001) 237–243.
48. G. Chin Jr, A. Marquez, S. Choudhury, J. Feo, Scalable triadic analysis of large-scale graphs: Multi-core vs. multi-processor vs. multi-threaded shared memory architectures, in: 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing, IEEE, 2012, pp. 163–170.
49. S. Parimalarangan, G. M. Slota, K. Madduri, Fast parallel graph triad census and triangle counting on shared-memory platforms, 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (2017) 1500–1509.
URL `https://api.semanticscholar.org/CorpusID:9824095`
50. Y. Santoso, A. Thomo, V. Srinivasan, S. Chester, Triad enumeration at trillion-scale using a single commodity machine, in: Advances in Database Technology-EDBT 2019, 22nd International Conference on Extending Database Technology, Lisboa, Portugal, March 26-29, Proceedings, OpenProceedings. org, 2019.
51. L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, C. Sohler, Counting triangles in data streams, in: Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '06, Association for Computing Machinery, New York, NY, USA, 2006, p. 253–262. `doi:10.1145/1142351.1142388`.
URL `https://doi.org/10.1145/1142351.1142388`
52. A. Pavan, K. Tangwongsan, S. Tirthapura, K.-L. Wu, Counting and sampling triangles from a graph stream, Proc. VLDB Endow. 6 (14) (2013) 1870–1881.

doi:10.14778/2556549.2556569.
URL https://doi.org/10.14778/2556549.2556569

53. R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon, Network motifs: simple building blocks of complex networks, Science 298 (5594) (2002) 824–827.
54. H. Huang, J. Tang, L. Liu, J. Luo, X. Fu, Triadic closure pattern analysis and prediction in social networks, IEEE Transactions on Knowledge and Data Engineering 27 (12) (2015) 3374–3389. doi:10.1109/TKDE.2015.2453956.
55. A. Alrhmoun, J. Kertész, Emergent local structures in an ecosystem of social bots and humans on twitter, EPJ Data Science 12 (1) (2023) 39.
56. L. Kovanen, M. Karsai, K. Kaski, J. Kertész, J. Saramäki, Temporal motifs in time-dependent networks, Journal of Statistical Mechanics: Theory and Experiment 2011 (11) (2011) P11005. doi:10.1088/1742-5468/2011/11/p11005.
URL http://dx.doi.org/10.1088/1742-5468/2011/11/P11005
57. A. Paranjape, A. R. Benson, J. Leskovec, Motifs in temporal networks, in: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 601–610. doi:10.1145/3018661.3018731.
URL https://doi.org/10.1145/3018661.3018731
58. Y. Hulovatyy, H. Chen, T. Milenković, Exploring the structure and function of temporal networks with dynamic graphlets, Bioinformatics 31 (12) (2015) i171–i180.
59. S. Purohit, L. B. Holder, G. Chin, Item: Independent temporal motifs to summarize and compare temporal networks, ArXiv abs/2002.08312 (2020).
60. A. Longa, G. Cencetti, B. Lepri, A. Passerini, An efficient procedure for mining egocentric temporal motifs, Data Mining and Knowledge Discovery 36 (1) (2022) 355–378.
61. A. Ceria, S. Havlin, A. Hanjalic, H. Wang, Topological–temporal properties of evolving networks, Journal of Complex Networks 10 (5) (2022) cnac041.
62. F. E. Faisal, T. Milenković, Dynamic networks reveal key players in aging, Bioinformatics 30 (12) (2014) 1721–1729. arXiv:https://academic.oup.com/bioinformatics/article-pdf/30/12/1721/48926324/bioinformatics_30_12_1721.pdf, doi:10.1093/bioinformatics/btu089.
URL https://doi.org/10.1093/bioinformatics/btu089
63. D. Aparício, P. Ribeiro, F. Silva, Graphlet-orbit transitions (got): A fingerprint for temporal network comparison, PLOS ONE 13 (10) (2018) 1–24. doi:10.1371/journal.pone.0205497.
URL https://doi.org/10.1371/journal.pone.0205497
64. M. Doroud, P. Bhattacharyya, S. F. Wu, D. Felmlee, The evolution of egocentric triads: A microscopic approach toward predicting macroscopic network properties, in: 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing, 2011, pp. 172–179. doi:10.1109/PASSAT/SocialCom.2011.101.

65. M. Zignani, S. Gaito, G. P. Rossi, X. Zhao, H. Zheng, B. Zhao, Link and triadic closure delay: Temporal metrics for social network dynamics, Proceedings of the International AAAI Conference on Web and Social Media 8 (1) (2014) 564–573. doi:10.1609/icwsm.v8i1.14507.
URL https://ojs.aaai.org/index.php/ICWSM/article/view/14507
66. C. T. Ba, M. Zignani, S. Gaito, Characterizing growth in decentralized socio-economic networks through triadic closure-related network motifs, Online Social Networks and Media 37-38 (2023) 100266. doi:https://doi.org/10.1016/j.osnem.2023.100266.
URL https://www.sciencedirect.com/science/article/pii/S2468696423000253
67. Q. Nguyen, O. Poquet, C. A. Brooks, W. Li, Exploring homophily in demographics and academic performance using spatial-temporal student networks, Educational Data Mining (2020).
URL https://www.semanticscholar.org/paper/Exploring-homophily-in-demographics-and-academic-Nguyen-Poquet/fa2bd2a99d28ce9ea98cf2ef59a3b95a025b59e0
68. H. Weber, M. Schwenzer, S. Hillmert, Homophily in the formation and development of learning networks among university students, Network Sci. (2020).
URL https://www.semanticscholar.org/paper/Homophily-in-the-formation-and-development-of-among-Weber-Schwenzer/11b530e9c2610df2fbb16f2e7bb5b6767c8742ba
69. L. Kovanen, K. Kaski, J. Kertész, J. Saramäki, Temporal motifs reveal homophily, gender-specific patterns, and group talk in call sequences, Proceedings of the National Academy of Sciences 110 (45) (2013) 18070–18075. arXiv:https://www.pnas.org/content/110/45/18070.full.pdf, doi:10.1073/pnas.1307941110.
URL https://www.pnas.org/content/110/45/18070
70. P. Kister, L. Tonetto, On the importance of structural equivalence in temporal networks for epidemic forecasting, Sci. Rep. (2023).
URL https://www.semanticscholar.org/paper/On-the-importance-of-structural-equivalence-in-for-Kister-Tonetto/a06dec21cf6fdf85db586adc6ead0d8c05c242cd
71. G. Palla, A.-L. Barabási, T. Vicsek, Quantifying social group evolution, Nature 446 (7136) (2007) 664–667.
72. J. Onnela, Community Structure in Time-Dependent, Multiscale, and Multiplex Networks, Science (2009). doi:10.1016/j.physrep.2012.03.001.
73. D. Greene, D. Doyle, P. Cunningham, Tracking the evolution of communities in dynamic social networks, in: 2010 international conference on advances in social networks analysis and mining, IEEE, 2010, pp. 176–183.
74. D. Kempe, A framework for community identification in dynamic social networks, Knowledge Discovery and Data Mining (2007). doi:10.1016/j.physrep.2012.03.001.

75. F. Giannotti, Tiles: an online algorithm for community discovery in dynamic social networks, Machine-mediated learning (2017). `doi:10.1016/j.physrep.2012.03.001`.
76. R. Agrawal, T. Imielinski, A. N. Swami, Mining association rules between sets of items in large databases, in: ACM SIGMOD Conference, 1993.
77. J. M. Ale, G. H. Rossi, An approach to discovering temporal association rules, in: Proceedings of the 2000 ACM Symposium on Applied computing-Volume 1, 2000, pp. 294–300.
78. C. Jiang, F. Coenen, M. Zito, A survey of frequent subgraph mining algorithms, The Knowledge Engineering Review 28 (1) (2013) 75–105.
79. K. M. Borgwardt, H.-P. Kriegel, P. Wackersreuther, Pattern mining in frequent dynamic subgraphs, in: Sixth International Conference on Data Mining (ICDM'06), IEEE, 2006, pp. 818–822.
80. E. Abdelhamid, M. Canim, M. Sadoghi, B. Bhattacharjee, Y.-C. Chang, P. Kalnis, Incremental frequent subgraph mining on large evolving graphs, IEEE Transactions on Knowledge and Data Engineering 29 (12) (2017) 2710–2723.
81. L. Kovanen, M. Karsai, K. Kaski, J. Kertész, J. Saramäki, Temporal motifs in time-dependent networks, Journal of Statistical Mechanics: Theory and Experiment 2011 (11) (2011) P11005.
82. A. Paranjape, A. R. Benson, J. Leskovec, Motifs in temporal networks, in: Proceedings of the tenth ACM international conference on web search and data mining, 2017, pp. 601–610.
83. N. Pržulj, Biological network comparison using graphlet degree distribution, Bioinformatics 23 (2) (2007) e177–e183.
84. F. E. Faisal, T. Milenković, Dynamic networks reveal key players in aging, Bioinformatics 30 (12) (2014) 1721–1729.
85. B. Bringmann, S. Nijssen, What is frequent in a single graph?, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2008, pp. 858–863.
86. L. Gauvin, M. Génois, M. Karsai, M. Kivelä, T. Takaguchi, E. Valdano, C. L. Vestergaard, Randomized reference models for temporal networks, SIAM Review 64 (4) (2022) 763–830.
87. T. A. Snijders, Statistical models for social networks, Annual review of sociology 37 (2011) 131–153.
88. P. Holme, F. Liljeros, Birth and death of links control disease spreading in empirical contact networks, Scientific reports 4 (1) (2014) 4999.
89. M. Karsai, K. Kaski, A.-L. Barabási, J. Kertész, Universal features of correlated bursty behaviour, Scientific reports 2 (1) (2012) 1–7.
90. J.-C. Delvenne, R. Lambiotte, L. E. Rocha, Diffusion on networked systems is a question of time or structure, Nature communications 6 (1) (2015) 7366.
91. M. Ley, The DBLP computer science bibliography: Evolution, research issues, perspectives, in: Proc. Int. Symposium on String Process. and Inf. Retr., 2002, pp. 1–10.

92. J. Kunegis, KONECT – The Koblenz Network Collection, in: Proc. Int. Conf. on World Wide Web Companion, 2013, pp. 1343–1350. URL `http://dl.acm.org/citation.cfm?id=2488173`
93. internationalbanker, The Enron Scandal (2001) — internationalbanker.com, `https://internationalbanker.com/history-of-financial-crises/the-enron-scandal-2001/`, [Accessed 19-09-2024].
94. S. Nakamoto, et al., Bitcoin: A peer-to-peer electronic cash system, Decentralized Business Review (2008) 21260.
95. S. Kumar, F. Spezzano, V. Subrahmanian, C. Faloutsos, Edge weight prediction in weighted signed networks, in: Data Mining (ICDM), 2016 IEEE 16th International Conference on, IEEE, 2016, pp. 221–230.
96. S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, V. Subrahmanian, Rev2: Fraudulent user prediction in rating platforms, in: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, ACM, 2018, pp. 333–341.
97. J. Leskovec, R. Sosič, Snap: A general-purpose network analysis and graph-mining library, ACM Transactions on Intelligent Systems and Technology (TIST) 8 (1) (2016) 1–20.
98. Grassroots Economics (2023). URL `https://www.grassrootseconomics.org/pages/about-us`
99. L. Doria, L. Fantacci, Evaluating complementary currencies: from the assessment of multiple social qualities to the discovery of a unique monetary sociality, Quality & Quantity 52 (2018) 1291–1314.
100. L. Ussher, L. Ebert, G. M. Gómez, W. O. Ruddick, Complementary currencies for humanitarian aid, Journal of Risk and Financial Management 14 (11) (2021) 557.
101. D. Reppas, G. Muschert, The potential for community and complementary currencies (ccs) to enhance human aspects of economic exchange, Digithum (2019).
102. A. Michel, M. Hudon, Community currencies and sustainable development: A systematic review, Ecological economics 116 (2015) 160–171.
103. A. C. An, P. T. X. Diem, L. T. T. Lan, T. V. Toi, L. D. Q. Binh, Building a product origins tracking system based on blockchain and poa consensus protocol, 2019 International Conference on Advanced Computing and Applications (ACOMP) (2019) 27–33.
104. M. Nadini, L. Alessandretti, F. Di Giacinto, M. Martino, L. M. Aiello, A. Baronchelli, Mapping the nft revolution: market trends, trade networks, and visual features, Scientific reports 11 (1) (2021) 1–11.
105. M. S. Kim, J. Y. Chung, Sustainable growth and token economy design: The case of steemit, Sustainability 11 (1) (2019) 167.
106. C. Li, B. Palanisamy, Incentivized blockchain-based social media platforms: A case study of steemit, in: Proceedings of the 10th ACM Conference on Web Science, 2019, pp. 145–154.

107. J. Leskovec, A. Rajaraman, J. D. Ullman, Mining of massive data sets, Cambridge university press, 2020.
108. P. Fournier-Viger, G. He, J. C.-W. Lin, H. M. Gomes, Mining attribute evolution rules in dynamic attributed graphs, in: International Conference on Big Data Analytics and Knowledge Discovery, Springer, 2020, pp. 167–182.
109. K.-N. T. Nguyen, L. Cerf, M. Plantevit, J.-F. Boulicaut, Discovering inter-dimensional rules in dynamic graphs, in: Proceedings of the 1st International Conference on Dynamic Networks and Knowledge Discovery-Volume 655, 2010, pp. 5–16.
110. B. Guidi, A. Michienzi, L. Ricci, Steem blockchain: Mining the inner structure of the graph, IEEE Access 8 (2020) 210251–210266.
111. T.-H. Kim, H. min Shin, H. Hwang, S. Jeong, Posting bot detection on blockchain-based social media platform using machine learning techniques, ArXiv abs/2008.12471 (2020).
112. K. Kapanova, B. Guidi, A. Michienzi, K. Koidl, Evaluating posts on the steemit blockchain: Analysis on topics based on textual cues, in: Proceedings of the 6th EAI International Conference on Smart Objects and Technologies for Social Good, EAI, 2020.
113. R. Ciriello, R. Beck, J. Thatcher, The paradoxical effects of blockchain technology on social networking practices, in: Proceedings of the Thirty Ninth International Conference on Information Systems, AIS, 2018.
114. B. Guidi, When blockchain meets online social networks, Pervasive and Mobile Computing 62 (2020) 101131.
115. A. Kiayias, B. Livshits, A. M. Mosteiro, O. Litos, A puff of steem: Security analysis of decentralized content curation, ArXiv abs/1810.01719 (2019).
116. U. W. Chohan, The concept and criticisms of steemit, CBRI Working Papers: Notes on the 21st Century, Available at SSRN: http://dx.doi.org/10.2139/ssrn.3129410 (2018).
117. B. Guidi, A. Michienzi, L. Ricci, A graph-based socioeconomic analysis of steemit, IEEE Transactions on Computational Social Systems PP (2020) 1–12. `doi:10.1109/TCSS.2020.3042745`.
118. B. Guidi, A. Michienzi, L. Ricci, Steem blockchain: Mining the inner structure of the graph, IEEE Access 8 (11 2020). `doi:10.1109/ACCESS.2020.3038550`.
119. C. T. Ba, M. Zignani, S. Gaito, The role of cryptocurrency in the dynamics of blockchain-based social networks: The case of steemit, PloS one 17 (6) (2022) e0267612.
120. C. T. Ba, M. Zignani, S. Gaito, Social and rewarding microscopical dynamics in blockchain-based online social networks, in: Proceedings of the Conference on Information Technology for Social Good, 2021, pp. 127–132.
121. C. T. Ba, A. Michienzi, B. Guidi, M. Zignani, L. Ricci, S. Gaito, Fork-based user migration in blockchain online social media, in: Proceedings of the 14th ACM conference on web science, 2022.
122. K. Vasan, M. Janosov, A.-L. Barabási, Quantifying nft-driven networks in crypto art, Scientific reports 12 (1) (2022) 1–11.

123. M. Franceschet, Hits hits art, Blockchain: Research and Applications 2 (4) (2021) 100038.
124. A. Kapoor, D. Guhathakurta, M. Mathur, R. Yadav, M. Gupta, P. Kumaraguru, Tweetboost: Influence of social media on nft valuation, arXiv preprint arXiv:2201.08373 (2022).
125. S. Oh, S. Rosen, A. L. Zhang, Investor experience matters: Evidence from generative art collections on the blockchain, Available at SSRN (2022).
126. A. Ramdas, N. G. Trillos, M. Cuturi, On wasserstein two-sample testing and related families of nonparametric tests, Entropy 19 (2) (2017) 47.
127. M. Jin, Y.-F. Li, S. Pan, Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs, Advances in Neural Information Processing Systems 35 (2022) 19874–19886.
128. R. A. Rossi, N. K. Ahmed, E. Koh, S. Kim, A. Rao, Y. Abbasi-Yadkori, A structural graph representation learning framework, in: Proceedings of the 13th international conference on web search and data mining, 2020, pp. 483–491.
129. R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon, Network motifs: simple building blocks of complex networks, Science 298 (5594) (2002) 824–827.
130. L. Kovanen, M. Karsai, K. Kaski, J. Kertész, J. Saramäki, Temporal motifs in time-dependent networks, Journal of Statistical Mechanics: Theory and Experiment 2011 (11) (2011) P11005.
131. B. Arregui-García, A. Longa, Q. F. Lotito, S. Meloni, G. Cencetti, Patterns in temporal networks with higher-order egocentric structures, Entropy 26 (3) (2024) 256.
132. C. E. Mattsson, T. Criscione, W. O. Ruddick, Sarafu community inclusion currency 2020–2021, Scientific data 9 (1) (2022) 426.
133. R. Mqamelo, Community currencies as crisis response: Results from a randomized control trial in kenya, Frontiers in Blockchain (2021) 44.
134. C. E. Mattsson, T. Criscione, F. W. Takes, Circulation of a digital community currency, arXiv preprint arXiv:2207.08941 (2022).
135. C. T. Ba, A. Galdeman, M. Zignani, S. Gaito, Temporal analysis of cooperative behaviour in a blockchain for humanitarian aid during the covid-19 pandemic, in: Proceedings of the 2022 ACM Conference on Information Technology for Social Good, 2022, pp. 292–299.
136. C. T. Ba, M. Zignani, S. Gaito, Cooperative behavior in blockchain-based complementary currency networks through time: The sarafu case study, Future Generation Computer Systems 148 (2023) 266–279.
137. X.-X. Zhan, C. Liu, Z. Wang, H. Wang, P. Holme, Z.-K. Zhang, Measuring and utilizing temporal network dissimilarity, arXiv preprint arXiv:2111.01334 (2021).
138. T. Milenković, N. Pržulj, Uncovering biological network function via graphlet degree signatures, Cancer informatics 6 (2008) CIN–S680.
139. M. Karsai, H.-H. Jo, K. Kaski, et al., Bursty human dynamics, Springer, New York, NY, 2018.

140. R. Kumar, J. Novak, A. Tomkins, Structure and evolution of online social networks, in: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, 2006, pp. 611–617.
141. S. Asur, S. Parthasarathy, D. Ucar, An event-based framework for characterizing the evolutionary behavior of interaction graphs, ACM Transactions on Knowledge Discovery from Data (TKDD) 3 (4) (2009) 1–36.
142. M. Takaffoli, F. Sangi, J. Fagnan, O. R. Zäıane, Community evolution mining in dynamic social networks, Procedia-Social and Behavioral Sciences 22 (2011) 49–58.
143. M. Goldberg, M. Magdon-Ismail, S. Nambirajan, J. Thompson, Tracking and predicting evolution of social communities, in: 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing, IEEE, 2011, pp. 780–783.
144. S. R. Kairam, D. J. Wang, J. Leskovec, The life and death of online groups: Predicting group growth and longevity, in: Proceedings of the fifth ACM international conference on Web search and data mining, 2012, pp. 673–682.
145. A. Patil, J. Liu, J. Gao, Predicting group stability in online social networks, in: Proceedings of the 22nd international conference on World Wide Web, 2013, pp. 1021–1030.
146. P. Bródka, P. Kazienko, B. Kołoszczyk, Predicting group evolution in the social network, in: Social Informatics: 4th International Conference, SocInfo 2012, Lausanne, Switzerland, December 5-7, 2012. Proceedings 4, Springer, 2012, pp. 54–67.
147. N. Dakiche, F. B.-S. Tayeb, K. Benatchba, Y. Slimani, A. Khelifati, H. Chabane, Epredictor: An experimental platform for community evolution prediction tests., in: SIMULTECH, 2021, pp. 295–302.
148. N. Ilhan, Ş. G. Öğüdücü, Feature identification for predicting community evolution in dynamic social networks, Engineering Applications of Artificial Intelligence 55 (2016) 202–218.
149. X. Wang, M. Zhang, GLASS: GNN with labeling tricks for subgraph representation learning, in: International Conference on Learning Representations, ICLR '22, 2022.
URL https://openreview.net/forum?id=XLxhEjKNbXj
150. X. Chen, Q. Qian, subge: Enhancing the subgraph representation of molecular compounds structure–activity relationship discovery, Eng. Appl. Artif. Intell. 119 (C) (mar 2023). doi:10.1016/j.engappai.2022.105727.
URL https://doi.org/10.1016/j.engappai.2022.105727
151. E. Alsentzer, S. Finlayson, M. Li, M. Zitnik, Subgraph neural networks, Advances in Neural Information Processing Systems 33 (2020) 8017–8029.
152. Y. Yu, Z. Lu, J. Liu, G. Zhao, J.-r. Wen, Rum: Network representation learning using motifs, in: 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, 2019, pp. 1382–1393.
153. J. Xu, A. Yu, L. Cai, D. Meng, Mmnr: A network representation framework based on multi-view motif fusion, in: 2019 IEEE Intl Conf on Parallel

& Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), IEEE, 2019, pp. 792–801.

154. K. Tu, J. Li, D. F. Towsley, D. Braines, L. D. Turner, Network classification in temporal networks using motifs, ArXiv abs/1807.03733 (2018).
URL `https://api.semanticscholar.org/CorpusID:49668820`

155. K. Tu, J. Li, D. Towsley, D. Braines, L. D. Turner, gl2vec: Learning feature representation using graphlets for directed networks, in: Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining, 2019, pp. 216–221.

156. M. Chen, K. Kuzmin, B. K. Szymanski, Community detection via maximization of modularity and its variants, IEEE Transactions on Computational Social Systems 1 (1) (2014) 46–65. `doi:10.1109/TCSS.2014.2307458`.

157. V. A. Traag, L. Waltman, N. J. Van Eck, From louvain to leiden: guaranteeing well-connected communities, Scientific reports 9 (1) (2019) 5233.

158. M. Coscia, M. Szell, Multiplex graph association rules for link prediction, ArXiv abs/2008.08351 (2020).

159. A. Galdeman, M. Zignani, S. Gaito, Unfolding temporal networks through statistically significant graph evolution rules, in: 2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA), IEEE, 2023, pp. 1–10.

160. G. Csardi, T. Nepusz, The igraph software, Complex syst 1695 (2006) 1–9.

161. B. Guidi, A. Michienzi, L. Ricci, A graph-based socioeconomic analysis of steemit, IEEE Transactions on Computational Social Systems 8 (2) (2021) 365–376.

162. S. Kumar, R. Zafarani, H. Liu, Understanding user migration patterns in social media, in: AAAI, 2011.

163. M. Senaweera, R. Dissanayake, N. Chamindi, A. Shyamalal, C. Elvitigala, S. Horawalavithana, P. Wijesekara, K. Gunawardana, M. I. E. Wickramasinghe, C. Keppitiyagama, A weighted network analysis of user migrations in a social network, 2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer) (2018) 357–362.

164. C. Davies, J. R. Ashford, L. Espinosa-Anke, A. D. Preece, L. D. Turner, R. M. Whitaker, M. Srivatsa, D. H. Felmlee, Multi-scale user migration on reddit, in: Workshop on Cyber Social Threats at the 15th International AAAI Conference on Web and Social Media (ICWSM 2021), AAAI, 2021.

165. C. T. Ba, M. Zignani, S. Gaito, The role of groups in a user migration across blockchain-based online social media, in: 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), IEEE, 2022, pp. 291–296.

166. B. Guidi, A. Michienzi, L. Ricci, Analysis of witnesses in the steem blockchain, Mobile Networks and Applications (2021) 1–12.

167. J. Wu, J. Liu, Y. Zhao, Z. Zheng, Analysis of cryptocurrency transactions from a network perspective: An overview, Journal of Network and Computer Applications 190 (2021) 103139.
168. R. Interdonato, M. Magnani, D. Perna, A. Tagarelli, D. Vega, Multilayer network simplification: approaches, models and methods, Comput. Sci. Rev. 36 (2020) 100246.
169. M. Magnani, O. Hanteer, R. Interdonato, L. Rossi, A. Tagarelli, Community detection in multiplex networks, arXiv preprint arXiv:1910.07646 (2019).
170. M. Rosvall, D. Axelsson, C. T. Bergstrom, The map equation, The European Physical Journal Special Topics 178 (1) (2009) 13–23.
171. M. De Domenico, A. Lancichinetti, A. Arenas, M. Rosvall, Identifying modular flows on multilayer networks reveals highly overlapping organization in interconnected systems, Physical Review X 5 (1) (2015) 011027.
172. D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, J. Mach. Learn. Res. 3 (null) (2003) 993–1022.
173. C. T. Ba, M. Zignani, S. Gaito, The role of cryptocurrency in the dynamics of blockchain-based social networks: the case of steemit, PloS One (2022).
174. M. Bastian, S. Heymann, M. Jacomy, Gephi: An open source software for exploring and manipulating networks (2009).
URL http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154
175. E. Newell, D. Jurgens, H. M. Saleem, H. Vala, J. Sassine, C. Armstrong, D. Ruths, User migration in online social networks: A case study on reddit during a period of community unrest, in: ICWSM, 2016.
176. H. Kaur, H. S. Pannu, A. K. Malhi, A systematic review on imbalanced data challenges in machine learning: Applications and solutions, ACM Computing Surveys (CSUR) 52 (4) (2019) 1–36.
177. W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, Advances in neural information processing systems 30 (2017).
178. M. Zhang, Y. Chen, Link prediction based on graph neural networks, Advances in neural information processing systems 31 (2018).
179. J. You, R. Ying, J. Leskovec, Position-aware graph neural networks, in: International conference on machine learning, PMLR, 2019, pp. 7134–7143.
180. Z. Zhang, J. Bu, M. Ester, J. Zhang, C. Yao, Z. Yu, C. Wang, Hierarchical graph pooling with structure learning, arXiv preprint arXiv:1911.05954 abs/1911.05954 (2019).
181. J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, in: International conference on machine learning, PMLR, 2017, pp. 1263–1272.
182. T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).
URL https://openreview.net/forum?id=SJU4ayYgl
183. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, arXiv preprint arXiv:1710.10903 (2017).

184. T. N. Kipf, M. Welling, Variational graph auto-encoders, arXiv preprint arXiv:1611.07308 (2016).
185. Y. Ma, Y. Tian, N. Moniz, N. V. Chawla, Class-imbalanced learning on graphs: A survey, arXiv preprint arXiv:2304.04300 (2023).
186. T. Zhao, X. Zhang, S. Wang, Graphsmote: Imbalanced node classification on graphs with graph neural networks, in: Proceedings of the 14th ACM international conference on web search and data mining, 2021, pp. 833–841.
187. P. Holme, J. Saramäki, Temporal networks, Physics Reports 519 (3) (2012) 97–125, temporal Networks.
188. I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, `http://www.deeplearningbook.org`.
189. L. Yang, A. Shami, On hyperparameter optimization of machine learning algorithms: Theory and practice, Neurocomputing 415 (2020) 295–316.
190. J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization., Journal of machine learning research 13 (2) (2012).
191. J. You, T. Du, J. Leskovec, Roland: graph learning framework for dynamic graphs, in: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2022, pp. 2358–2366.
192. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 abs/1412.6980 (2014).