MINING SOCIAL AND CRYPTOCURRENCY NETWORKS

PH.D. THESIS

Author Ferenc Béres
Institute for Computer Science and Control
Eötvös Loránd Research Network

Supervisor András A. Benczúr
Institute for Computer Science and Control
Eötvös Loránd Research Network

DOI 10.15476/ELTE.2022.171

EÖTVÖS LORÁND UNIVERSITY

DOCTORAL SCHOOL OF INFORMATICS

2022

Acknowledgment

First and foremost, I would like to express my deepest gratitude to my supervisor András Benczúr for his patience, assistance, and continuous guidance throughout the research process.

I am also grateful to my colleagues Domokos Miklós Kelen and Róbert Pálovics from our research group at the Hungarian Institute for Computer Science and Control, who contributed greatly to the success of my work.

My Thesis could not have been the same without István András Seres, with whom we had an excellent working dynamics and addressed several open questions that fascinated the blockchain community.

Finally, I would like to thank my family, especially my wife, Anilla Ella Szirtes, for her endless support and encouragement during my studies. Not to mention the constant help with our little children during the times of COVID-19 restrictions when I was working from home.

The research in this Thesis was supported by the Hungarian Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program.

Contents

Chapter 1

Introduction

In the last decade, network science was flourishing, since graph structures underlie several applications that we use during our daily routine. Social networks are probably the largest source for graph data. Facebook and Instagram both gained billions of new active users during this period[1]. Another important graph data source is related to mobility, flights between cities, ride-hailing, or route-planning applications. Finally, for most cryptocurrencies, there is an underlying transaction network where users exchange their coins or other funds without any governmental or third-party supervision. In contrast to most social network and user mobility platforms, the user interactions for various cryptocurrency networks are available to everyone due to the public nature of the blockchain. That is why Bitcoin and Ethereum, the two most well-known cryptocurrency networks, are also in the focus of my research.

A general problem with graph data is that it cannot be fed to classical machine learning methods in a straightforward way. Algorithms like logistic regression, decision trees or deep neural networks only work well with tabular data. Since the number of neighbors of a node can widely vary, raw network data cannot be considered tabular. One possibility to solve general graph mining tasks such as node classification, link prediction, or community detection is to learn a vector space representation of network nodes for downstream machine learning tasks. Research in the related field of node embedding was recently catalyzed by the Word2Vec algorithm [93] for learning word representations in human language text. The main idea of network embedding is to explore the graph through multiple random walks and feed these node sequences to a neural

[1]https://www.businessofapps.com/data/facebook-statistics/, https://www.businessofapps.com/data/instagram-statistics/

network architecture (e.g. Skip-Gram model) that learns a representation for every node. Since the time complexity to embed a network is linear in the number of vertices, it can be deployed for large networks with millions of nodes.

Unfortunately, linear time complexity in the number of nodes can be prohibitive for real-time dynamic network applications. In many data-intensive tasks where interactions between network participants are constantly arriving over time, we need to update graph mining models regularly to capture the latest changes in the data distribution, such as sudden bursts in popularity or some irregular network behavior. Fitting batch algorithms for large graph snapshots could cause a significant time-delay in the prediction. That is why online graph learning techniques are much preferred in such highly dynamic scenarios.

The main goal of our research is to analyze and model user behavior in social and cryptocurrency networks. Specifically, we intend to answer the following questions:

- What are the main advantages of online graph mining techniques over batch models for large-scale social networks and how to best compare their performance? In our research, we focus on graph centrality and node embedding techniques.

- How to mine cryptocurrency networks with novel network science tools to answer open questions in the domain of cryptoeconomics and privacy?

By collecting various new Twitter and cryptocurrency network data sets, we were among the first to deploy and analyze node embedding models in several network applications such as vaccine skepticism detection or Ethereum address deanonymization.

Our findings are related to the fields of network science and machine learning. In our work, we analyze user interactions in social and cryptocurrency networks as well as user-related metadata that we used to formulate supervised evaluation for most of the addressed graph mining tasks.

1.1 Our contributions

Next, we explain our main results one by one. For each topic, we list our main contributions and the original source of publication. In Section 1.2, we also present the inter-relation of our results.

Temporal networks

In Chapter 2, we introduce temporal networks by first laying out the theoretical background for two dynamic graph computational models, the snapshot-based and the edge stream approaches. In Section 2.1, we rigorously compare these concepts by following the arguments in our recent tutorial:

> András Benczúr, Ferenc Béres, Domokos Kelen, and Róbert Pálovics. Tutorial on graph stream analytics. DEBS '21, page 168–171, New York, NY, USA, 2021. Association for Computing Machinery.

In Section 2.2, we present our Twitter data sets that we collected to quantitatively analyze the performance of selected online graph algorithms using the stream of @-mention edges. Our first algorithm is a temporal walk based centrality metric (Chapter 3), and the other two are online node embedding methods (Chapter 4).

We collected two data sets, *RG17* and *UO17*, related to Roland-Garros 2017, the French Open Tennis Tournament, and to US Open 2017, the United States Open Tennis Championship. For both of these sport events, we collected tweets containing predefined hashtags, and then we extracted the underlying at-mention network. The following properties make *RG17* and *UO17* highly suitable for evaluating algorithms on dynamic graphs:

- Temporal @-mention network: every at-mention link in the graph has a timestamp covering a long time range.
- Large scale: both mention graph contains more than 300K edges and 70K nodes (Twitter accounts).
- Dynamic temporal ground truth information available from an external source, which makes supervised evaluation possible for edge stream based online graph algorithms. Based on the official event schedule, we compiled a binary node relevance label with daily granularity for the nodes of the RG17 and UO17 mention networks.

We first introduced the *RG17* and *UO17* Twitter collections in

> Ferenc Béres, Róbert Pálovics, Anna Oláh, and András A Benczúr. Temporal walk based centrality metric for graph streams. *Applied Network Science*, 3(32):26, 2018.

Both of these data sets were included in a publication that received the best resource paper award at CIKM ’21,

Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Béres, Guzmán López, Nicolas Collignon, and Rik Sarkar. Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. CIKM ’21, page 4564–4573, New York, NY, USA, 2021. Association for Computing Machinery.

Temporal walk based centrality metric for graph streams

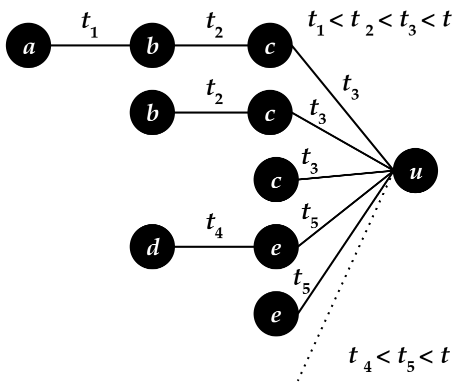


Fig. 1.1: Definition of temporal Katz centrality: weighted sum of time-respecting paths ending at node u up to time t.

In Chapter 3, we will investigate network centrality measures [24]. We propose the temporal Katz centrality, an extension of the successful Katz index [72] suitable for dynamic graphs. As illustrated in Figure 1.1, the temporal Katz centrality of node u at a given time t is the weighted sum of all time-respecting paths that end in u up to time t.

Our contributions:

- Our centrality metric is online updateable, thus ideal for data-intensive applications. To the best of our knowledge, only two previous studies [52, 131] propose data stream updateable centrality measures.
- Temporal Katz centrality incorporates arbitrary time decay functions that can be adapted to the task in question.
- We conducted a supervised evaluation on our Twitter data collections introduced in Section 2.2. Using only network centrality, we tried to detect daily tennis player accounts as

early as possible. Our measurements on the *RG17* and *UO17* data sets show that our method, the temporal Katz centrality, outperforms both static and online baselines.

- Our further results include two convergence theorems that mathematically justify the connection between our method and Katz index [72].
- Finally, we perform extensive parameter analysis for properties such as score variability between consecutive snapshots as well as adaptation to concept drift.

This chapter is a summary of several papers presented at different stages of our research,

> Ferenc Béres and András A. Benczúr. Online centrality in temporally evolving networks. In *Book of Abstracts of the 6th International Conference on Complex Networks and Their Applications*, pages 184–186, 2017,
>
> Ferenc Béres, Róbert Pálovics, and András A. Benczúr. Temporal walk based centrality metric for graph streams. In *14th International Workshop on Mining and Learning with Graphs, held in conjunction with KDD'18*, 2018,
>
> Ferenc Béres, Róbert Pálovics, Anna Oláh, and András A Benczúr. Temporal walk based centrality metric for graph streams. *Applied Network Science*, 3(32):26, 2018.

Node embeddings in dynamic graphs

In Chapter 4, we investigate methods to encode (*embed*) the nodes of a dynamic network by vectors in a low-dimensional vector space in a way that representations in the embedded space reflect the neighborhood or structural properties of the nodes in the original graph.

Over the last years, a myriad of static node embedding methods have been proposed and applied in node classification and link prediction tasks. In our work, we propose two data stream updateable node embedding methods, StreamWalk and Online Second Order Similarity, with applications similar to static embedding models.

Our contributions in Chapter 4 are the following:

- We describe StreamWalk, an online node embedding algorithm. Similar to temporal Katz centrality [20], StreamWalk is also based on time-respecting random walks.
- We describe Online Second Order Similarity, which directly learns the neighborhood similarity of node pairs in the graph stream by approximating their neighborhood Jaccard

similarity at a given time.

- We conduct supervised node similarity search evaluation on our Twitter data collections introduced in Section 2.2. We show that our models can efficiently differentiate daily tennis player accounts from other network participants. Our measurements on the *RG17* and *UO17* data sets show that our online node embedding models outperform static baselines such as LINE, node2vec or DeepWalk.
- Finally, we show that the combination of StreamWalk and Online Second Order Similarity further improves the accuracy of similarity search.

We presented our initial work at a conference, while our final results were published in a journal:

> Ferenc Béres, Róbert Pálovics, Domokos M. Kelen, Dávid Szabó, and András A. Benczúr. Node embeddings in dynamic graphs. In *Book of Abstracts of the 7th International Conference on Complex Networks and Their Applications*, pages 178–180, 2018,

> Ferenc Béres, Domokos M. Kelen, Róbert Pálovics, and András A Benczúr. Node embeddings in dynamic graphs. *Applied Network Science*, 4(64):25, 2019.

Vaccine skepticism detection by network embedding

In Chapter 5, we deploy node embedding models for vaccine skepticism detection. We analyze social network data related to Covid-19 vaccination. We focus on two groups of people commonly referred to as pro-vaxxers and vax-skeptic users. In short, the first group supports vaccination, while the second questions vaccine efficacy or the need for general vaccination against Covid-19. We intended to develop techniques that can efficiently differentiate content based on the expressed vaccine view.

Our contributions are the following:

- We collect and annotate a large Twitter data set related to Covid-19 vaccination.
- We quantitatively assess the performance of node embeddings for the task of vaccine skepticism detection by deploying them on the reply network that we extracted from the data.
- By training a binary classifier to predict the expressed vaccine view for each tweet, we found that node embedding models can significantly improve performance compared to text-only approaches. Furthermore, they can even reveal pro-vaxxer and vax-skeptic user

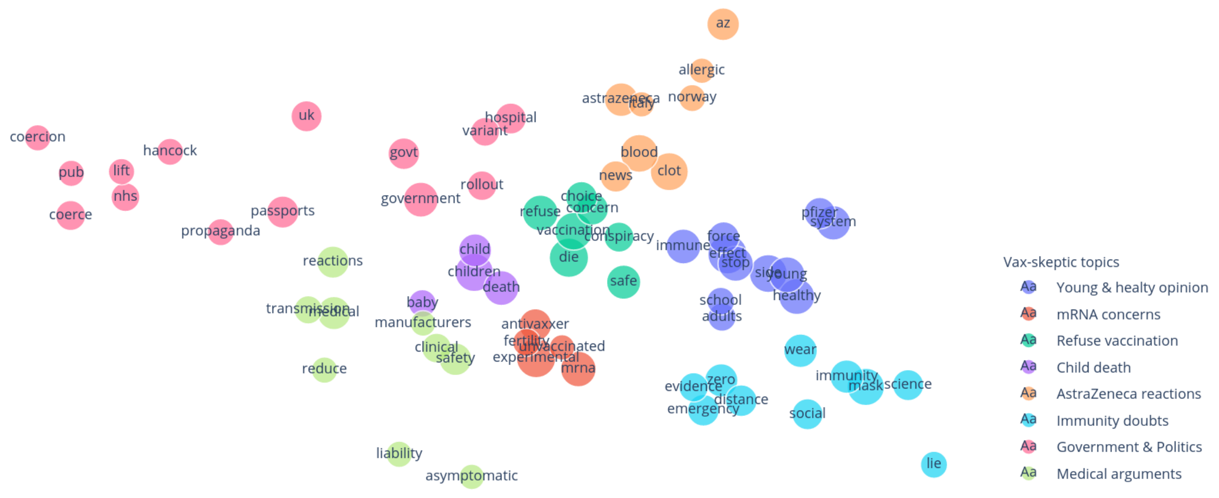


Fig. 1.2: Vax-skeptic topic space uncovered by node embeddings. In the center, there are anti-vaxxer topics (e.g. child death cases, fear from the mRNA technology) that are surrounded by less offensive discussions (e.g. politics, medical arguments, immunity concerns).

clusters as well as their underlying topic hierarchy, see Figure 1.2.

- We released our data and source code on GitHub.

We presented our results at a conference:

Ferenc Béres, Rita Csoma, Tamás Vilmos Michaletzky, and András A. Benczúr. Vaccine skepticism detection by network embedding. In *Book of Abstracts of the 10th International Conference on Complex Networks and Their Applications*, pages 241–243, 2021.

Profiling and Deanonymizing Ethereum Users

Ethereum is the largest public blockchain by usage. It is an account-based cryptocurrency where users store their assets in accounts that they tend to frequently re-use to interact with a wide range of services and decentralized applications (e.g. games, exchanges). As it is a blockchain-based cryptocurrency, the transaction history for each account is publicly observable.

We embed the nodes of the Ethereum transaction graph to profile and deanonymize Ethereum users based on their network activity. The nodes in this graph are Ethereum addresses (accounts) and the transactions are directed links between them. Each physical entity (e.g. users, companies) may own multiple addresses, and the exact address-entity relations are usually hidden from the public. However, in this work, we reveal that node embedding models can efficiently link addresses that belong to the same user.

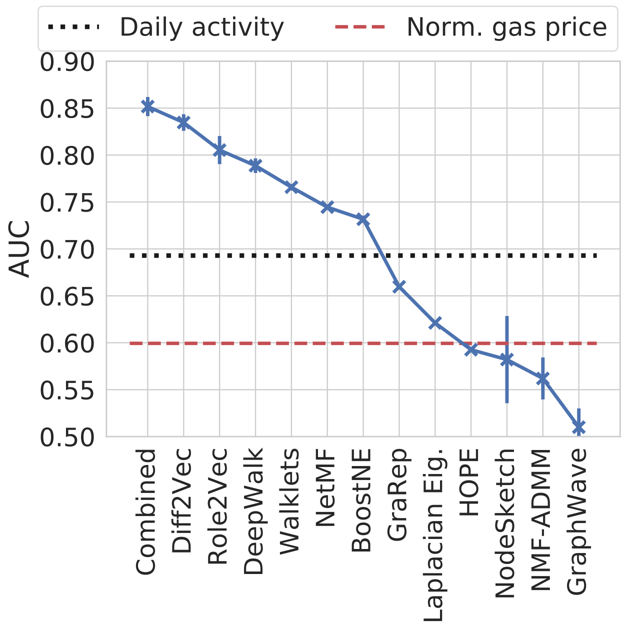


Fig. 1.3: Deanonymization task: find accounts of the same user. AUC is presented for 13 node embedding models as well as time-of-day activity and gas price profile based baselines (horizontal lines).

Our contributions in Chapter 6 are the following:

- We collect Ethereum related data from several sources, including Ethereum name service (ENS), Etherscan blockchain explorer, Tornado Cash mixer contracts, and Twitter.
- Using ENS identifiers as ground truth information, we quantitatively compare the proposed models in a deanonymization task where we link accounts of the same user. As illustrated in Figure 1.3, some node embedding methods significantly outperform user activity based baselines.
- As a direct application, we show that node embedding based profiling can significantly decrease the privacy guarantees of the Tornado Cash (TC) mixer service, which was originally proposed to obfuscate the relationship between addresses of the same user.
- Finally, in light of our results, we propose a few best practices for Ethereum users to follow in order to increase their privacy.

Our results appeared in

Ferenc Béres, István András Seres, András A Benczúr, and Mikerah Quintyne-Collins. Blockchain is watching you: Profiling and deanonymizing ethereum users. In *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 69–78, 2021.

Cryptoeconomic traffic analysis of Bitcoin's Lightning network

In Chapter 7, we analyze the Lightning Network (LN), a payment channel network that was designed to solve Bitcoin's scalability issues. It allows participants to exchange transactions locally, without broadcasting them to the blockchain. Thus, LN opens the way for instant low-value payments with negligible fees.

In a payment channel network, nodes are users and the edges are payment channels. A given node can issue payments only to those participants that it can reach through a series of edges. Intermediary nodes of a given payment path can independently decide the transaction fees that they charge for relaying the payment.

In this work, we designed a payment traffic simulator to analyze the profitability of central router nodes, as original LN payments are cryptographically hidden from us. A main contribution compared to previous simulation-based studies was that we managed to identify more than 100 merchant nodes on LN. By simulating payments from ordinary users towards merchants, we quantitatively confirmed several concerns related to LN that the cryptocurrency community had been speculating about for a long time.

By simulating payments at different value and daily transaction volume levels, we made several observations related to the state of LN in 2019:

- Low routing fees do not sufficiently compensate the routing nodes that essentially hold the network together. Based on our measurements, the annual return of investment (RoI) for every major router is less than 4%. However, they could achieve significantly better RoI, shown in Figure 1.4, by reducing capacity on their currently over-provisioned payment channels.
- We further assess the importance of router entities by monitoring the changes in the number of failed payments after we exclude them one by one from LN.
- Finally, we observe that despite onion routing, routers can gather strong statistical evidence

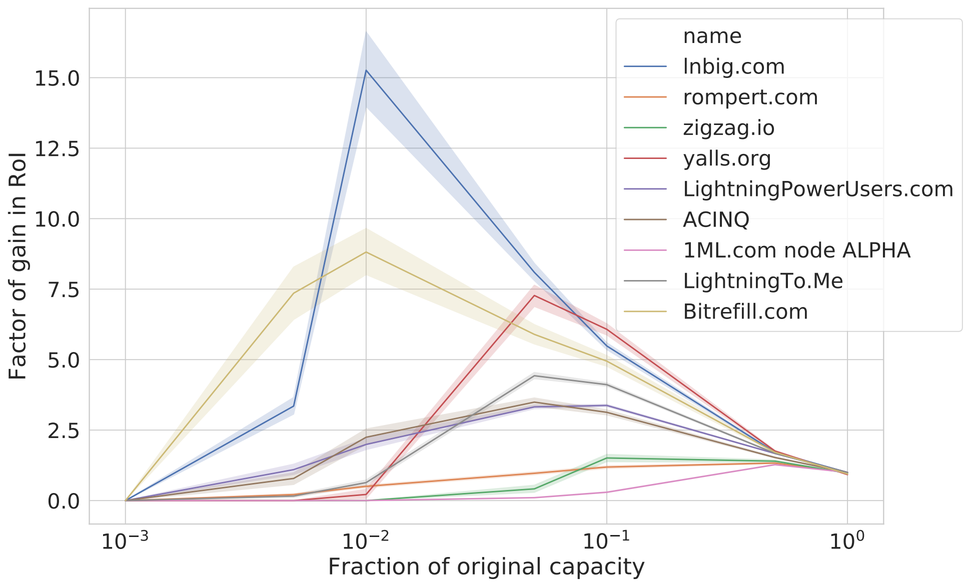


Fig. 1.4: RoI gain after reducing node capacities to the given fractions.

about the sender and receiver of LN payments, since a substantial portion of payments involves only a single routing intermediary. Thus we propose to use longer, suboptimal paths to gain more privacy. Our genetic algorithm based solution only marginally increases the costs for LN users.

Our results were published in

Ferenc Béres, István András Seres, and András A Benczúr. A cryptoeconomic traffic analysis of bitcoin's lightning network. *Cryptoeconomic Systems*, 1(1), 2021.

1.2 Presentation overview

The dependence of the thesis chapters is visualized in Figure 1.5.

In Chapters 2–4, we analyze temporal networks. First, we give the theoretical background in Chapter 2, where we also introduce our Twitter data collections that we use to evaluate dynamic graph algorithms in multiple works. Then in Chapters 3–4, we propose online updateable graph algorithms that perform well in data-intensive applications.

In Chapters 6–7, we turn to the research of cryptocurrency networks. Using different graph mining tools, we quantitatively analyze the Ethereum transaction graph (Chapter 6) as well as the Bitcoin Lightning Network (Chapter 7) to gain valuable insights related to user behavior and

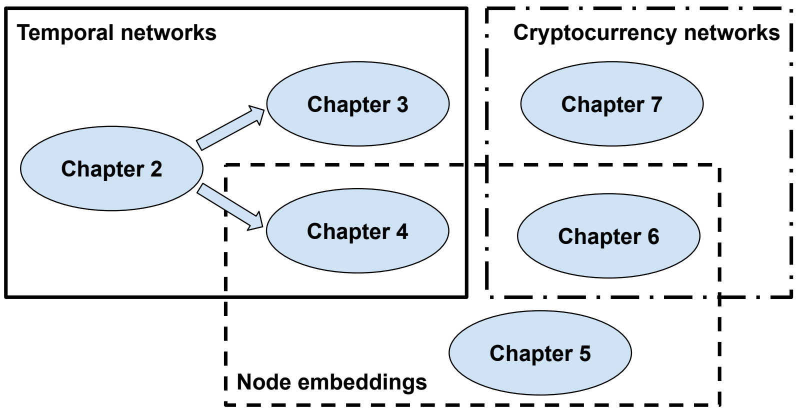


Fig. 1.5: Structure of the presented work.

privacy.

Our research on these domains is connected by the theory of node embeddings that we assess in Chapters 4–6. Our work explores several problems where we successfully applied node embedding algorithms. For example, the applicability of node embedding in temporally evolving networks (Chapter 4), vaccine skepticism detection (Chapter 5), and Ethereum account deanonymization (Chapter 6). For some of these tasks, we were the first to deploy these techniques.

1.3 Credits

The first part of my research is related to temporally evolving networks. We developed new online network centrality and node embedding techniques that outperformed existing snapshot-based approaches. In this work, I collected and annotated dynamic network data, implemented and measured most of the algorithms. Róbert Pálovics and Domokos Miklós Kelen participated in node embedding model implementations [19]. They also verified experimental results and contributed to algorithm descriptions in our articles [19, 20].

My research related to cryptocurrency networks published in [32, 33] is joint work with István András Seres, who contributed with his knowledge on cryptocurrencies, defined the problems, and described the cryptocurrency related background in both papers. The analysis of the basic graph properties of the Bitcoin Lightning Network and their change in time in Section 7.4.1 and [32] is also his contribution. In our works, I designed, implemented, and evaluated the experiments related to traffic simulation and node embedding. Finally, I augmented and collected Bitcoin and

Ethereum related cryptocurrency network data sets that are rigorously assessed in my Thesis.

Chapter 2

Temporal networks

Most of the networks in nature, society, and technology change over time. In graph theory terminology, nodes and edges get additional temporal characteristics and form a *temporal network* [37]. A large variety of temporal network algorithms have appeared for connectivity, spanning trees, matchings, and many more, which are surveyed, for example, in [1, 66]. In Section 2.1, we introduce two dynamic graph computational models that provide frameworks for these algorithms.

Due to the evolving nature of temporal networks, model evaluation is much more challenging than for static graphs. In order to evaluate models on a static graph, static ground truth labeling is required, which itself often requires tedious human effort. In a dynamic graph, depending on time granularity, the same human data curation may be required in each time step. In our best effort to provide quantitative evaluation for dynamic graph algorithms, we collected two Twitter data sets with daily ground truth information related to Open Tennis Tournaments from 2017. We introduce these data sets in Section 2.2.

2.1 Edge streams

The usual approach to analyze temporal networks is to create a series of snapshots, and track dynamics for various parameters in these static graphs [79, 127, 138]. For example, one can collect all retweets on Twitter with corresponding hashtags every day to track the popularity of a political party during the election period and then analyze daily changes in retweet patterns to estimate online and offline popularity of this party [6, 50].

Unfortunately, for high temporal granularity networks, the snapshot-based approach is not always feasible due to several problems. First, if we collect data for the range of hours or days to process as a graph snapshot, we impose an additional delay on the model prediction. On the other hand, running complex graph algorithms for large networks sometimes cannot be executed quickly enough to keep the model predictions up to date. Thus, high temporal granularity networks are usually considered in the *edge* or *graph stream* model [90], where edges must be processed once they arrive in the stream.

Edge streaming for graphs forms a subclass of the *data stream* computational model [9,100]. In this model, data arrives continuously in a potentially infinite stream that has to be processed by a resource-constrained system.

The fact that only a small portion of the data can be kept available for immediate analysis [63] has both algorithmic and statistical consequences for machine learning. Suboptimal decisions on earlier parts of the data may be difficult to unwind, and if needed, require low memory sampling and summarization procedures. Many of the usual data processing operations would need random access to the data [9]. For example, only a subset of SQL queries can be served from the data stream. As surveyed in [100], data stream algorithms can tackle this constraint by a variety of strategies, including adaptive sampling in sliding windows, selecting representative distinct elements, and summarizing data in low-memory data structures, also known as sketches or synopses.

For temporal graphs, there are two data streaming models, the *adjacency stream* model where the graph is presented as a sequence of edges in temporal order and there is no bound on the degree of a vertex, and the *incidence stream* model where graphs are of bounded degree and all edges incident to a vertex are presented successively [15]. In this work, we propose three online updateable graph algorithms over the adjacency stream model that we describe in Chapters 3 and 4.

The need for machine learning over edge streams is motivated by a rapidly growing number of industrial applications of graph algorithms [109,153,154,163] and online machine learning [21, 42,164,166]. However, very few graph learning algorithms are capable of immediately updating their models from edge streams. Similarly, in the literature, we rarely find real graph streaming methods where node labels are highly dynamic: even link prediction tasks are evaluated in batches for sets of edges that appear over a longer period in time.

2.2 Twitter Tennis data sets

In our best effort to provide quantitative evaluation for online updateable graph algorithms on edge streams, we collected two Twitter data sets with daily ground truth information. Our data collections, *RG17* and *UO17*, are related to Roland-Garros 2017, the French Open Tennis Tournament, and for US Open 2017, the United States Open Tennis Championship. The events took place between May 22 and June 11 and August 22 and September 10, respectively.

The two tournaments have similar structure. First, there is a qualification phase that only lasts for a few days. It is followed by one or two days without any tennis games. Finally, the main tournament spans across multiple weeks. It is important to note that some of the most successful professional players usually do not play during the qualifications. Thus, the first part of these sport events does not receive as much social attention as the main tournament phase.

2.2.1 Data collection

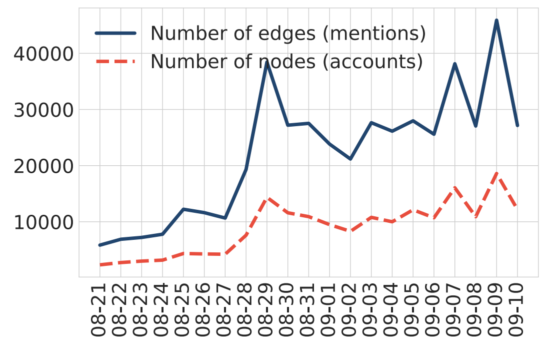
We gathered data with the Twitter Search API by using the following two separate sets of keywords:

{@rolandgarros, #RolandGarros2017,
#rolandgarros2017, #RolandGarros, #rolandgarros,
#FrenchOpen, #frenchopen, #RG17, #rg17}

{#usopen, #Usopen, #UsOpen, #USOPEN,
#usopen17, #UsOpen17, #Usopen2017, @usopen,
#WTA, #wta, #ATP, #atp, @WTA, @ATPWorldTour,
#Tennis, #tennis, #tenis, #Tenis}

The RG17 data covers the events of the championship starting May 24 with $444,328$ tweets and $336,234$ time-stamped mentions. The UO17 data consists of $636,810$ tweets and $482,061$ mentions. Note that we imposed no language restrictions on the text of the tweets during the data collection process.

For the two data sets, we extracted the underlying mention networks where the nodes are Twitter



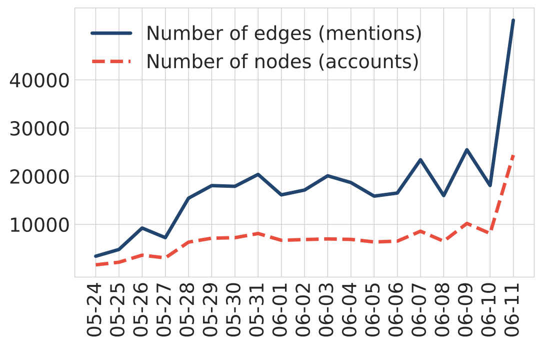


Fig. 2.1: Number of nodes and edges in the **UO17 (left)** and **RG17 (right)** mention graphs.

accounts and the directed edges are @-mentions between them. Both networks are temporal as we assign the timestamp of the containing tweet for each mention link.

As a first investigation, we show the number of edges and nodes for each day in Fig. 2.1. For both data sets, the network dynamics are in strong correlation with the event timeline. During the qualifiers, the number of interactions (mentions) is low. Then user activity increases as the main tournament starts from Aug 28 or May 28, respectively. For UO17, the two bursts on September 7 and 9 are related to Women's Singles semi-final and final. A similar behavior can be observed for RG17 due to Men's Singles finals on June 7, 9, and 11.

Our data sets are special in a way that in addition to the dynamic graph structure, they include ground truth information with daily granularity for the nodes of the mention network. To obtain the ground truth, first we downloaded the official championship schedule. The daily timetables in HTML format contain the following information for each tennis game:

- Full names of the participating players (two for singles and four for doubles games)
- Approximate time of the game during the day (e.g.: after 11:00, not before 15:00, etc.)
- Category and round identifier of the game (e.g. Women's Singles—Round 1, Men's Singles—Final)
- Court name, where the game took place (e.g. Grandstand, Arthur Ashe Stadium, etc.)
- Information about whether the game was canceled, resumed from a previous day, or the final result if completed.

One of the most time-consuming part of the data curation process was to assign Twitter accounts to tennis players. We used player-account pairs to define a relevance score over the nodes of the mention network based on the official event schedule.

The total number of professional participants is 798 for US Open and 698 for Roland-Garros. Since Twitter does not belong to mainstream social media platforms in several countries, for many players, we have not found any Twitter accounts.

We assigned players to accounts by the Twitter Search API's people endpoint; however, the API was sometimes unable to identify the accounts of the players. In these cases, we applied the following semi-automatic procedure:

- For most social network platforms, it is a popular naming convention to choose a similar account name to the full name of the account owner. Thus, using edit distance for each active player, we automatically selected accounts where the *account name* or the displayed *name* is very similar to the full name. During the assignment procedure, we prioritized match in the *account name* attribute as sometimes users tend to post short messages or special characters in their displayed *name* that could easily mislead edit distance.

- In order to match accounts and player names, we first listed the accounts that have minimum edit distance from a given player's name. We removed whitespaces and transformed all characters to lower case. Since name matching can lead to false player-account pairs, we manually searched the lists of different edit distance values to find valid player account matches. We first considered screen names, and in case there was no match, we continued with the displayed name of Twitter accounts.

- Note that the same player often has multiple Twitter accounts, especially the popular players, who usually have official sites and distinct accounts for fans with different nationalities. In this case, we assigned multiple accounts for the same tennis player.

- As the last step, we excluded invalid assignments.

Using the above semi-automatic procedure, we managed to find Twitter accounts for 58.4% of the US Open players, as seen in Fig. 2.2. We achieved better player coverage of 64.2% for Roland-Garros.

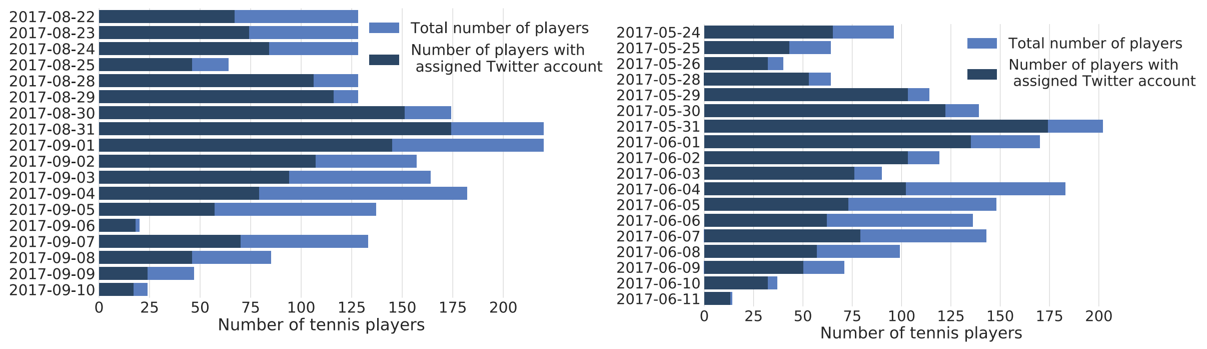


Fig. 2.2: The number of players active on a given day and the number of them with identified Twitter accounts. **Left:** UO17; **Right:** RG17. Days with no tennis game between the qualifiers and the championship (Aug 26-27 and May 27, respectively) are not shown.

2.2.2 Dynamic node relevance label

By relying on the official event schedule and the tennis player to Twitter account assignments, we compiled a binary node relevance label with daily granularity 2.1 for the nodes of the RG17 and UO17 mention networks.

Based on the approximate time of the games, we consider a Twitter account **active** for a given day if he or she participated in a *completed game*, a *canceled game*, or a *resumed game* on the same day, as all of these events are expected to cause a social media burst.

We define relevance as follows. Each node n is a Twitter account in the mention network, which we define *relevant* if it corresponds to a tennis player that participated in the tournaments (e.g. completed, canceled, resumed game) of the current day:

$$\mathrm{rel}(n) := \begin{cases} 1, & n \text{ plays on the current day} \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Our choice of temporal relevance poses challenging task for the dynamic network algorithms, since we assign zero relevance score for various globally popular accounts in the data. For example, accounts related to sport news agencies, commercial products and celebrities other than the players who play in the tournaments are considered irrelevant in terms of the evaluation. We highlight that our temporal relevance label is solely based on an external source, the official event schedule. The fact that it was not constructed from the network structure makes it ideal for the evaluation of online graph algorithms.

In Chapter 3, we use our relevance definition to quantitatively analyze the performance of online and static centrality measures over the UO17 and RG17 Twitter collections. For each centrality measure, we compute the list of the nodes with the highest centrality in *each hour*. For evaluation, we use NDCG [4] with temporal relevance defined by (2.1). First, for a list of length K that contains the top nodes sorted by their centrality metric, we compute the weighted sum of node relevances:

$$\mathrm{DCG@}K = \sum_{i=1}^{K} \frac{\mathrm{rel}(n_i)}{\log_2(i+1)}, \tag{2.2}$$

where n_i is the node at position i in the list. Finally, NDCG is the normalized version of DCG:

$$\mathrm{NDCG@K} = \frac{\mathrm{DCG@K}}{\mathrm{IDCG@K}}, \tag{2.3}$$

where IDCG is the “ideal” DCG we get by ordering the nodes according to their true relevance.

By a different use of the daily tennis tournament schedule, we will investigate the temporal behavior of online and static node embedding algorithms in Chapter 4.

Chapter 3

Temporal walk based centrality metric for graph streams

There is a wide range of commercial and research applications devoted to identifying important, popular, and influential users on social media platforms [43]. Since popularity and importance are social phenomena and judged in a social context, a way to quantify them is through a complex combination of social and behavioral factors. These often include graph characteristics like degree, PageRank, and other centrality metrics [14, 36, 114, 155] measured over the social network. The definitions of centrality can vary greatly and can incorporate both global and local factors of a user's location within the social network [24]. The high variability of centrality scores reflects the nature of popularity observed in real-world [97] and online social networks [10]. Several models have been suggested to explain the emergence of high variability, habitually involving some variation of the preferential attachment mechanism, also extended to the dynamic setting [65].

Three axioms of centrality are defined in [24]. There is a single measure, harmonic centrality, that satisfies all three of them. Since the computation of harmonic centrality for a given node u involves all the distances from the node u in question, the measure is computationally challenging even in a static graph.

In this chapter, we present **temporal Katz centrality**, an online updateable graph centrality metric for tracking and measuring user importance over time.

3.1 Our results

Our proposed metric, temporal Katz centrality, is based on the concept of time-respecting walks containing a sequence of adjacent edges with timestamps ordered in time. As seen in Fig. 3.1, for node u, temporal Katz centrality aggregates each temporal walk ending before time t at u.

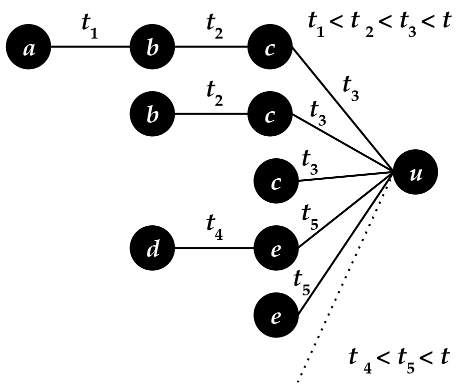


Fig. 3.1: Temporal walks ending at node u before time t.

In our first main result, we extend the definition of the Katz index [72] to the edge stream graph computational model where the edges of the network arrive continuously in time. Although many studies tried to identify the best estimates for the importance of a social media user, to the best of our knowledge, only two previous studies [52,131] propose **data stream updateable** centrality measures.

Another key property of our new centrality measure is the incorporation of arbitrary time decay functions that can be adapted to the task in question. The algorithm of [131], which we analyze in Section 3.2.2, cannot incorporate the actual edge arrival times in its calculations. We believe our method is superior in using the exact time of interaction between two social media users, resulting in better performance in our prediction task.

We also address the difficulty of the timely evaluation of fast changes in social media. In order to evaluate a static centrality measure, static ground truth labeling is required, which itself often requires tedious human effort. In [24], for example, the Text Retrieval Conference (TREC) topics are used [39]. In a dynamic graph, depending on time granularity, the same human data curation may be required in each time step.

For example, in the study most similar to ours [131], only small temporal social network snapshots are collected, and evaluation is mostly based on convergence to static centrality measures.

We conduct supervised evaluation for dynamic centrality in a data-intensive setting over the RG17 and UO17 Twitter data sets introduced in Section 2.2. The underlying dynamic mention graphs have several hundred thousand edges. Both data collections are related to Open Tennis Tournaments from 2017. We used the official event schedule of these sport events to compile ground truth labels for network nodes with daily granularity. Our measurements on these data sets show that our temporal Katz centrality measure outperforms static baselines as well as the temporal PageRank of [131].

Our further results include two convergence theorems that mathematically justify the connection between our dynamic version and the original static Katz index [72].

Finally, we perform extensive parameter analysis to measure score variability and concept drift adaptation in the sequence of graph snapshots.

3.2 Review of related work

For temporal networks, a few generalizations of static centrality measures to dynamic settings have been suggested recently [5, 56, 76, 143, 145]. In these works, tracking centrality of a single node and determining its variability play a major role [145], as it has been observed in the literature that centrality of nodes can change drastically from one time period to another [25].

In this work, we address a practically important variant of dynamic centrality: Our goal is to compute online updateable measures that can be computed from a data stream of time-stamped edges. However, the above results [5, 56, 76, 143, 145] cannot be used for computing and updating centrality online. The following results devise methods that are variants of our snapshot baselines: In [145], the spectrum of a set of discrete graph snapshots is analyzed in time; however, the spectrum cannot be dynamically updated with fine time granularity, as required by our application. Similarly, in [56], sequences of snapshots are considered. Finally, in [5, 76, 143], degree, closeness, and betweenness are considered in dynamic graphs, but these measures, with the exception of the degree, cannot be efficiently updated online. To the best of our knowledge, the only previous such algorithms are temporal PageRank [131] and degree [76]—other measures are inefficient to update online. For example, in [52], a heuristic version of betweenness centrality was proposed for "ego-graphs", which have paths of length two only. They applied their algorithms for small graphs of less than 250 nodes only. Based on the comparative evaluation of centrality measures in [24], we chose not to include experiments with betweenness centrality in our experiments.

The starting point of our temporal Katz centrality measure is PageRank [113], which along with the Katz index satisfies the last two axioms defined in [24]. PageRank is considered a success story in link analysis and listed as one of the ten most influential data mining algorithms [158]. The importance of PageRank in our work has multiple reasons. On the one hand, it is widely used and has favorable properties by the axioms of [24]. On the other hand, temporal PageRank [131] is a modification of PageRank, which to the best of our knowledge, is the only temporal ranking metric proposed in the literature prior to our work.

PageRank, Katz index, and temporal PageRank are all based on counting paths in the underlying networks. Next, we review the general properties of the path counting centrality metrics and temporal PageRank [131]. Then in Section 3.3, we describe our temporal Katz centrality measure.

3.2.1 Path counting centrality metrics

As perhaps the first centrality metric based on path counting, Katz introduced his index [72] as the summation of all paths coming into a node, but with an exponentially decaying weight based on the length of the path:

$$\vec{\mathrm{Katz}} = \mathbf{1} \cdot \sum_{l=0}^{\infty} \beta^l A^l, \tag{3.1}$$

where $\vec{\mathrm{Katz}}$ is the Katz index vector, A is the directed adjacency matrix, and $\beta < 1$ is a constant. Hence the Katz index of a node is the weighted sum of the number of paths of different lengths l terminating in u, where the weight is β^l:

$$\vec{\mathrm{Katz}}(u) := \sum_{v} \sum_{l=0}^{\infty} \beta^l |\{\text{paths of length } l \text{ from } v \text{ to } u\}|, \tag{3.2}$$

The Katz index is finite only if $\beta < 1/|\lambda_1|$, where λ_1 is the eigenvalue of A with largest absolute value [72]. Since $1/|\lambda_1|$ is often very small, around 0.05 in our graphs, the relative weight of a length two path stays very small compared to a single edge. In order to be able to use larger values of β, we introduce the truncated Katz index as

$$\vec{\mathrm{Katz}}^{[k]} = \mathbf{1} \cdot \sum_{l=0}^{k} \beta^l A^l. \tag{3.3}$$

Note that $\vec{\mathrm{Katz}}^{[\infty]} = \vec{\mathrm{Katz}}$.

By the basic definition, PageRank is normally considered to be the static distribution of a random walk with damping [113]. In order to compare PageRank and the Katz index, and to motivate online update rules, we use the result of [49], who show—and use as an efficient algorithm—that PageRank is equal to the path counting formula

$$\vec{\text{PageRank}} = \mathbf{1} \cdot \frac{c}{N} \cdot \sum_{l=0}^{\infty} (1-c)^l M^l, \tag{3.4}$$

where c is the damping constant and M is the random walk transition matrix. In other words, M is the outdegree normalized adjacency matrix: $M = (D^{-1}A)^T$ where D is a diagonal matrix with the outdegrees in the diagonal.

3.2.2 Temporal PageRank

To our knowledge, temporal PageRank [131] is the only published work about temporal generalizations of PageRank that is online updateable. Other results focus on coarse, static snapshots such as Bonacich's centrality [80], or use temporal information to calculate edges of a static graph [67, 87]. Finally, another line of research considers updating PageRank in dynamic or online scenarios [11, 12, 77, 111, 133]; however, in these results PageRank is considered a stationary distribution over the current, static graph.

In [131], temporal PageRank, a dynamic variant of PageRank, is defined as follows. In a dynamic graph, edges are time-stamped and can appear multiple times. The main idea is to aggregate **time respecting temporal walks**

$$z = (u_0, u_1, t_1), (u_1, u_2, t_2), \cdots, (u_{j-1}, u_j, t_j); \qquad t_{i-1} \leq t_i. \tag{3.5}$$

ending in a certain node, as illustrated in Fig. 3.1, to compute its temporal centrality. In such a walk, they model an information flow from the start node u_0 to the destination u_j by passing along edges that arrive subsequently in time.

For each edge (u_{i-1}, u_i, t_i) in walk z, they assign the transition weight as β^s, where $\beta < 1$ is a decay constant and s is the number of edges (u_{i-1}, y, t') that appear after the previous edge but not later than the present edge in the walk, that is, $t_{i-1} < t' < t_i$. They incorporate this weight assignment in formula (3.4); for full details, see [131].

Intuitively, their notion of edge transition weight decays exponentially with the number of possible continuations of the temporal walk at node u_{i-1}. The more edges appear before (u_{i-1}, u_i, t_i), in their model it is exponentially less likely that the information is sent along the given edge—and not another edge that appears earlier.

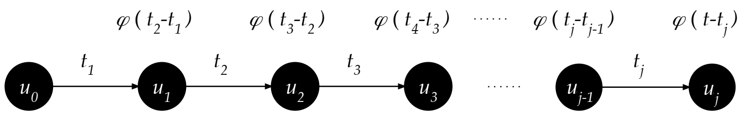


Fig. 3.2: Edge weights along a temporal walk at time t.

The main problem with the above path counting algorithm is that it overvalues nodes with low activity. Consider a node that communicates to ten contacts in a few minutes. The tenth contact will only receive a propagated score proportional to β^{-10}. By contrast, if another node sends only one message per day, the neighbor receives the full score even though the information may already be highly outdated.

One key motivation of the above definition for temporal PageRank is that it possesses a computationally low cost update algorithm. While it is tempting to modify the weight formula to incorporate the actual time elapsed, the stream-based computation of such a modified temporal PageRank becomes unclear.

3.3 Temporal Katz centrality

We define our temporal Katz centrality measure over the stream of edges arriving in time from a dynamic network. Our goal is to specify a metric that is based on the weighted sum of time respecting walks, updateable by the edge stream, and that can incorporate the actual elapsed time in the weights of the walks.

To motivate our new method, we reconsider the temporal PageRank [131] edge transition weight rule: Weight β^s is assigned to an edge uv in a path where s is the number of edges that appear after the previous edge entering u but not later than the appearance of edge uv. The definition involves time decay in an indirect way through a combination with the activity of the nodes. As an advantage, the definition guarantees that the weight will incur the degree normalization required in the PageRank equation (3.4), and hence temporal PageRank will converge to static PageRank if edges are played several times in random order. As a disadvantage, the notion of time is difficult to directly capture in the temporal PageRank algorithm. The more time elapses before the next edge appears, the more other edges have the chance to appear in between. However, this notion also depends on the activity of the node in question, and longer delays are penalized less at inactive nodes compared to active nodes.

We define **temporal Katz centrality** by introducing a natural, purely time-dependent edge transition weight $\varphi(\tau)$, which is an arbitrary function of the time elapsed since the previous edge in a path. Intuitively, we define a time dependent decay for each edge, as shown in Fig. 3.2. We will use the edge decay values to compute an aggregated freshness of the information flow along a given path, which we will in turn aggregate for the final nodes of the paths.

1. temporal Katz centrality is the weighted sum of all time respecting walks that end in node u,
$$r_u(t) := \sum_{v} \sum_{\substack{\text{temporal paths } z\\ \text{from } v \text{ to } u}} \Phi(z,t) \tag{3.6}$$
where $\Phi(z,t)$ is the weight of walk z at time t. Truncated temporal Katz centrality is defined similar to equation (3.3) by restricting to walks of length at most k.

2. For a temporal walk as in equation (3.5) where edges appeared at $(t_1, t_2, ..., t_j)$, we define weight $\Phi(z,t)$ as
$$\Phi(z,t) := \prod_{i=1}^{j} \varphi(t_{i+1} - t_i). \tag{3.7}$$
where φ is a time-aware weighting function, and for $i = j$ we let $t_{j+1} := t$.

3. Hence $\Phi(z,t)$ is the product of individual edge transition weights $\varphi(t_{i+1} - t_i)$ as seen in Fig. 3.2. The last term of the product $\varphi(t - t_j)$ captures the delay between present time t and the appearance of the last edge in the path.

By combining Equations (3.6)–(3.7) temporal Katz centrality can be considered a variant of the Katz index Equation (3.2), in which time respecting paths are weighted by $\Phi(z,t)$:
$$r_u(t) := \sum_{v} \sum_{\substack{\text{temporal paths } z\\ \text{from } v \text{ to } u}} \prod_{i=1}^{j} \varphi(t_{i+1} - t_i). \tag{3.8}$$

By using different edge weight functions, we cover two important special cases for temporal Katz centrality:

- If $\varphi(\tau) := \beta$ is constant, we obtain a variant of the Katz equation (3.2) with summation for temporal paths instead of all paths irrespective of time.

- In another special case, $\varphi(\tau) := \beta \cdot \exp(-c\tau)$. Since φ is an exponential function, $\varphi(a) \cdot \varphi(b) = \varphi(a+b)$. Hence the path weight in (3.7) becomes

$$\Phi(z,t) = \beta \exp(-c[t - t_j]) ... \beta \exp(-c[t_2 - t_1]) = \beta^{|z|} \exp(-c[t - t_1]), \tag{3.9}$$

that is, it involves a Katz-style decay proportional to the length of the path, combined with an exponential decay depending on the time elapsed since the first interaction t_1 over the path occurred. This weight is capable of capturing the temporal decay of information spreading and propagation.

3.3.1 Update formula

In this section, we show how we can maintain temporal Katz centrality r_u for each node u, which is the sum of temporal paths z as in equation (3.5) with weight $\Phi(z, t)$ as in (3.7). We base our analysis below on the fact that the sum of all temporal paths to u can be derived by using the number of temporal paths ending at the in-edges of u. As seen in Fig. 3.3, if edge vu appears at time t_{vu}, the future centrality of node u at time t increases as

1. a new time respecting walk appears that starts from v and has weight $\varphi(t - t_{vu})$,
2. for each time respecting walk that ended in v at t_{vu}, a new walk with the new edge vu appears. The total weight of paths that ended in v is $r_v(t_{vu})$, hence the weight of the new walks is $r_v(t_{vu}) \cdot \varphi(t - t_{vu})$.

Adding up the weight of the two types of new walks, we get

$$r_u(t) = \sum_{vu \in E(t)} (1 + r_v(t_{vu}))\, \varphi(t - t_{vu}), \tag{3.10}$$

where $E(t)$ is the multi-set of edges appearing no later than t. Based on the above recursive formula, if edge vu appears at time t_{vu}, it increases the future centrality of node u by $(1 + r_v(t_{vu}))\, \varphi(t - t_{vu})$. The increase of the centrality of u can be computed by maintaining the values t_{vu} and $w_{vu} := 1 + r_v(t_{vu})$. The algorithm for updating temporal Katz centrality is hence the following:

- For each node u, we initialize temporal Katz centrality r_u as constant 0. For each edge vu, we maintain the edge weight w_{vu} and the time of appearance t_{vu}, initially all set to 0 and $-\infty$, respectively. We let $E(t)$ denote the multi-set of edges that appeared before time t.
- Next, we consume the stream of edges vu and we update r and w as follows. First we calculate the current value of r_v as

$$r_v := \sum_{zv \in E(t)} w_{zv} \cdot \varphi(t - t_{zv}). \tag{3.11}$$

Here $E(t)$ is a multi-set, and each past occurrence of edge zv is counted separately, with different t_{zv} and hence different decay. Note that when edge vu appears, $t = t_{vu}$.

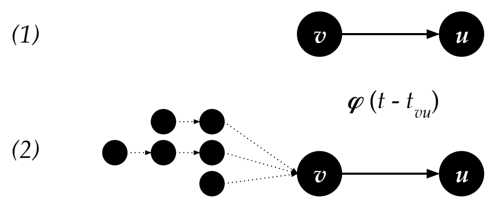


Fig. 3.3: At time t when edge vu becomes active, (1) a new walk appears starting from v, and (2) each time respecting walk that ended in v continues to u.

- Then we add a new edge vu to the multi-set of edges with $w_{vu} := r_v + 1$ to propagate the centrality score along edge vu, and set $t_{vu} := t$.
- The above algorithm can also be applied to update truncated temporal Katz centrality by the following modification: We maintain an array $w_{vu}^{[l]}$ for $l = 1, \ldots, k$ for each edge in the multi-set $E(t)$, and set

$$\begin{aligned} w_{vu}^{[1]} &:= 1 \\ w_{vu}^{[l]} &:= 1 + \sum_{zv \in E(t)} w_{zv}^{[l-1]} \cdot \varphi(t - t_{zv}) \quad \text{for } 1 < l \le k. \qquad (3.12) \\ r_v^{[l]} &:= \sum_{zv \in E(t)} w_{zv}^{[l]} \cdot \varphi(t - t_{zv}) \qquad (3.13) \end{aligned}$$

Time ordering is consistent with information propagation: For a path of three nodes u, v, and z, we can propagate a certain share of the r_u score along edge vz only by first propagating along uv; hence uv must appear before vz.

To relate temporal Katz centrality to (online) PageRank, notice the difference of the Katz and PageRank path counting formulas (3.1) and (3.4). In Katz, the exponential decay is applied to powers of the binary valued adjacency matrix A, while in PageRank, to the degree normalized random walk matrix M.

Observe the lazy behavior of the algorithm: Ranks are updated only for the tail v of each new edge vu. We assign based on the centrality of v $r_v + 1$, as the weight w_{vu}. If we query the rank of u, we propagate r_v along edges vu; however, we add a time decay to account for the freshness of the edges vu: More recent edges propagate scores with higher intensity.

3.3.2 Time complexity

The time complexity of maintaining r_v by formula (3.11) is linear in the degree of v. We can further improve the online update complexity to constant time per update if φ satisfies

$\varphi(a+b) = \varphi(a) \cdot \varphi(b)$. In this case, it is easy to see that at query time t, we can recompute r_v by the actual time t in formula (3.11) as

$$r_v := r_v \cdot \varphi(t - t_v) \tag{3.14}$$

where t_v is the last time node v was updated.

We can combine formulas (3.11), (3.10) and (3.14) to update r_u for each new edge (vu) by

$$\begin{aligned} r_v &:= r_v \cdot \varphi(t - t_v); \\ r_u &:= r_u \cdot \varphi(t - t_u) + (r_v + 1) \cdot \beta; \\ t_u &:= t, \;\; t_v := t \end{aligned} \tag{3.15}$$

Querying the centrality score of a single node can be served in constant time by formula (3.14). Hence computing a centrality top list can be done in time linear in the number of vertices. For the special case when $\varphi(t) = 1$, the scores change only when formula 3.15 is applied, hence the scores can be stored, for example, in a heap to quickly access the maximum score. In other cases, we can deploy heuristics such as [146] to quickly find u that maximizes the product (3.14); however, such an optimization is out of scope in this work.

Overall, for the decay functions φ used in our experiments, the time complexity of our method is identical to that of time decayed degree. In the special case of $\varphi = 1$, our time complexity is equal to that of static degree, while for other decay functions, we can bring the running time very close to static degree by applying heuristics to find the maximum of a product [146].

We experimentally compared the running time of our method with static indegree, static PageRank, temporal PageRank, and harmonic centrality in Fig. 3.4. We generated random Barabási–Albert graphs [16] by the `barabasi_albert_graph` method of the `networkx` Python package[1] and constructed temporal graphs by using a 10% sample of the edges in random order. We split the temporal graph into ten equal sized slices and computed all node centrality values at the end of each of the ten slices. The size of the graphs are found in Table 3.1.

As seen in Fig. 3.4, except for harmonic centrality, all algorithms scale linear with the number of edges. For our temporal Katz centrality algorithm, more than half of the running time is

[1]https://networkx.github.io/documentation/networkx-1.9.1/reference/generated/networkx.generators.random_graphs.barabasi_albert_graph.html. We set $m = 3$ for the number of edges from a new node to existing nodes.

Table 3.1: The size of the random Barabási–Albert graphs generated for the scalability experiments.

Nodes	**Edges**	**Edge sample size**
10 000	59 982	5 998
50 000	299 982	29 998
100 000	599 982	59 998
1 000 000	5 999 982	599 998
2 000 000	11 999 982	1 199 998
3 000 000	17 999 982	1 799 998
4 000 000	23 999 982	2 399 998
5 000 000	29 999 982	2 999 998

consumed by multiplying the centrality values by the time decay as in Equation (3.14) at the time of reading the observations. Hence we also report the running times of our method without time decay synchronization at the end of the time frames. Overall, we observed that the running time of these methods show implementational rather than algorithmic differences.

3.3.3 Normalization for numeric stability

Next we describe how to normalize the temporal Katz centrality scores throughout the computations for numeric stability. The main reason is that in our experiments, the values often resulted in numeric overflow for the best performing values of β. Since for a ranking method, the actual values of the score are indifferent, and only the rank order matters, we can apply any method to normalize temporal Katz centrality. The main challenge is that the normalization method must also be online updateable.

First, we discuss the numerical importance of normalizing temporal Katz centrality. Katz index (3.1) converges only if β is less than the inverse of the largest eigenvalue of A [72]. Typical maximal values of β for real graphs are in the range of 0.01–0.05, which gives small weight for longer paths. By contrast, temporal Katz centrality performed best in our experiments for detecting important nodes of the network for much larger values β. For the high values of β, the centrality scores quickly grow to infinity, as it happened in our experiments. For this reason, next we propose a method for normalizing temporal Katz centrality.

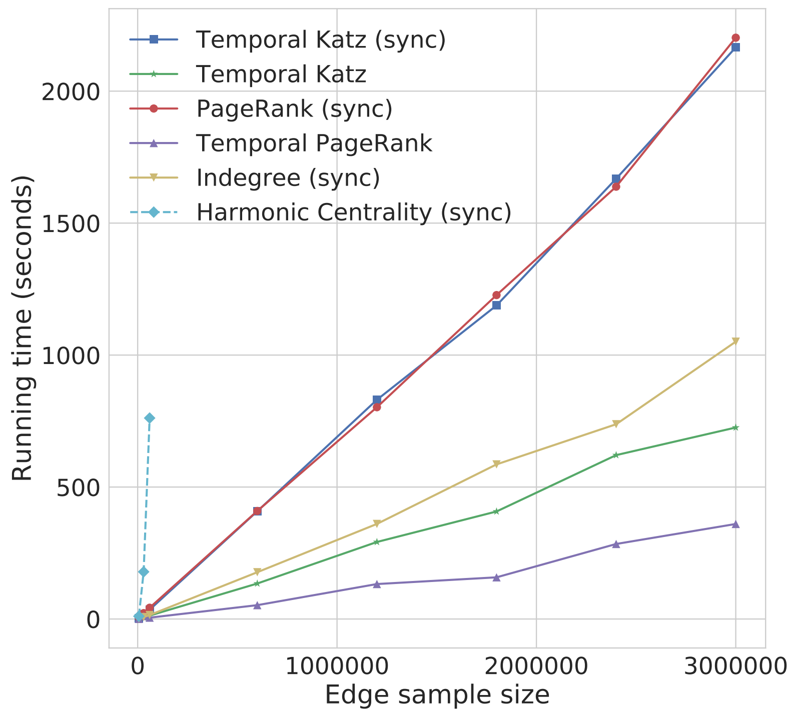


Fig. 3.4: The running time of temporal PageRank, static PageRank, static indegree, harmonic centrality, and temporal Katz centrality with and without synchronizing with time decay at the end of each time frame, as in Equation (3.14), measured over random Barabási–Albert graphs with sizes as in Table 3.1. All static centrality measures are considered to be synchronized.

To normalize the centrality scores, it is sufficient to maintain the sum of the raw scores. Given the sum, we can always divide raw scores by the sum to obtain the normalized values. In order to ensure that the raw values and the sum do not grow unbounded, we have to periodically apply the normalization to all values. Unfortunately, synchronized normalization of all values is not possible in the data streaming model. Instead, we apply lazy normalization and maintain the time-stamped history of the multipliers. Whenever we touch a centrality value, we first check its time stamp to see if pending normalization steps need to be taken first before using the value.

Finally, we describe the algorithm to maintain the sum of the centrality scores. Instead of the lazy algorithm in Section 3.3.1, which updates centrality r_v only when a new edge vu appears that will later propagate the value of r_v to node u, we theoretically maintain the actual score at every time instance. First, for every clock tick of time τ, we multiply each r_v, and hence also the sum, by $e^{-\tau}$ as in equation (3.14). Second, we consider an event when edge vu appears. At this time, the value of r_v is computed by the update equation (3.11). This new edge propagates the score r_v to u and thus increases r_u by r_v. Hence for all new edges, the increase of the sum at the time edge vu appears is r_v measured at that time. To maintain the total sum of the centrality scores, all is required is to add up r_v in equation (3.14) whenever it is applied by the update algorithm, and multiply by $e^{-\Delta t}$ at every clock tick of time Δt.

3.3.4 Convergence properties

Let us assume that we sample a sequence of T edges from a graph with edge set of size E. We intend to compute the expected value of temporal Katz centrality over the sampled edge stream, under the assumption that the activation of the links of the underlying graph is random. We give estimates on the number of times a given path is expected to appear in time respective order, which yields in convergence theorems for temporal Katz centrality to an expression similar to the Katz index. Note that we assume that sampling is done in a uniform way over time, hence in what follows, time t corresponds to the number of sampled edges in the process.

Theorem 3.3.1. Let us compute (truncated or normal) temporal Katz centrality with $\Phi(z,t) = \beta^{|z|}$ (no decay). If we sample a sequence of T edges from an edge set of size E, the expected value of temporal Katz centrality is

$$\vec{\text{TemporalKatz}} = \mathbf{1} \cdot \sum_{l=0}^{k} \beta^l A^l \binom{T}{l} \cdot E^{-l} \simeq \mathbf{1} \cdot \sum_{l=0}^{k} \beta^l A^l (T/E)^l / l!. \tag{3.16}$$

Proof. The expected number of times the edges of a given path of length l appear in a given order, in an edge sample of size T can be computed as

$$s_{T,l} = \binom{T}{l} \cdot E^{-l}, \tag{3.17}$$

since a given edge has a probability of $1/E$ to appear at a given position in the sequence of T edges. To complete the proof, observe that by equation (3.8), temporal Katz centrality is

$$\vec{\text{TemporalKatz}} = \mathbf{1} \cdot \sum_{l=0}^{k} \beta^l A^l \cdot s_{T,l} = \mathbf{1} \cdot \sum_{l=0}^{k} \beta^l A^l \binom{T}{l} \cdot E^{-l} \tag{3.18}$$

□

Theorem 3.3.2. Let us sample a sequence of T edges from an edge set of size E. Let us compute (truncated or normal) temporal Katz centrality with exponential weighting, $\varphi(\tau) := \beta \exp(-c\tau)$.

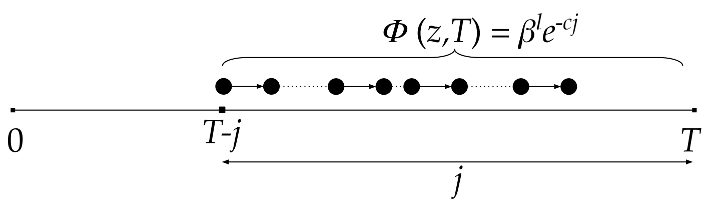


Fig. 3.5: Explanation of Theorem 3.3.2. Each occurrence of a given path of length l that starts at time $T-j$ has the same weight $\beta^l \exp -cj$.

Then as $T \mapsto \infty$, the limit of the expected value of temporal Katz centrality is

$$\text{Tempo}\vec{\text{r}}\text{alKatz} = \mathbf{1} \cdot \sum_{l=0}^{k} A^l \left(\frac{\beta}{E}\right)^l \left(\frac{1}{e^c - 1}\right)^l . \tag{3.19}$$

In particular, if $c = c'/E$ with $c' \ll E$, then the expected value of temporal Katz centrality is approximately

$$\text{Tempo}\vec{\text{r}}\text{alKatz} = \mathbf{1} \cdot \sum_{l=0}^{k} A^l \left(\frac{\beta}{c'}\right)^l . \tag{3.20}$$

Proof. We intend to compute

$$\text{Tempo}\vec{\text{r}}\text{alKatz} = \lim_{T\to\infty} \mathbf{1} \cdot \sum_{l=0}^{k} A^l s_{T,l} = \mathbf{1} \cdot \sum_{l=0}^{k} A^l \lim_{T\to\infty} s_{T,l} \tag{3.21}$$

where $s_{T,l}$ denotes the expected total weight of a given path of length l in an edge sample of size T.

Let us consider a given path of length l starting at time $t_1 = T - j$ as seen in Fig. 3.5. Each possible occurrence of the path starting at the same time $t_1 = T - j$ has the same weight $\Phi(z, T) = \beta^l e^{-cj}$ (see (3.7) and (3.9)). Since we fix the first edge of these occurrences, by equation (3.17), the expected number of the occurrences is $\frac{1}{E^l}\binom{j-1}{l-1}$. As a result, the expected total weight of a given path of length l is

$$s_{T,l} = \beta^l \frac{1}{E^l} \sum_{j=l}^{T} \binom{j-1}{l-1} e^{-cj}. \tag{3.22}$$

Since $\sum_{n=m}^{\infty} \binom{n}{m} x^n = x^m/(1-x)^{m+1}$,

$$\begin{aligned}
\lim_{T\to\infty} s_{T,l} &= \lim_{T\to\infty} \left(\frac{\beta}{E}\right)^l \sum_{j=l}^{T} \binom{j-1}{l-1} e^{-cj} \\
&= \left(\frac{\beta}{E}\right)^l e^{-c} \sum_{j=l}^{\infty} \binom{j-1}{l-1} e^{-c(j-1)} \qquad (3.23) \\
&= \left(\frac{\beta}{E}\right)^l \frac{e^{-cl}}{(1-e^{-c})^l} = \left(\frac{\beta}{E}\right)^l \frac{1}{(e^c-1)^l}. \qquad (3.24)
\end{aligned}$$

Hence

$$\text{Tempo}\vec{\text{ral}}\text{Katz} = \mathbf{1} \cdot \sum_{l=0}^{k} A^l \lim_{T\to\infty} s_{T,l} = \mathbf{1} \cdot \sum_{l=0}^{k} A^l \left(\frac{\beta}{E}\right)^l \left(\frac{1}{e^c-1}\right)^l. \qquad (3.25)$$

If $c = c'/E$ with $c' \ll E$, then $c'/E << 1$ and $e^{c'/E} \approx 1 + c'/E$; hence

$$\text{Tempo}\vec{\text{ral}}\text{Katz} = \mathbf{1} \cdot \sum_{l=0}^{k} A^l \left(\frac{\beta}{E}\right)^l \left(\frac{1}{1+c'/E-1}\right)^l = \mathbf{1} \cdot \sum_{l=0}^{k} A^l \left(\frac{\beta}{c'}\right)^l. \qquad (3.26)$$

□

There is always a certain amount of fluctuation in temporal centrality as the effect of the most recently selected edges. We can compute the expected increase for the weight of paths that end with the most recently selected edge.

For the case with no decay, the additional count is the number of times the length $l-1$ prefix appears, which is $s_{T-1,l-1}$. The increase is approximately a multiplicative $(1+l/E)$ factor, which may be large for a large l; however, the weight of long paths is diminishing exponentially as β^l.

For the case with decay, the increase is given by equation (3.24) applied with $l-1$ instead of l, which approximately gives an expected multiplicative increase $(1 + 1/(Ee^{-c}))$, which is approximately $1 + c'$ for the special case of Theorem 3.3.2.

3.4 Unsupervised evaluation

In addition to the tennis tournament data sets with ground truth relevance labels as defined in Section 2.2, we used the data sets of [131] for unsupervised analysis (see Table 3.2). These small temporal networks (Students, Facebook, Enron, Tumblr) have no more than 10,000 edges[2], as seen in Table 3.2.

[2]GitHub repository of the temporal PageRank research:
https://github.com/polinapolina/temporal-pagerank

Table 3.2: Summary of the data sets used.

	Edges	**Nodes**	**Days**
Students	10,000	1,654	121
Facebook	10,000	4,752	104
Enron	6,251	1,944	892
Tumblr	7,645	1,757	89
UO17	482,061	106,920	21
RG17	336,234	78,095	19

3.4.1 Stability vs. changeability

We assess the amount of variability of temporal Katz centrality in time, depending on the parameters β and the time decay exponent to exhibit the speed of focus shift in daily interactions. We use the weight function $\varphi(\tau) = \beta \cdot 2^{-c\tau}$; c can be considered as the half-life of the information sent over an edge. We update temporal Katz centrality without truncation after each edge arrival, and compute the top 100 nodes with highest centrality scores for each snapshot. We generate the lists at the beginning of each day for the small data sets of [131], and each hour for our Twitter collections RG17 and UO17. Spearman correlation is calculated between lists of adjacent snapshots, for different values of c and β, as shown in Fig. 3.6.

Our measurements show that the similarity between adjacent lists depends on two different factors. We can turn temporal Katz centrality more static by using longer half-life in the decay. If the half-life is short, we even get negative correlations as the number of nodes present in both lists decreases. Another option is to use larger β. By increasing β, the contribution of long walks will be more relevant, which cannot be dominated by recently added edges as easily as for a small β. The two approaches can also be used in combination. We observed the highest similarity using $\beta = 1.0$ with large half-life values.

3.4.2 Adaptation to concept drift

Rozenshtein et al. [131] showed that temporal PageRank can adapt to the changes in the edge sampling distribution over semi-temporal networks. We conducted similar measurement for temporal Katz centrality on the same data sets: We created concept drift by changing the sampling distribution that generates the temporal graphs and measuring how quickly the different

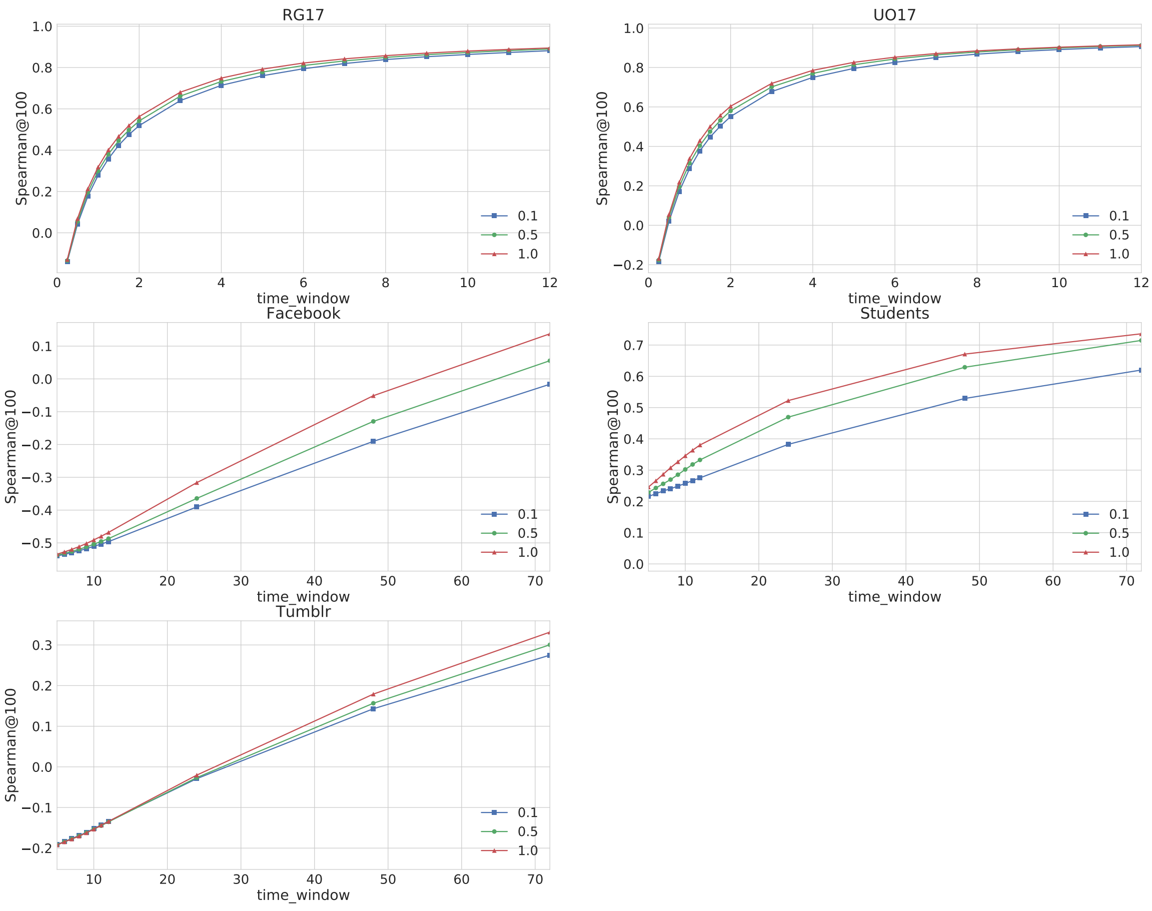


Fig. 3.6: Average Spearman correlation between temporal Katz centrality scores of adjacent snapshots. Daily snapshots are used for Facebook, Students and Tumblr data sets, and hourly snapshots are used for RG17 and UO17 Twitter collections. The correlation is presented for β values 0.1,0.5,1.0 and several time decay intensity in the horizontal axis.

methods get closer to the static centrality measure of the new distribution.

We created concept drift by changing the sampling distribution that generates the edge stream. We measured how quickly different temporal centrality measures converge to the static centrality measure of the new distribution.

In our experiment for concept drift adaptation, we randomly selected 500 nodes as a base graph and formed three overlapping subsamples of 400 nodes each. Similar to the approach in [131], we formed a temporal edge stream of three segments corresponding to the three subsamples, in each segment selecting 10,000 random edges from the corresponding subsample. We compute temporal PageRank and temporal Katz centrality without truncation by assuming that a single new edge in the stream appears in each time unit. In other words, we measure the elapsed time τ by the number of edges in the stream.

We computed weighted Kendall tau [150] rank distance between temporal Katz centrality and static Katz index restricted to the nodes of the actual subsample. This results in concept drift with three different versions of the static centrality score corresponding to the three time periods. By using weighted Kendall tau for measuring concept drift adaptation, we put more emphasis on nodes with high centrality compared to (unweighted) Kendall tau. For the same reason, we use the asymmetric version as in [150] by using the weight of 1/rank for the static Katz index and zero for the online methods. By this choice, Kendall tau measures the distance from the Katz index acting as ground truth.

In Fig. 3.7, we evaluated our model for various values of the exponential decay against the Katz index with $\beta = 0.01$. The results show that in case of weak decay $c = \frac{1}{|E|}$, temporal Katz centrality becomes similar to static Katz index as the graphs evolve, which is in accordance to Theorem 3.3.2 stating that temporal Katz centrality converges to an expression similar to the static Katz index. On the contrary, strong decay shifts the focus of temporal centrality towards the recently sampled edges, thus correlation decrease for $c = \frac{10}{|E|}$ and $c = \frac{100}{|E|}$. Also note the noise in temporal Katz centrality rank distance curves due to the effect of the most recently selected edges, as described in Section 3.3.4.

To summarize our experiments in Fig. 3.7, we considered the behavior of temporal Katz centrality with different parameters as well as temporal PageRank after the two changes in sampling distribution marked by vertical bars in the Figure. We observed that temporal PageRank forgets the old distribution very slow, while temporal Katz centrality very quickly becomes similar to

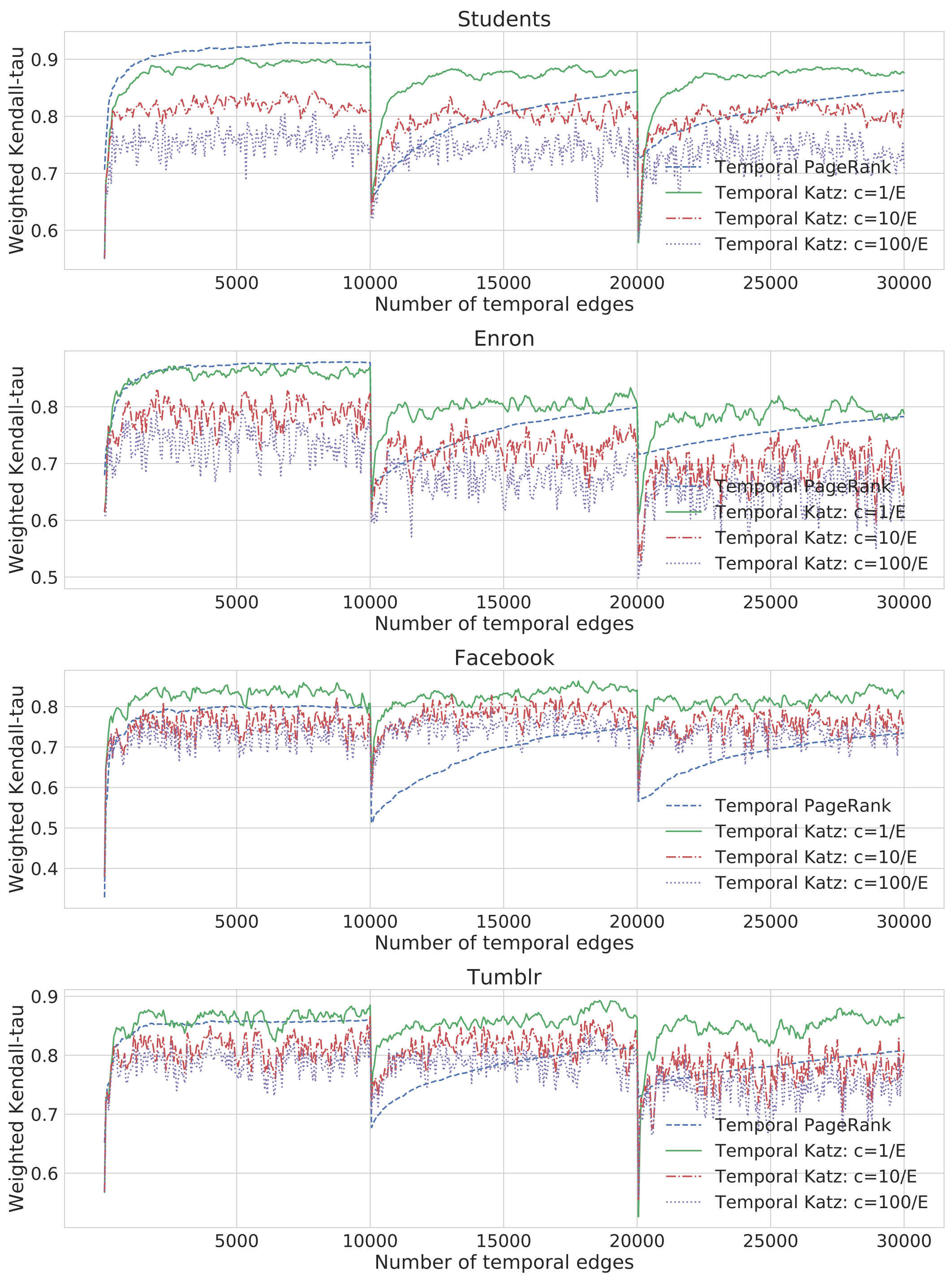


Fig. 3.7: Weighted Kendall tau rank distance of static Katz index and online methods by sampling to simulate concept drift over Students, Enron, Facebook and Tumblr data. Static Katz index has $\beta = 0.01$. The Weighted Kendall tau curves for temporal Katz centrality with $c = \frac{1}{|E|}$ are green, with $c = \frac{10}{|E|}$ are red, with $c = \frac{100}{|E|}$ are purple, and for temporal PageRank are dashed blue. Noise in temporal Katz centrality is due to the effect of the most recently selected edges. The two vertical bars mark the time of the concept drift, when a new sampling distribution is used to generate the temporal edges.

the new static distribution. The best parameter for temporal Katz centrality is a weak decay $c = \frac{1}{|E|}$, which is still sufficient to forget the old distribution but gives less fluctuation compared to the very highly adaptive, stronger decay versions with larger values of c.

3.5 Supervised evaluation

In this section, we quantitatively analyze the relevance of temporal centrality measures over the UO17 and RG17 Twitter collections. We compare the relevance of temporal Katz centrality to temporal PageRank and other *online* and *static* baseline methods described in Section 3.5.1.

To evaluate online metrics, we perform continuous update as the new edges arrive, by considering our mention graph data as a time-ordered edge stream. For the static metrics, we consider different graph snapshots. For each centrality measure, we compute the list of the nodes with the highest centrality in *each hour*. We use NDCG [4] for evaluation, defined as follows. For a list of length K that contains the top nodes sorted by their centrality metric, we compute the weighted sum of node relevances:

$$\mathrm{DCG@}K = \sum_{i=1}^{K} \frac{\mathrm{rel}(n_i)}{\log_2(i+1)}, \tag{3.27}$$

where n_i is the node at position i in the list and $rel(n_i)$ is its relevance: An account n_i is relevant (see Section 2.2.2 for the justification) if it corresponds to a tennis player that participated in the tournaments of the current day:

$$\mathrm{rel}(n_i) := \begin{cases} 1, & n_i \text{ plays on the current day} \\ 0, & \text{otherwise.} \end{cases} \tag{3.28}$$

Finally, NDCG is the normalized version of DCG:

$$\mathrm{NDCG@K} = \frac{\mathrm{DCG@K}}{\mathrm{IDCG@K}}, \tag{3.29}$$

where IDCG is the “ideal” DCG we get by ordering the nodes according to their true relevance.

3.5.1 Baseline metrics

We compare temporal Katz centrality to *online* (or time-aware) and *static* (or batch) metrics. Online metrics are updated after the arrival of each edge. By contrast, static metrics are only updated once in each hour. At hour t a static metric is computed on the graph constructed from edges arriving in time window $[t-T, t]$ from the edge stream. For each baseline, we experimentally select the best value of T.

We consider four *static* centrality measures as baseline:

- *PageRank* [113]: We set $\alpha = 0.85$, and 50 iterations.
- *indegree*: We calculate the indegree of each node in time window $[t-T, t]$ by counting each edge once, that is, without multiplicity.
- *negative β-measure* [24]: The normalized version of indegree, for node u
$$\sum_{z \in N_{in}(u)} \frac{1}{\text{outdegree}(z)} \tag{3.30}$$
where $N_{in}(u)$ denotes the in-neighbors of u.
- *harmonic centrality* [24]: For node u
$$\sum_{z \neq u} \frac{1}{d(z,u)}. \tag{3.31}$$

Furthermore, we compare temporal Katz centrality with two *online* metrics, temporal PageRank [131] and decayed indegree.

- *temporal PageRank:* We set $\alpha = 0.85$ and $\beta \in \{0.001, 0.01, 0.05, 0.1, 0.5, 0.9\}$ for transition weight.
- *decayed indegree:* Using the notations of Section 3.3.1, the decayed indegree of node u at time t is
$$\sum_{zu \in E(t)} \varphi(t - t_{zu}) \tag{3.32}$$
where φ is the time decay function that we set $\varphi(t - t_{zu}) := \exp(-c(t - t_{zu}))$ similarly to temporal Katz centrality.

3.5.2 Results

As the final and main analysis of the relevance of centrality measures, we compute hourly lists of top centrality nodes and calculate the NDCG@50 against the ground truth. We show two different ways to aggregate hourly NDCG@50 values:

1. For each hour of the day between 1:00 and 24:00, we show averages over the days of the tournament.

2. As a single global value, we average NDCG@50 for all days with all hours between 10:00 and 20:00.

The hour of the day has a key effect on performance. In the early hours, activity is low, and hence information is scarce to identify the players of the coming day. By contrast, in the late hours after the games are over, we expect that all models easily detect the players of the day based on the tweets of the results. The effect of the hour of the day can be seen in Fig. 3.8, where we plot the average daily performance for temporal Katz centrality measured over the UO17 data. This observation, along with the fact that daily tennis games start around 10:00 is the motivation to average NDCG@50 scores only between 10:00 and 20:00.

First, we analyze our baseline models. Each static metric is computed at hour t over the graph defined by edges arriving in time frame $[t - T, t]$. Hence the key parameter of these methods is the length of the time window T. Similarly, online decayed indegree depends on the half-life parameter $\tau := \ln 2/c$. Fig. 3.9 shows the overall performance of the static baselines as the function of time frame T, and the quality of decayed indegree as the function of half-life τ. For both data sets, PageRank and harmonic centrality outperform degree-related methods. Furthermore, these path-based methods prefer larger time frames, while degree-based models perform best at smaller values of T.

Next we turn to analyzing temporal Katz centrality with exponential decay. The key parameters of our method are the parameters of the exponential decay β and $\tau := \ln 2/c$, and truncation k. We then parameterize exponential decay with half-life $\tau := \ln 2/c$ instead of c.

First, we examine the effect of k and half-life τ by setting $\beta = 1$. Fig. 3.10 shows the perfor-

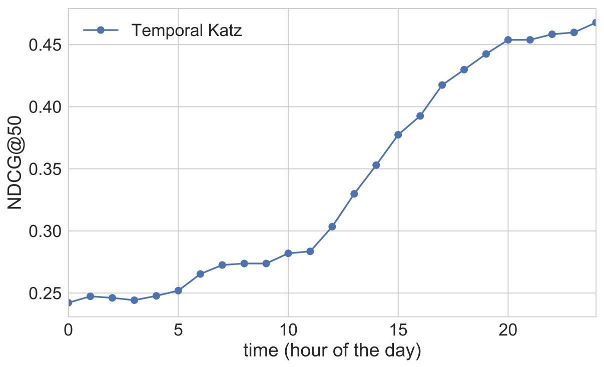


Fig. 3.8: Average daily NDCG@50 performance of temporal Katz centrality on the UO17 data.

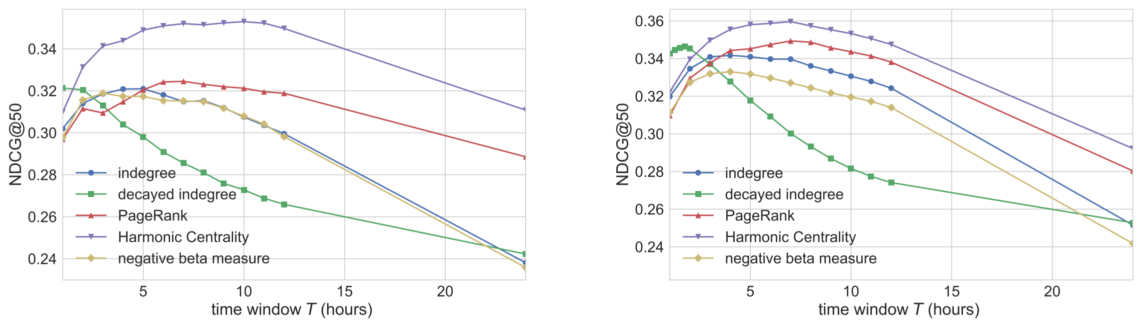


Fig. 3.9: NDCG@50 performance of the baseline methods as the function of time window T. For online decayed indegree results are shown as the function of half-life τ. **Left:** UO17, **Right:** RG17.

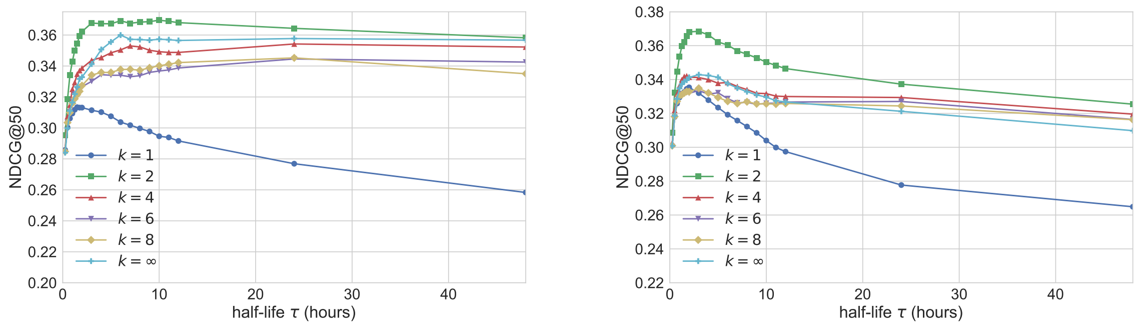


Fig. 3.10: NDCG@50 performance of temporal Katz centrality as the function of half-life parameter τ. Different curves correspond to different values k of truncation. We set $\beta = 1$. **Left:** UO17, **Right:** RG17.

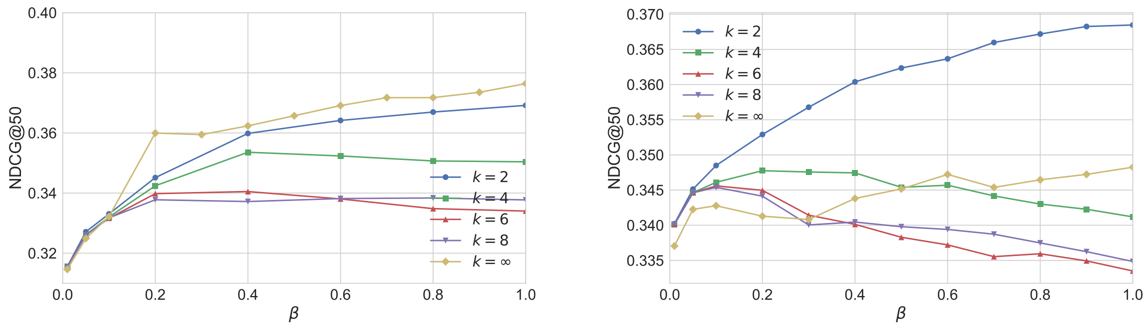


Fig. 3.11: NDCG@50 performance of temporal Katz centrality as the function of parameter β. Different curves correspond to different values k of truncation. We set $\tau = 6h$ for the UO17 data, and $\tau = 3h$ for the RG17 data. **Left:** UO17, **Right:** RG17.

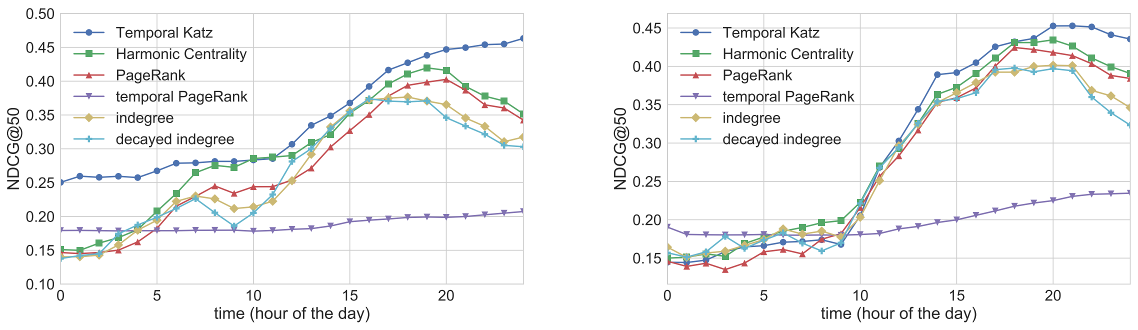


Fig. 3.12: Overall best daily NDCG@50 performance of temporal Katz centrality and the baselines. **Left:** UO17, **Right:** RG17.

Table 3.3: Best average NDCG@50 performance of each centrality metric.

NDCG@50	UO17	RG17
indegree	0.321	0.342
decayed indegree	0.321	0.346
negative beta	0.319	0.333
PageRank	0.325	0.349
temporal PageRank	0.187	0.195
harmonic centrality	0.353	0.359
temporal Katz centrality	**0.370**	**0.368**

mance of temporal Katz centrality at various parameter settings for UO17 the RG17. We plot NDCG@50 against parameter τ. Different curves correspond to different k parameters. The effect of k is significant: Models with $k > 1$ strongly outperform models with $k = 1$, a very simple version of temporal Katz centrality similar to online degree. The best performance can be achieved on both data sets by setting $k = 2$ and $\tau \approx 3h$.

In Fig. 3.11 we analyze the importance of parameter β. For models with larger k (e.g. $k = 8$), the importance of β is to decrease the effect of paths that are too long, with optimal value around $\beta \approx 0.1 - 0.2$. For methods with lower k (e.g. $k = 2$), β is nearly meaningless, and the use of small β in combination with strong exponential decay results in performance deterioration.

The final conclusion of our experiments is drawn in Fig. 3.12 where we compare the hourly performance of each method at their best parameter settings. For temporal Katz centrality we set $\beta = 1, \tau = 3h, k = 2$. In the case of both data sets, temporal Katz centrality can keep up with the performance of harmonic centrality, the strongest baseline model. The quality of temporal PageRank is significantly lower than the quality of other methods. We summarize the best NDCG@50 scores for temporal Katz centrality and the baselines in Table 3.3. Temporal Katz centrality generally performs better than other baselines. Note that only harmonic centrality, a measure that is static and not online updateable, delivers performance comparable to temporal Katz centrality.

We illustrate various centrality measures by showing the 20 accounts with highest score for the Roland-Garros semifinals. On June 9, more than 70 players participated in several categories (Men's singles, Girl's and Boy's singles, etc.). In Table 3.4, we show top accounts at 12:00 by temporal Katz centrality with $k = \infty$ and $\tau = 3h$, and in Table 3.5 for harmonic centrality and decayed indegree, the latter also at 12:00.

We show the accounts of tennis players playing participating in the June 9 semifinals in orange and of those who did not play that day in yellow. For example, women semi-finalists of the previous day, Simona Halep, Timea Bacsinszky, Caroline Garcia and Gabriela Dabrowski are yellow. All methods listed 4–6 daily players among the most central 20 accounts. All methods assigned high centrality to Men semi-finalists Rafael Nadal, Andy Murray, Stanislas Wawrinka and Dominic Thiem. Furthermore, temporal Katz centrality with $\beta = 1.0$ and harmonic centrality could recover two additional young daily players, Whitney Osuigwe and Nicola Kuhn. Retired tennis legends Ana Ivanovic and Gustavo Kuerten are not relevant in our experiment as they did not participate in this event.

Notice that decayed indegree and temporal Katz centrality with $\beta = 0.2$ rank sports media accounts (Tennis Channel, WTA, ATP World Tour, Eurosport) higher compared to harmonic centrality and temporal Katz centrality with $\beta = 1.0$. We did not attempt to curate the relevance to media sources, as the number of such Twitter accounts is abundant. Finally, sponsors 'yonex.com' and 'NikeCourt', as well as the official Twitter account of the event 'Roland-Garros' also rank high. Most of these accounts are active every day, with little observable change in time, which justifies why we do not consider them relevant for the temporal evaluation.

3.6 Conclusion

In this Chapter, we introduced an online updateable, dynamic graph centrality measure based on the Katz index. Our proposed metric can incorporate arbitrary time decay functions to

Table 3.4: Temporal Katz centrality toplist for RG17 semi final day (June 9) at 12:00 with $\beta = 1.0$ (left) and $\beta = 0.2$ (right). Relevant daily players are highlighted orange. Accounts of players who did not play on this day are highlighted yellow.

	$\beta = 1.0$		$\beta = 0.2$	
Rank	Account	Label	Account	Label
1	Simona Halep	0	Roland-Garros	0
2	Stanislas Wawrinka	1	Stanislas Wawrinka	1
3	Andy Murray	1	Andy Murray	1
4	Rafa Nadal	1	Simona Halep	0
5	Roland-Garros	0	Rafa Nadal	1
6	Ana Ivanovic	0	Dominic Thiem	1
7	Timea Bacsinszky	0	Timea Bacsinszky	0
8	Karolina Pliskova	0	Rohan Bopanna	0
9	Rohan Bopanna	0	Ana Ivanovic	0
10	Dominic Thiem	1	WTA	0
11	Gaby Dabrowski	0	Gaby Dabrowski	0
12	Gustavo Kuerten	0	Tennis Channel	0
13	Nicola Kuhn	1	Rafa Nadal Academy	0
14	yonex.com	0	Karolina Pliskova	0
15	Whitney Osuigwe	1	yonex.com	0
16	Caroline Garcia	0	Gusti Fernandez	0
17	NikeCourt	0	rolandgarrosFR	0
18	Novak Djokovic	0	Eurosport.es	0
19	WTA	0	ATP World Tour	0
20	ATP World Tour	0	Caroline Garcia	0

Table 3.5: Harmonic centrality (left) and decayed indegree (right) top list for RG17 semi final day (June 9) at 12:00. Relevant daily players are highlighted orange. Accounts of players who did not play on this day are highlighted yellow.

	Harmonic centrality		decayed indegree	
Rank	Account	Label	Account	Label
1	Roland-Garros	0	Roland-Garros	0
2	Rafa Nadal	1	Andy Murray	1
3	Andy Murray	1	Stanislas Wawrinka	1
4	Stanislas Wawrinka	1	Rafa Nadal	1
5	Simona Halep	0	Dominic Thiem	1
6	Dominic Thiem	1	Timea Bacsinszky	0
7	Rohan Bopanna	0	Simona Halep	0
8	Timea Bacsinszky	0	Rohan Bopanna	0
9	Ana Ivanovic	0	Ana Ivanovic	0
10	Tennis Channel	0	Tennis Channel	0
11	yonex.com	0	Gaby Dabrowski	0
12	WTA	0	Gusti Fernandez	0
13	Caroline Garcia	0	Rafa Nadal Academy	0
14	Rafa Nadal Academy	0	WTA	0
15	Gaby Dabrowski	0	yonex.com	0
16	ATP World Tour	0	Eurosport.es	0
17	Whitney osuigwe	1	Caroline Garcia	0
18	Nicola Kuhn	1	Eurosport UK	0
19	NikeCourt	0	Stéphanie Loire	0
20	Anabel Medina	0	Emilie Lopez	0

emphasize the time-related relevance of the edges based on their time of creation. Our algorithm models information spreading over the stream of edges created subsequently in time.

We presented multiple unsupervised experiments to show that our method can adapt to changes in the distribution of the edge stream. Furthermore, with time decay parameter c and β we can properly control the effect of recently added edges. We also proved that our metric converges to the Katz index in case of static edge distribution.

In order to assess the quality of our centrality measure, we compiled a supervised evaluation for the mention graphs of Twitter tennis tournament collections along with temporal importance ground truth information. To the best of our knowledge, these are the first Twitter collections enhanced with dynamic node importance labels. We made our data set, as well as our codes publicly available[3].

In our final experiment, we compared our temporal Katz centrality metric with static graph-based measures as well as with other dynamically updateable algorithms. We found that temporal Katz centrality can identify accurately and quickly the emerging, new important nodes and that it worked particularly well in the US Open 2017 (UO17) collection.

[3]GitHub repository of our research: https://github.com/ferencberes/online-centrality

Chapter 4

Node embeddings in dynamic graphs

Embedding methods on graphs encode the nodes of the network to vectors in a low-dimensional vector space. In general, representations in the embedded space should reflect the structure of the original graph. Perhaps the most well-known method is Laplacian eigenmaps [17]. Another class of models is based on the adjacency matrix of the graph; one popular example is graph factorization [2]. Recently, random walk-based approaches have been proposed, like Node2Vec [57], LINE [141], and DeepWalk [118]. These methods sample node pairs that co-occur in random walks, and then optimize for their similarity in the embedded space. Walk sampling is motivated by the skip-gram model from natural language processing [94]. Furthermore, the aforementioned techniques can be unified under a matrix factorization framework [124].

Any embedding method can be applied in dynamic graphs by considering graph snapshots in time. However, such solutions do not only react slowly, but also build new representations for every snapshot, hence they require an entire model retraining for downstream machine learning tasks [60].

In this chapter, we propose two online updateable node embedding models that very efficiently update the existing embedding by only considering a small neighborhood in the graph.

4.1 Our results

In order to generate time-aware node embeddings, we have to solve the challenge of maintaining node embeddings for tracking and measuring node properties and similarities as the edges arrive. Most graph algorithms are difficult to update online. For example, to compute random walk-

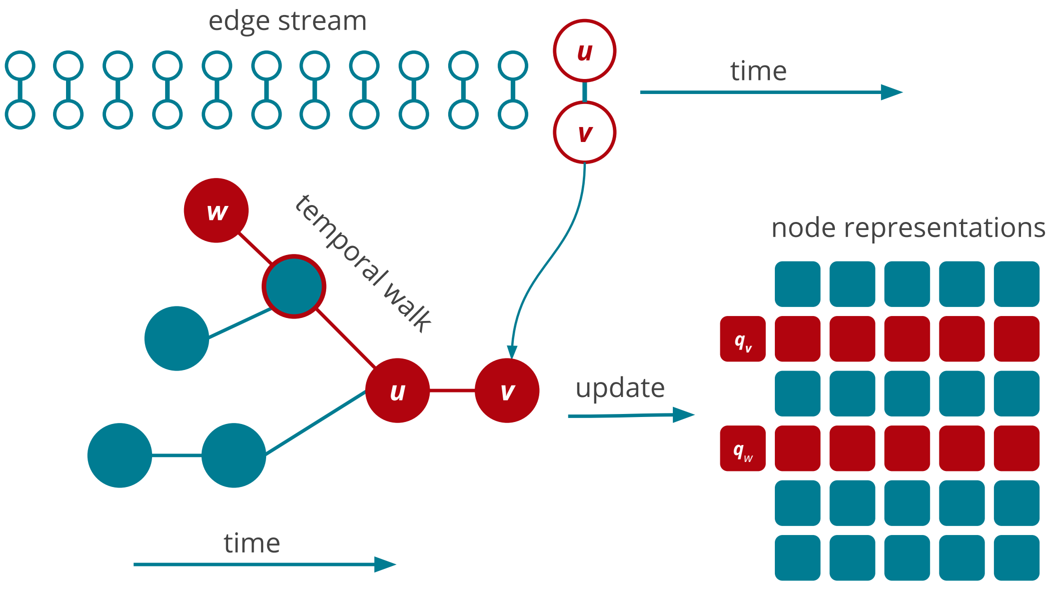


Fig. 4.1: Concept of StreamWalk. When uv edge arrives in the stream, vertices from the temporal neighborhood of u are sampled via temporal random walks. The method optimizes for the similarity of v and the sampled node w.

based embeddings [57], we have to be able to maintain not just the embedding but also the set of walks whenever a new edge appears in the stream.

Our first algorithm is StreamWalk, an online updateable node embedding model based on the concept of time respecting paths introduced in Chapter 3. Compared to earlier results, StreamWalk is capable of updating the set of walks for every single new edge in the stream and not just larger batches of insertions and deletions.

The key ingredients of StreamWalk are online gradient descent optimization [68] and time respecting temporal walks [131]. As illustrated in Fig. 4.1, after the arrival of edge uv in the stream, our algorithm picks node samples for node v from its temporal neighborhood, using temporal walks ending in the latest edge uv. Given the sample w, the optimization step moves the embedding vectors of w and v closer to each other. Our algorithm performs online machine learning [21] by continuously updating a model as new links arrive from the edge stream.

Our second algorithm directly learns the neighborhood similarity of node pairs in the graph stream, which we call second order similarity. By MinHash fingerprinting [48], we efficiently approximate the neighborhood Jaccard similarity of any two nodes at a given time. Then we optimize the embedding to make pairs similar, proportional to the overlap of their neighborhood.

To fully utilize the power of graph embedding, our main focus for evaluation is similarity search, in which we assess the information encoded in the embedding about node pairs rather than just a global property required for predicting links. For similarity search, the prime source of difficulty lies in ground truth compilation as it often requires tedious human effort, especially if the ground truth target also changes dynamically.

As our final result, we design a quantitative experiment for accessing the quality of temporal node embeddings based on the Twitter tennis tournament mention graphs, introduced in Section 2.2, which include temporally changing node labels. In a supervised experiment, we show that online updateable embeddings capture node similarities better than static embeddings.

4.2 Review of related work

Representation learning methods on graphs encode the nodes of the network to points in a low-dimensional vector space. In general, representations in the embedded space should reflect the structure of the original graph. The research area of node embeddings has been recently catalyzed by the Word2Vec algorithm [93], developed for natural language processing. Several node embedding methods have been proposed recently [57,118,124,141] and applied successfully for multi-label classification and link prediction in a variety of real-world networks from diverse domains.

We briefly review the methodology of the above approaches by following [61]. Static embedding methods learn an embedding vector q_u for each node u in the graph. Usually, the objective is to learn vectors that are similar for neighboring nodes, but a few approaches were also proposed to capture structural node similarities [3]. Let $s(u)$ denote the neighborhood of u; then our goal is to satisfy $q_v \approx q_u$ for $v \in s(u)$. Shallow embedding approaches for static graphs differ in the objective function they use to ensure the similarity of the embeddings, and in the definition of the network neighborhood $s(u)$.

Graph factorization [2], GraRep [34], and HOPE [112] optimize for the squared error (SE) over node pairs in the neighborhood:

$$\sum_{u} \sum_{v \in s(u)} \left[q_u q_v - \mathrm{sim}(u,v) \right]^2 ; \tag{4.1}$$

where $\mathrm{sim}(u,v)$ is the similarity of two nodes measured from the graph structure and $q_u q_v$ is the dot product of the two vectors. The definition of the neighborhood is based on the adjacency matrix. Graph factorization calculates with adjacent neighbors, while GraRep uses higher powers of the adjacency matrix, for example, two-hop neighbors.

As a different method, random walk-based approaches [57, 118] sample vertices from the neighborhood of a node. Sampling is done by initiating random walks from node u. Instead of SE, these approaches optimize for cross-entropy loss:

$$\sum_{u} \sum_{v \in s^*(u)} -\log \left[\frac{\exp(q_u q_v)}{\sum_w \exp(q_u q_w)} \right]; \tag{4.2}$$

where $s^*(u)$ is a random sample from the neighborhood of u.

Many of the above mentioned algorithms use the Word2Vec model as an underlying abstraction by training the model, using sampled walks analogously to sentences, and using the learned embeddings as node embeddings. We follow this approach, and also investigate the use of either the input ($W1$) or the output embedding ($W2$) of the model [123] as the vector space representation of the graph.

The above models learn static embeddings on graph snapshots; however, they mention extensions towards online learning from graph streams. In DeepWalk [118], the possibility of an online incremental update is proposed but not analyzed. An incremental update for LINE with a batch of edge insertions and deletions is described in [160], but no attempt is made to analyze the online, single edge insertion behavior. Closest to our work is the continuous-time dynamic network embedding result [108], which does not learn online but computes an embedding for a single point in time. Similarly, the HTNE algorithm [167] produces a temporal node embedding, but training is done in batch instead of executing online updates. A promising direction for computing the embedding dynamically involves recurrent neural networks, for example, Long Short-Term Memory networks [152]. A few results on applying recurrent networks for graphs have appeared after our work [13, 82, 116, 162].

4.3 Dynamic Vector Space Embedding Methods in Edge Streams

We describe two node embedding approaches that are applicable in edge streams. The input of both algorithms consists of an edge stream (u, v, t) ordered by time t in which each edge can occur multiple times. As required by the data stream algorithmic model, we process the edges in the order of arrival without storing the entire input.

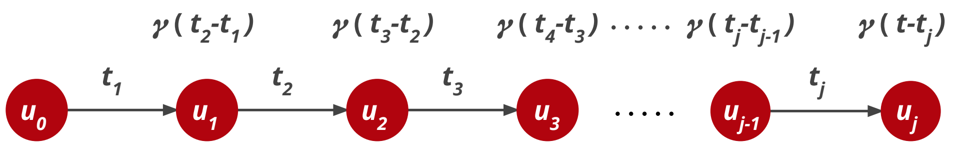


Fig. 4.2: Temporal walk. Adjacent edges in the graph are in time respecting order. The probability of the walk is based on the delays between the appearances of the adjacent edges.

Our goal is to dynamically learn node representations by reflecting the current node similarity structure of the evolving graph as we dynamically change the location of the nodes in the vector space. To this end, we give two embedding methods in the next two subsections. Two nodes are required to be mapped close in the vector space whenever they lie on short paths formed by recent edges in the first model, and whenever the set of their recent neighbors is similar in the second model.

4.3.1 Similarity based on reachability through short temporal walks

In our first algorithm, our goal is to enforce that the embedding of node v be similar to the embedding of nodes with the ability to reach v across edges that appeared recently, as shown in Fig. 4.1. In other words, the embedding of a node should be similar to the embedding of nodes in its temporal neighborhood. We define time respecting *temporal walks* [131] in order to sample for each node u at any time t nodes from its temporal neighborhood. As seen in Fig. 4.2, a temporal walk consists of adjacent edges ordered in time:

$$z = (u_0, u_1, t_1), (u_1, u_2, t_2), \cdots, (u_{j-1}, u_j, t_j); t_{i-1} \leq t_i. \tag{4.3}$$

For example, there are three temporal walks leading to node v in Fig. 4.4: e_1, e_3, and e_2, e_3. Since edges can appear multiple times, we consider the edge set as a multiset and distinguish between the walk (e_2, e_3), which is a temporal walk, from (e_2, e_1), which is not, since e_1 comes earlier than e_2.

To define the similarity, we want to give more weight to shorter walks and more weight to fresh edges. Towards this end, for a temporal walk where edges appeared at $(t_1, t_2, \ldots, t_j)$, we define the probability of the walk at time t as

$$p(z, t) := \beta^{|z|} \cdot \prod_{i=1}^{j} \gamma(t_{i+1} - t_i); \tag{4.4}$$

Table 4.1: Notations used in the StreamWalk algorithm.

z	a temporal walk
uv	the new directed edge that is being processed in the edge stream
$t(xy)$	time of arrival of a multi-edge instance
t_u	last time an edge arrived leading to u
β	decay exponent for the length of the walk
γ	time-aware edge weighting function
$p(z, t)$	weight of walk z at time t
$p(w, t)$	sum of weight of walks ending in node w at time t
$p(xy)$	sum of weight of walks ending with edge xy at time $t(xy)$

where $\beta \leq 1$ is an exponential decay on the length of the walk, $t = t_{j+1}$, and $\gamma(\tau)$ is a time-aware weighting function that is based on the delay τ between adjacent edges. The concept of (4.4) is that a walk is more likely if edges along the walk appeared close to each other in time. We use exponential time weight $\gamma(\tau) := \exp(-c\tau)$. Since $\gamma(a) \cdot \gamma(b) = \gamma(a + b)$, the probability of the path in (4.4) becomes

$$p(z, t) = \beta^{|z|} \exp(-c(t - t_j)) \cdot \ldots \cdot \exp(-c(t_2 - t_1)) = \beta^{|z|} \exp(-c(t - t_1)). \tag{4.5}$$

The notation is summarized in Table 4.1.

Temporal walk sampling from edge stream

Given a node v, a naive idea would be to compute the walk weight $\sum\{p(z, t) : z$ is a temporal walk from w to $v\}$ for all other nodes w and set the embedding of w close to that of v proportional to the walk weight. The problem with this approach is that it requires a time consuming walk enumeration procedure at each time instance, and has no ability to update the similarity measure by focusing only on the new edges as they arrive.

Given the new edge uv that arrives at time t, we would like to only consider walks from any w that reach v by the new edge uv. Towards this end, we propose a sampling update procedure for temporal walks as follows. We select a start node w of a random temporal walk z ending in u with probability proportional to $p(z, t)$ in (4.5); see Fig. 4.3. We generate the walks by taking steps backwards from u. To make sure the walks are temporal, we always use edges that appeared before the previous one. Among the possible edges entering the current node, we

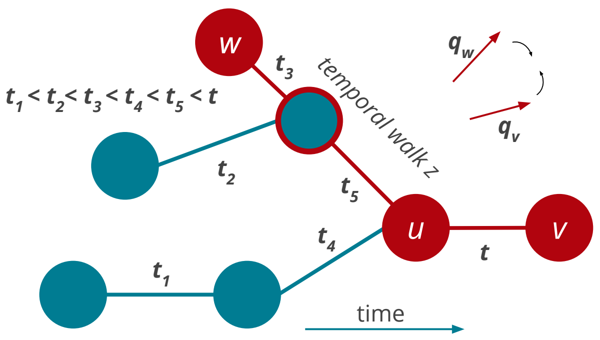
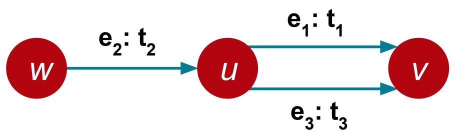


Fig. 4.3: StreamWalk: learning embeddings based on random walks to the past in temporal networks.



time	$p(e_1,t)$	$p(e_2,t)$	$p(e_3,t)$	$p((e_2,e_3),t)$	$p(v,t)$
$t=t_1$	β	n/a	n/a	n/a	$p(e_1,t_1)=\beta$
$t=t_2$	$\beta\gamma(t_2-t_1)$	β	n/a	n/a	$p(e_1,t_2)=\beta\gamma(t_2-t_1)$
$t=t_3$	$\beta\gamma(t_3-t_1)$	$\beta\gamma(t_3-t_2)$	β	$\beta^2\gamma(t_3-t_2)$	$p(e_3,t_3)+p((e_2,e_3),t_3)+p(e_1,t_3)=$ $\beta(1+p(e_2,t_3))+\gamma(t_3-t_2)p(v,t_2)$

Fig. 4.4: Computation of $p(v,t)$ for the arrival times $t_1 < t_2 < t_3$ of the three edges e_1, e_2, and e_3. The bottom right cell illustrates the update formula (4.7).

select proportional to the time-aware weighting function γ. For example, in Fig. 4.3, we select t_5 backwards from u, and then t_3 backwards from the next node. Finally, we also compute a stopping probability corresponding to the length decay β so that we select no new edge from w in the example; the actual formula (4.10) is explained later.

The actual implementation is somewhat tricky in that we have to handle multi-sets of edges. A way to illustrate the implementation is to consider an edge uv that appears before another wu and then reappears, see Fig. 4.4. The second instance can form a temporal walk w, u, v, while the same walk is not temporal with the first instance of uv. However, the second instance of uv has a higher edge weight γ, hence we have to store the weight of the first instance as well to be able to correctly compute the weight of all temporal walks that reach node v.

The implementation of the StreamWalk algorithm

In Algorithm 1, we describe StreamWalk, our implementation of temporal walk sampling. Recall that the notation is summarized in Table 4.1. For every edge uv in the multi-set of edges arriving in the stream, we maintain the total weight of all walks ending at v at time $t(uv)$:

$$p(v, t(uv)) = \sum_{z} p(z, t(uv)); \tag{4.6}$$

where we sum over all temporal walks z ending in v using edges arriving no later than $t(uv)$. The actual computation in procedure UPDATEWALKS accumulates the weight of the walks seen in Fig. 4.5. There is a new single edge temporal walk uv with weight β. Furthermore, we can continue each temporal walk z that ended in u before $t(uv)$ with uv. The total weight of these walks is $p(u, t_u) \cdot \beta \cdot \exp(-c(t(uv) - t_u))$ where t_u is the most recent timestamp for which $p(u, t_u)$ is known. In other words, t_u denotes the last time an edge entering u arrived in the edge stream. The exponential term accounts for the time decay of temporal walk weights since the arrival of this last edge entering u. Finally, we add all the walks that terminated at v before, with exponential time decay. The final formula becomes

$$p(v, t(uv)) \leftarrow \beta \cdot (1 + p(u, t_u) \cdot \exp(-c(t(uv) - t_u))) + p(v, t_v) \cdot \exp(-c(t(uv) - t_v)); \tag{4.7}$$

where t_v is the most recent timestamp for which $p(v, t_v)$ is known. The update rule is illustrated in the last step in Fig. 4.4 and in Fig. 4.5.

For each edge uv in the stream, we finally update the embedding of v by sampling a fixed number of temporal walks ending in u; we do this by calling procedure SAMPLEWALKS k times as described at the end of this section. Given the start node w of a walk in the sample, we optimize for the similarity of the embedding pair (q_v, q_w) with stochastic gradient descent. For loss function, we either set MSE or cross-entropy as in equations (4.1) and (4.2). In the case of MSE, for each w we apply online negative sampling [115] by selecting pairs vw' proportional to the popularity of w' in the edge stream up to the current timestamp. We refer to [69] for online incremental updates for cross-entropy based loss.

Since we train by sampling k walks per edge, time complexity is affected by the cost of sampling temporal walks. To reduce storage, we can work over a sliding window of the stream and periodically remove the oldest edges; these edges will already have a very small γ value.

Finally, we describe the algorithm to sample temporal walks as implemented in Procedure SAMPLEWALKS of Algorithm 1. Our goal is to sample proportional to $p(y, \tau)$ at a given time τ. We

Algorithm 1 *StreamWalk.*

```
procedure UpdateWalks($u, v$)
                                        ▷ Update the weight for all walks ending at $v$
    $t_u, t_v \leftarrow$ last timestamp such that $p(u, t_u)$ and $p(v, t_v)$ are known, respectively
    $p(v, \text{now}) \leftarrow \beta \cdot (1 + p(u, t_u) \cdot \exp(-c(\text{now} - t_u))) + p(v, t_v) \cdot \exp(-c(\text{now} - t_v))$
    $t(uv) \leftarrow$ now

procedure SampleWalks($y, \tau$)        ▷ Recursively sample a temporal walk ending at $y$
    $t \leftarrow$ most recent timestamp with $t \le \tau$ such that $p(y, t)$ is known
    $p(y, \tau) \leftarrow p(y, t) \cdot \exp(-c(\tau - t))$
    With probability $1/(1 + p(y, \tau))$ do
        return $y$
    else
        for all $xy$ multi-edges with $t(xy) < \tau$ do
            Select $x$ with probability $p(xy) \cdot \exp(-c(\tau - t(xy)))/p(y, \tau)$
        return SampleWalks($x, t(xy)$)

procedure StreamWalk($u, v$)
                                        ▷ Update embedding for $v$
    call $UpdateWalks(u, v)$
    repeat $k$ times
        $w \leftarrow$ SampleWalks($u$, now)
        Optimize the representations $q_w$ and $q_v$ by equation (4.1) or (4.2)
```

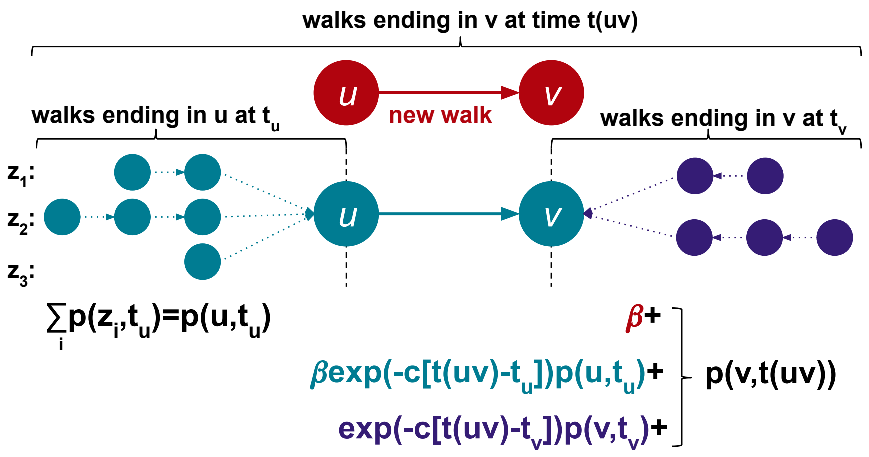


Fig. 4.5: Whenever a new edge uv appears, a new walk starts from u (red), and each temporal walk (z_1,z_2,z_3) that ended in u up to time t_u continues via uv (blue). We get $p(v, t(uv))$ by summing up the contribution of the previous two type of walks (red and blue) with the decayed weight of walks that have already reached node v (purple) before time $t(uv)$.

define a random walk backwards from y. We select a backward edge with probability proportional to the weight of walks ending with that edge, which we define as

$$p(xy) = \sum_z p(z, t(xy)); \tag{4.8}$$

where z are temporal walks ending with the given instance of the edge xy that appeared at time $t(xy)$. Recall that the edges are taken from a multi-set. The value of $p(xy)$ can be calculated as follows. From the total temporal walk weight ending in y at time $t(xy)$, we have to subtract the total weight of all walks ending in y before $t(xy)$; the difference contains the weight of only those paths that use the edge instance xy of timestamp $t(xy)$:

$$p(xy) = p(y, t(xy)) - p(y, \bar{t}) \cdot \exp(-c(t(xy) - \bar{t})); \tag{4.9}$$

where $\bar{t} < t(xy)$ is the timestamp of the last edge in the stream entering y before $t(xy)$. The exponential term corresponds to the time decay of the walk weight since time $\bar{t}$.

We also define the termination for the walk, which is based on the contribution of the single node y as a zero-edge walk relative to all other walks that end at y. At any time of observation τ, the weight of the zero-edge walk is 1, and the total weight of the remaining walks is $p(y, t)$ for the last recorded time $t \leq \tau$, decayed proportional to the elapsed time, $\tau - t$. Hence with the probability below, we take no further steps but stop the walk:

$$1/(1 + p(y, t) \cdot \exp(-c(\tau - t))). \tag{4.10}$$

The steps of Procedure SampleWalks are summarized as follows.

1. We start the random walk from $y \leftarrow u$ and set $\tau = \text{now}$.
2. With probability such as in equation (4.10), we stop the walk and return the current node y.
3. Optionally, we can also terminate the walk if its length reaches a predefined limit.
4. Else, we select an edge xy with $t(xy) < \tau$ with probability proportional to the time-decayed total weight of walks ending with xy, which is $p(xy) \cdot \exp(-c(\tau - t(xy)))$ by definition.
5. We repeat from step 2 by setting $y \leftarrow x$ and $\tau \leftarrow t(xy)$.

As the final implementation details, we can sample by selecting a random value between zero and $p(y, \tau)$ and binary search in the multi-set of xy edges ordered by $t(xy)$. For a given edge xy, we compute $p(xy)$ by equation (4.9) and continue the binary search based on the time-decayed value $p(xy) \cdot \exp(-c(\tau - t(xy)))$. Lastly, it can happen that sampling intends to select a very old edge that was already deleted from the sliding window. This happens when binary search does not terminate at the oldest t still kept in the records. In this case, we can repeat the sampling with a new random value.

4.3.2 Online Learning of Second Order Node Similarity

Our next online algorithm optimizes the embedding to match the neighborhood similarity of the nodes, which we call second order proximity by following [141]. Our goal is to optimize for (4.1) online, by considering $\text{sim}(u, x)$ as a time-aware Jaccard similarity of the neighborhood of u and x, as illustrated in Fig. 4.6. We consider the neighbors y of u as a multi-set $N(u, t)$ in which we use the decayed weight of edge uy as the weight of y:

$$w(y) = \exp(-c(t - t(uy))); \tag{4.11}$$

where $t(uy)$ is the time the corresponding instance of edge uy appeared in the stream. Whenever we add a new edge to u, we discard elements $y \in N(u, t)$ with probability $1 - w(y)$. This way we emphasize the importance of new edges and also limit the size of $N(u, t)$ by discarding old edges with low weight that have little effect on similarity values.

In order to design a streaming algorithm to compute second order similarity, we face the same problems as in the StreamWalk algorithm: we want to focus on the increase of similarity when we

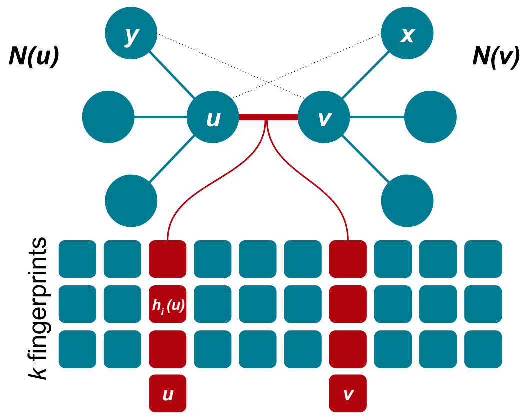


Fig. 4.6: Online learning of second order node similarity.

add a new edge uv, and we want to avoid the costly full computation of similarities of u with all neighbors x of v. Note that the similarity of x and u depend on their neighborhood, which means that all nodes of distance two from v should be enumerated for the full computation. In the next subsection, we describe a randomized approximation method for neighborhood similarities based on [48], which will be used in our final algorithm.

Approximation by fingerprinting

Our algorithm relies on MinHash fingerprinting [27] to approximate the Jaccard similarity. The notations are summarized in Table 4.2. Let there be k independent random permutations over the nodes π_i for $i = 1 \ldots k$. We define the k fingerprints of A as

$$h_i(A) := \text{argmin}\{\pi_i(a) : a \in A\}; \quad h_i(\emptyset) = \text{NaN}; \quad i = 1 \ldots k; \tag{4.12}$$

For short,

$$h_i(u) := h_i(N(u,t)). \tag{4.13}$$

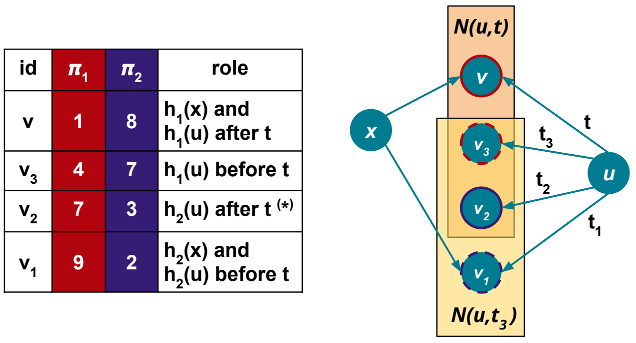
We maintain k fingerprints defined in (4.12) for the neighborhood of each node where the weights of the elements are defined by (4.11). We approximate the time-aware Jaccard similarity of any node pair with the fraction of common fingerprint values:

$$\text{sim}(u,v,t) \approx \sum_{i=1}^{k} I[h_i(u) = h_i(v)]/k. \tag{4.14}$$

We illustrate the fingerprinting idea in Fig. 4.7 for $k = 2$. The two fingerprints of u, $h_1(u)$ and $h_2(u)$, are defined based on two permutations π_1 and π_2 of the entire vertex set. The permutations are fixed, but the fingerprints change in time as new edges arrive and past edges become too old and get removed from $N(u,t)$.

Table 4.2: Notations used in the second order similarity algorithm.

uv	the new directed edge that is being processed in the edge stream
$t(xy)$	time of arrival of a multi-edge instance
π_i	a permutation of the entire vertex set, fixed in time
$N(u,t)$	the pruned neighbors of u at time t
$h_i(u) = h_i(N(u,t))$	the i-th fingerprint of u at time t



id	π_1	π_2	role
v	1	8	h_1(x) and h_1(u) after t
v_3	4	7	h_1(u) before t
v_2	7	3	h_2(u) after t (*)
v_1	9	2	h_2(x) and h_2(u) before t

Fig. 4.7: Illustration of how the fingerprints of node u change when adding the new edge uv. Neighbors of u are ordered in time as $t_1 < t_2 < t_3 < t$. Two fixed random permutations π_1 and π_2 define the fingerprints $h_1(u)$ and $h_2(u)$. In π_1 (red), v has minimum value, hence the previous $h_1(u)$ will be reassigned to v. In π_2 (purple), the minimum is the oldest node v_1, which becomes too old and gets removed from $N(u,t)$. The correct value for $h_2(u)$ would be v_2 $(\star)$. Instead we heuristically set $h_2(u) = v$ after the removal of v_1.

Next, we show how the similarity of u and a neighbor x of v can be approximated in the example of Fig. 4.7. Assume that $h_1(x) = v$ and $h_2(x) = v_1$. By using formula (4.14), before edge uv arrives, the similarity approximation is $\text{sim}(u,x,t_3) \approx (0+1)/2$ as $h_2(x) = h_2(u) = v_1$ at time t_3. When edge uv arrives, the similarity will on one hand increase, since $h_1(u)$ gets assigned with v. On the other hand, the similarity can decrease as edges become too old. For example, if we drop edge uv_1, equation $h_2(x) = h_2(u) = v_1$ will no longer hold. However, since we want to avoid the cost of updating $h_i(x)$ for all i and all neighbors x of v, we heuristically only consider the increase of similarity, which can be caused by adding v as new fingerprint of u.

As a final heuristic, in our implementation we always replace fingerprints corresponding to pruned neighbors by v, since obtaining the π_i values of the entire neighborhood is computationally costly. In the example of Fig. 4.7, we drop edge uv_1 as $t_1 \ll t$. The correct new value of $h_2(u)$ would be the next oldest vertex v_2, however this can only be calculated by enumerating all neighbors of u. Instead, in our implementation we heuristically assign $h_2(u) \leftarrow v$.

Algorithm 2 Online learning second order similarity

```
procedure UpdateFingerprints($(u, v)$)
    for all $i$ in $1 \dots k$ do
        if $h_i(u)$ is too old or $\pi_i(v) < \pi_i(h_i(u))$ then
            $h_i(u) \leftarrow v$

procedure GetSimilarityDelta($(u, v, x)$)
    $\ell \leftarrow 0$
    for all $i$ in $1 \dots k$ do
        if $h_i(u) = h_i(x) = v$ then
            $\ell \leftarrow \ell + 1$
    return $\ell$

procedure OnlineSecondOrderSim($(u, v)$)
    UpdateFingerprints($(u, v)$)
    for all in-neighbors $x$ of $v$ that are not too old do
        $\ell \leftarrow$ GetSimilarityDelta($(u, v, x)$)
        Optimize the representations $q_u$ and $q_x$ repeated $\ell$ times, by using equation (4.1) or (4.2)
    Repeat with $v$ and $u$ swapped and edge directions reversed
```

Algorithm for Online Learning Second Order Similarity

Our method is described in Algorithm 2 by using the notations in Table 4.2. Our goal is to approximate the change of similarity between u and the in-neighbors x of v, and modify the embedding vectors whenever certain x gets more similar to u after adding the new edge uv. Note that x becomes more similar if the edge xv also appeared recently; in terms of fingerprints, this means that for some fingerprint index i, both x and u have v as fingerprint node. We perform the steps below to update the fingerprints of u and check for v as fingerprint in the in-neighbors x of v:

1. For node u, we maintain the present neighborhood $N(u)$ by removing very old edges, and recompute the k fingerprints $h_i(u)$ for $i = 1 \dots k$ by calling Procedure UpdateFinger-

PRINTS. Fingerprint $h_i(u)$ can take the new value v for the new edge uv if it is too old or if $\pi_i(v)$ becomes the new MinHash value for permutation i. In the former case, we can either heuristically replace $h_i(u)$ with the new neighbor v or compute the true MinHash value $\text{argmin}\{\pi_i(y) : y \in N(u)\}$.

2. Finally, for each in-neighbor $x \in N(v)$, we compute the number of fingerprints ℓ that match those of u and have value v in Procedure GETSIMILARITYDELTA, and ℓ times optimize the representations q_u and q_x by using equation (4.1) or (4.2).

Symmetrically, we also check for the similarity increase of v with the out-neighbors of u by performing the same steps, replacing u and v on the reverse direction graph.

4.4 Similarity Search Experiments

In this section, we describe our evaluation, in which we assess how well the closeness of two nodes in the embedding reflect their similarity against an external ground truth. Towards this end, we first describe a network enriched with a time dependent external similarity ground truth information. Then, at a time instance, we compute the list of nodes closest to selected ones in the embedding, and compare these lists against the similarity ground truth.

We analyze node embedding methods for similarity search over the Twitter tennis tournament collections of Section 2.2. We use the mention graphs extracted from the last 15 and 14 days of RG17 and UO17, respectively. For the quantitative analysis, we use the annotation of the nodes for the accounts of the tennis players that participate in a game on a given day as described in Section 2.2.2. In this sense, we expect that the players of the same day are more similar than other players and non-player accounts, as we will describe in Section 4.4.1. We compare the performance of StreamWalk and online second order similarity with online and static baseline methods, which we will describe in Section 4.4.2.

4.4.1 Evaluation Metrics

We evaluate similarity search by a supervised experiment. In Section 2.2.2, we defined *active accounts* as those corresponding to players with a game on the given day. For an active account as query, we consider the other active account as the required similar pair. For each embedding algorithm, we generate 128-dimensional node representations every six hours (6:00, 12:00, 18:00, 24:00). For online methods, we perform continuous updates over the edge stream. For the static methods, we build the corresponding graph snapshots.

We use NDCG [4] to evaluate how other active accounts are similar to a selected one. NDCG is a measure for ranked lists that assigns higher score if active accounts appear with higher rank in the similarity list. In our experiments, we compute the average of the NDCG@100 for the active accounts as query nodes to measure the performance of a single model in any given snapshot.

4.4.2 Baseline Models

We compare StreamWalk and online second order similarity to *online* (or time-aware) and *static* (or batch) embedding methods. Online models are updated after the arrival of each edge. By contrast, static representations are only updated once every six hours when the graph snapshot ends. At hour t a static model is computed on the graph constructed from edges arriving in time window $[t - T, t]$ from the edge stream. For each batch baseline, we experimentally select the best value of T.

We consider four static centrality measures as baseline:

- DeepWalk [118]
- Node2Vec [57]
- LINE [141]: the first and the second order versions of LINE, as well as the combination of the two versions
- Static indegree, calculated in time window $[t-T, t]$ by counting each edge with multiplicity

Furthermore, we compare our proposed algorithms with a simple online baseline:

- Decayed indegree, defined for node u at time t as

$$\sum_{zu \in E(t)} \exp(-c(t - t_{zu})); \tag{4.15}$$

  where $E(t)$ is the multi-set of edges that occurred up to time t with edge activation time t_{zu}.

We use the 128-dimensional representations of StreamWalk (SW), second order similarity (SO), DeepWalk, Node2Vec, and LINE to measure node similarity over time. For the two degree methods, we rank by degree without reference to the query node in the NDCG@100 formula.

4.4.3 Results

In our experiments, we measure how the similarity of node representations evolves over time by a supervised evaluation in which the active nodes should be similar to each other. We show two different ways to describe the performance of a single model:

1. For each day, we present the mean NDCG@100 of the snapshots evaluated at 6:00, 12:00, 18:00, and 24:00.

2. As a single global value (NDCG@100), we take the average of NDCG@100(u) for each daily player u in every snapshot.

For a given parametrization of every embedding-based method, we always show the average performance of ten independent instances.

During our experiments, we found that the following parameters had a great impact on the quality of online node embeddings, see Table 4.3:

W1 and W2: In Word2Vec, we have the option to optimize node representations for the input ($W1$) or the output ($W2$) matrices [123]. It is application dependent whether $W1$ or $W2$ yields the better representation. For SW, we achieved the best results by $W2$.

Initialization: We experimented with Xavier [53] and uniform random initialization of $W1$ and $W2$.

Mirror: In our algorithms, the input to Word2Vec consists of node pairs. Given a training instance (x, y), we *mirror* if we feed both (x, y) and (y, x), not just (x, y).

Decay: We heuristically map the representations of nodes with no recent activity to the null

Table 4.3: Mean global performance (NDCG@100) of StreamWalk with different settings on the first three day of RG17 and UO17 respectively.

matrix	init type	mirror	decayed	RG17	UO17
W1	Xavier	yes	no	0.1865	0.1695
W1	uniform	yes	no	0.2001	0.1818
W2	uniform	yes	no	0.2898	0.2386
W2	uniform	no	no	0.3272	0.2898
W2	uniform	no	yes	0.3341	0.2999

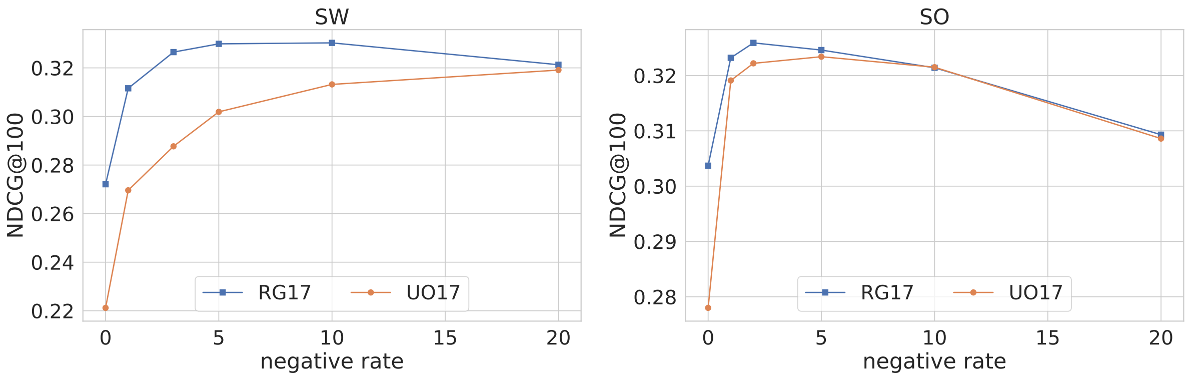


Fig. 4.8: The effect of negative sample rate on the global mean performance (NDCG@100) of StreamWalk (**left**) and online second order similarity (**right**).

vector.

Negative sampling rate and past positive samples used: Key parameters of Word2Vec analyzed separately in Fig. 4.8–4.9. Past positive samples are edges that appeared longer time ago; using such edges for negative training helps forgetting the past.

Next, we examine the quality of node representations with respect to node pair *sampling-related parameters*:

Time decay and half-life: By transforming the time decay parameter c, we show our results as the function of half-life $h = \ln(2)/c$ in Fig. 4.10.

Number of walks sampled: The number of new training instances for SW and SO for a new edge arrival is a parameter analyzed in Fig. 4.11.

We also combine the output of StreamWalk (SW) and second order similarity (SO) by using the weighted average of the corresponding inner products as similarity. This method denoted as SW+SO outperforms SW and SO, as seen in Fig. 4.12. The optimal weight of SO in the combination is 0.3 for both RG17 and UO17.

In Table 4.4 we present the best global mean performance for each model. Fig. 4.13 shows the daily mean performance of the best models.

For illustration, in Table 4.5 we present the 20 accounts most similar to that of Rafael Nadal for different node embeddings on 2017-May-31 18:00. Since Rafael Nadal played on this day, the active accounts (yellow) belong to tennis players who participated in a game on this day. The combined model SW+SO has the highest number of active player accounts. Furthermore,

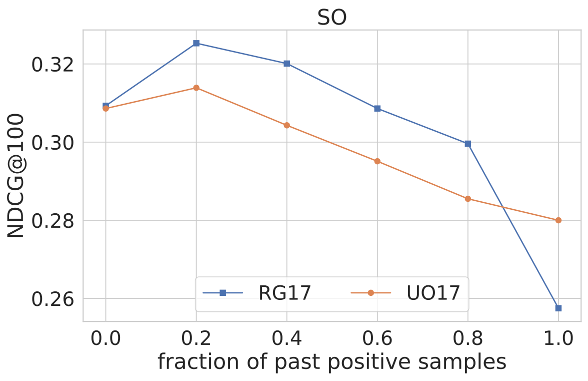


Fig. 4.9: The effect of past positive samples used as negative samples for SO when the latest edge arrives.

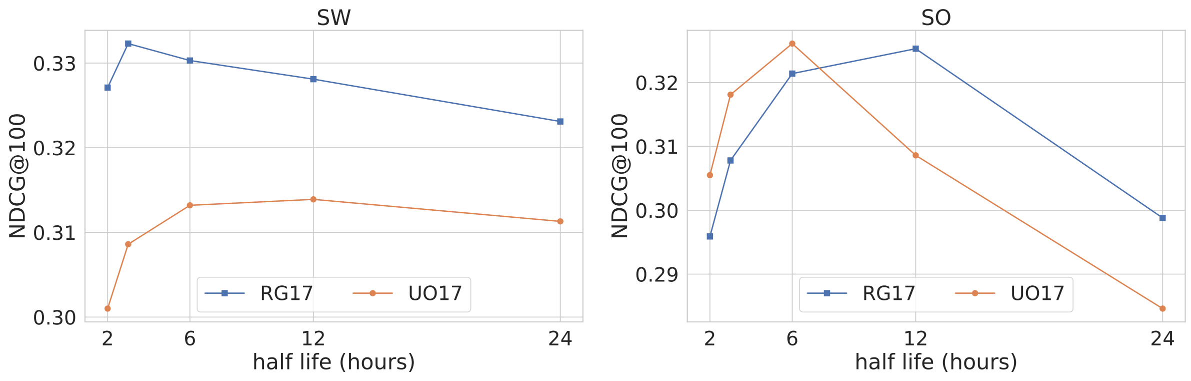


Fig. 4.10: The effect of half-life on the global mean performance (NDCG@100) of StreamWalk (**left**) and online second order similarity (**right**).

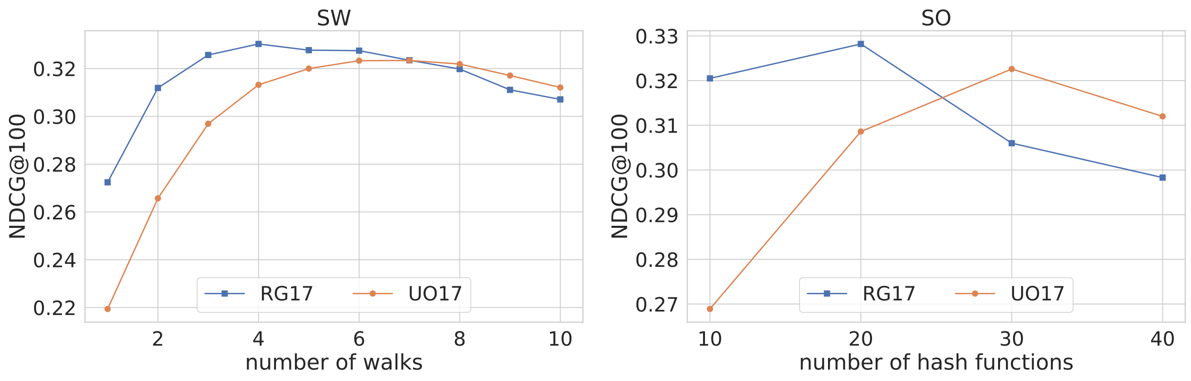


Fig. 4.11: The effect of sampled walks (**left**) and hash functions (**right**) on the global mean performance (NDCG@100) for SW and SO respectively. These parameters control the number of sampled node pairs we feed to online Word2Vec at every edge arrival.

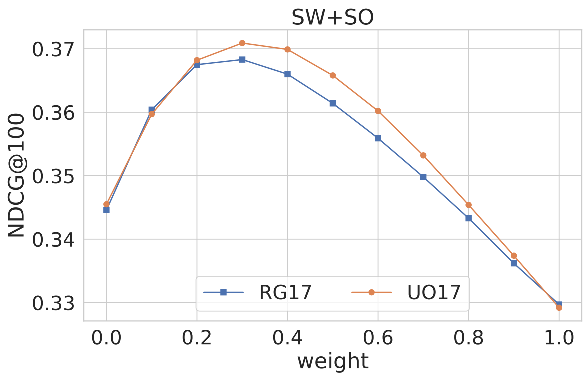


Fig. 4.12: Global mean performance of the combination of SW and SO with respect to the weight of SO. The combined model (SW+SO) has superior performance with the optimal weight 0.3.

Table 4.4: Best mean global performance (NDCG@100) of each model for the Twitter tennis data sets. Best performing methods are marked boldface.

model	RG17	UO17
SW+SO	**0.3683**	**0.3709**
SW	0.3446	0.3455
SO	0.3283	0.3261
LINE	0.2946	0.2936
Node2Vec	0.2617	0.2562
decayed indegree	0.2973	0.2548
static indegree	0.2288	0.2073

accounts present in both SW and SO columns (e.g. *B. Mattek-Sands*, *N. Djokovic*, *G. Dimitrov*, etc.) typically achieve higher position by SW+SO than SW. It is interesting to see that SW and SO find different active accounts, which explains why the combination SW+SO achieves superior performance. While static LINE and Node2Vec have less relevant hits than our online methods, most of the irrelevant accounts still belong to tennis players (e.g. *A. Murray*, *S. Wawrinka*, etc.). The main difference is that the daily active players are better found by the online than the static methods.

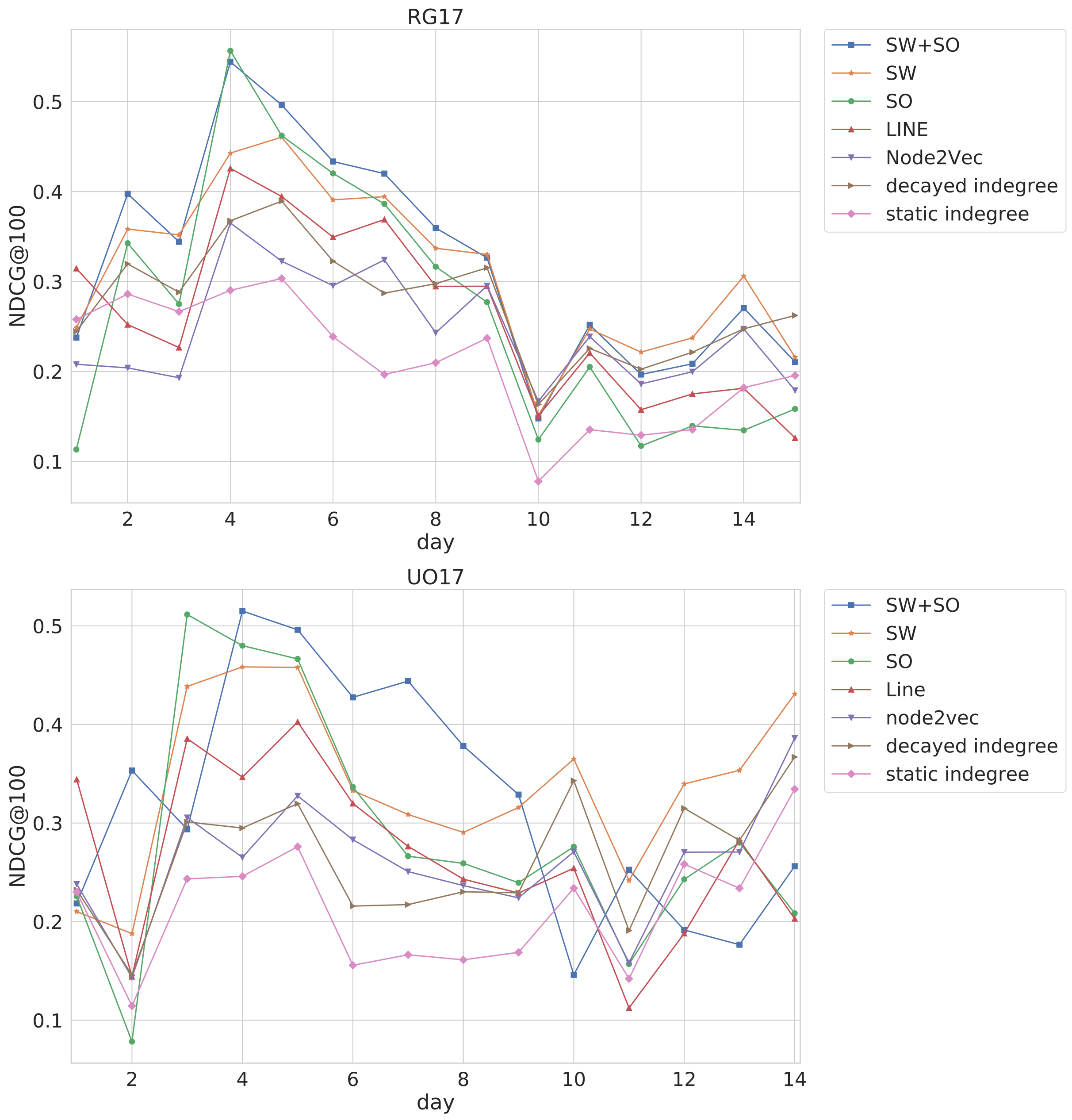


Fig. 4.13: Best model performance over time for RG17 (**top**) and UO17 (**bottom**) tennis data sets. For each day the average NDCG@100 over the four daily snapshots is presented.

Table 4.5: Similarity list of Rafael Nadal for models based on embeddings generated at 18:00 on May 31 (fourth day of RG17). Active player accounts are highlighted in yellow. With the exception of a few renamed accounts (since 2017) most of the presented accounts can be still searched for on Twitter.

	SW+SO	**SW**	**SO**	**LINE**	**Node2Vec**
1	R. Haase	R. Haase	N. Basilashvili	N. Djokovic	R. Olivo
2	K. Mladenovic	K. Mladenovic	A. Hesse	B. Paire	G. Monfils
3	B. Mattek-Sands	S. Johnson	G. Muguruza	A. Murray	K. Mladenovic
4	S. Johnson	B. Mattek-Sands	G. Dimitrov	F. Verdasco	N. Djokovic
5	B. Coric	B. Coric	N. Djokovic	G. Muguruza	D. Schwartzman
6	rolandgarros	rolandgarros	M. Raonic	JW. Tsonga	JW. Tsonga
7	N. Djokovic	C. Wozniacki	PH. Herbert	R. Haase	A. Murray
8	C. Wozniacki	N. Djokovic	A. Bedene	JM. del Potro	A. Agassi
9	Cici Bellis	Cici Bellis	P. Kvitova	G. Monfils	K. Nishikori
10	S. Errani	JP. Sousa	rolandgarros	C. Wozniacki	H. Zeballos
11	G. Muguruza	S. Errani	WTA	S. Wawrinka	G. Muguruza
12	JP. Sousa	G. Muguruza	P. Parmentier	V. Williams	L. Mladenovic
13	G. Dimitrov	V. Williams	D. Thiem	K. Mladenovic	G. Bouchard
14	V. Williams	G. Dimitrov	K. Mladenovic	D. Thiem	N. Kyrgios
15	M. Raonic	R. Olivo	D. Cibulkova	M. Granollers	T. Kokkinakis
16	JW. Tsonga	S. Williams	A. Cornet	N. Kyrgios	JM. del Potro
17	J. Chardy	JW. Tsonga	D. Goffin	P. Kvitova	Babolat
18	A. Kontaveit	A. Kontaveit	G. Monfils	C. Garcia	V. Williams
19	D. Thiem	K. Edmund	B. Mattek-Sands	Suspended	F. Verdasco
20	D. Goffin	S. Bolelli	T. Bacsinszky	R. Gasquet	D. Ferrer

4.5 Conclusion

We introduced two online machine learning algorithms to extract temporal node representations from graph streams. The StreamWalk algorithm optimizes for the similarity of node pairs extracted along temporal walks from the data stream, whereas online second order similarity efficiently learns neighborhood similarity over graph streams by MinHash fingerprinting.

We measured the quality of these models in a supervised evaluation task that we implemented using daily changing node relevance labels. In the RG17 and UO17 Twitter collections, we analyzed the similarity of node representations over time for both online and static node embedding algorithms. Our methods SW and SO significantly outperformed static Node2Vec, LINE, and simple degree related baselines. The combination of SW and SO achieved superior performance.

Chapter 5

Vaccine skepticism detection by network embedding

5.1 Introduction

We compiled a data set to demonstrate the applicability of network embedding to vaccine skepticism, a controversial topic of long-past history that became more important than ever with the Covid-19 pandemic. Only a year after the first international cases were registered, multiple vaccines were developed and passed clinical testing. Besides the challenges of development, testing and logistics, another factor in the fight against the pandemic are people who are hesitant to get vaccinated, or even state that they will refuse any vaccine offered to them. In this study, we focus on two groups of people commonly referred to as a) pro-vaxxer, those who support vaccinating people b) vax-skeptic, those who question vaccine efficacy or the need for general vaccination against Covid-19. It is very difficult to tell exactly how many people share each of these views. It is even more challenging to understand all the reasoning why vax-skeptic opinions are getting more popular.

In this work, our intention was to develop techniques that are able to differentiate between provaxxer and vax-skeptic content efficiently. After multiple data preprocessing steps, we evaluated Twitter content and user interaction network classification by combining text classifiers with several node embedding and community detection models from an open-source Python library [128]. While several methods exist to embed by text content [140] as well as by network structure [128], we are aware of only a few results that combine the two [55, 159, 165]. Similar experiments [107] and a data set [99] were published very recently.

5.2 Data collection

From January 7 to August 7, we collected data that anyone can view on Twitter using the free Twitter API. By specifying the keywords “vaccine”, “vaccination”, “vaccinated”, “vaxxer”, “vaxxers”, “#CovidVaccine” and “covid denier”, we collected 54,427 seed tweets, each with at least 50 replies (recursively). To eliminate drift towards topics of general politics such as US parties, we excluded the keywords “Trump”, “Biden”, “republican”, “democrat”. We considered seed tweets only in English. For each seed tweet, we collected the corresponding replies as well to build a reply network between Twitter users. In total, we collected almost 9 million replies. In our experiments, we reduced the reply network to $579,159$ nodes and $4,156,502$ edges by dropping users with less than 3 connections.

We randomly annotated 9.35% of the seed tweets in our data set with four different labels: pro-vaxxer ($2,626$ tweets), irrelevant ($1,438$ tweets), vax-skeptic (681 tweets) and anti-vaxxer (344 tweets). We found that it is even hard for humans to differentiate between vax-skeptic and anti-vaxxer content thus we merged anti-vaxxers into the vax-skeptic category. Furthermore, we excluded tweets labeled as irrelevant to avoid overfitting for politics, celebrities or other special events reported by the mainstream media. This way, we prepared the data for binary classification that includes $2,626$ (71.93%) pro-vaxxer and $1,025$ (28.07%) vax-skeptic tweets. In Figure 5.1, we show the 50 most popular words for these categories. Child vaccination, side effects, and death case reports were some of the most popular topics for vax-skeptic users. Personal experience of the vaccination, daily administered vaccine status reports, and eligibility for different age groups were in the highlight for pro-vaxxers. A good separation in sentiment can also be observed for the two groups through emojis. In Figure 5.2, there are only two emojis (syringe, clapping hands) in the intersection for the 20 most popular emojis of pro-vaxxers and vax-skeptic users.

5.3 Results

We trained a binary classifier to predict the vaccine view (pro-vaxxer or vax-skeptic) for each tweet. For classification, we used the following three modalities with logistic regression:

1. **text:** $1,000$ dimensional TF-IDF vector of tweet text (cleaned emojis, hashtags included);

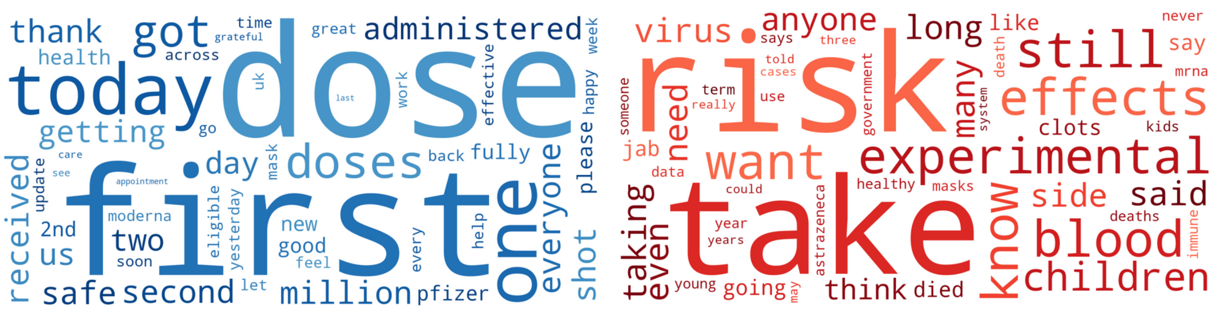
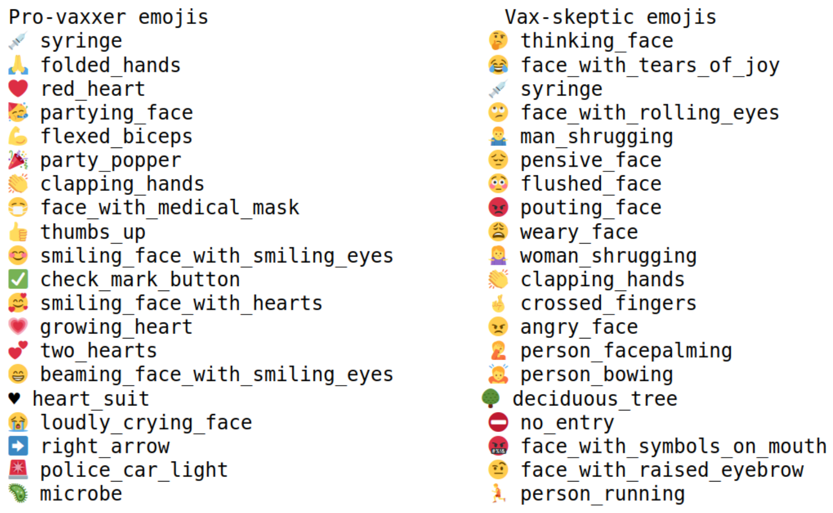


Fig. 5.1: Popular words for pro-vaxxer **(left)** and vax-skeptic **(right)** tweets.



Pro-vaxxer emojis	Vax-skeptic emojis
💉 syringe	🤔 thinking_face
🙏 folded_hands	😂 face_with_tears_of_joy
❤️ red_heart	💉 syringe
🥳 partying_face	🙄 face_with_rolling_eyes
💪 flexed_biceps	🤷‍♂️ man_shrugging
🎉 party_popper	😔 pensive_face
👏 clapping_hands	😳 flushed_face
😷 face_with_medical_mask	😡 pouting_face
👍 thumbs_up	😩 weary_face
😊 smiling_face_with_smiling_eyes	🤷‍♀️ woman_shrugging
✅ check_mark_button	👏 clapping_hands
🥰 smiling_face_with_hearts	🤞 crossed_fingers
💗 growing_heart	😠 angry_face
💕 two_hearts	🤦 person_facepalming
😁 beaming_face_with_smiling_eyes	🙇 person_bowing
♥ heart_suit	🌳 deciduous_tree
😭 loudly_crying_face	⛔ no_entry
➡️ right_arrow	🤬 face_with_symbols_on_mouth
🚨 police_car_light	🤨 face_with_raised_eyebrow
🦠 microbe	🏃 person_running

Fig. 5.2: Popular emojis for pro-vaxxer **(left)** and vax-skeptic **(right)** tweets.

2. **history:** Minimum, maximum, mean, and standard deviation of the past tweet labels of the same user;

3. **embedding:** 128-dimensional user representation obtained by Walklets [120] over the Twitter reply network.

We split the tweet data in time to 70% training and 30% testing. Our results are summarized in Table 5.1. Not surprisingly, user statistics have a strong contribution as users usually stick to their past opinion. User representations from the Twitter reply network improve performance, as seen in Figure 5.3. Indeed, tweets posted by users with no past label could be better inferred based on their social relations. Walklets [120], the best performing node embedding model in Figure 5.4, even managed to find pro-vaxxer and vax-skeptic user clusters, see Figure 5.5.

Next, we show that due to its multi-scale learning capability, Walklets could also reveal the topic hierarchy of vax-skeptic and pro-vaxxer users. For each word, we take the weighted average rep-

Table 5.1: Model performance with different feature components. Performance gain is shown with respect to the simple baseline using only textual information.

Feature components	AUC	gain (%)	Accuracy	gain (%)
text	0.8385	-	0.7559	-
text+history	0.8743	4.27	0.7769	2.78
text+embedding	0.9049	7.92	0.8427	11.48
text+embedding+history	0.9130	8.88	0.8473	12.09

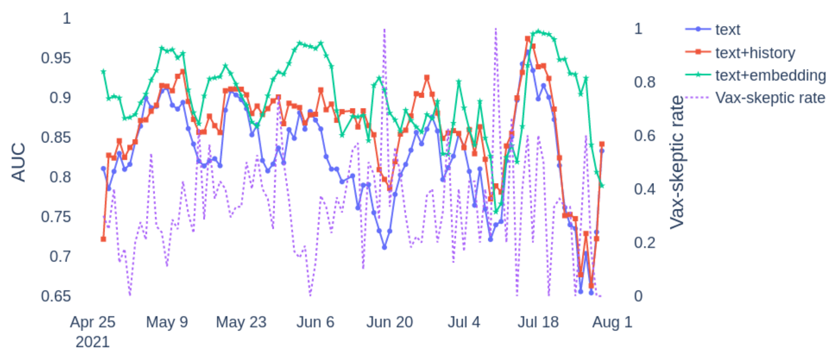


Fig. 5.3: Dynamic model performance based on a 7-day sliding window.

resentation of the users that mentioned it in their tweets. First, we analyze the vax-skeptic topic space. In the center of Figure 5.6, there are three anti-vaxxer topics related to child death, fear from the mRNA-based technology, and vaccine refusal induced by safety or free choice concerns. On the other hand, the periphery includes less offensive topics like politics, medical arguments, discussions related to AstraZeneca reactions, immunity doubts, and whether young and healthy people should generally vaccinate or not. As for pro-vaxxers in Figure 5.7, scientific news related to different levels of protection offered by Covid-19 vaccines are in the center. Two adjacent topics in the top-right region of the representation space are international news and the vaccination process in general, including eligibility for different age groups. Finally, in the bottom-left region, we can see personal vaccination reports sometimes related to friends and family. A few side effects and post-vaccination symptoms are closely related to these conversations.

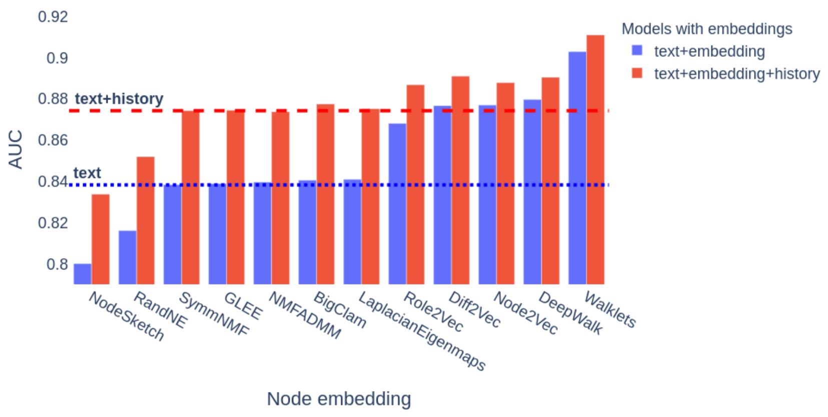


Fig. 5.4: Performance with respect to different node embedding and community detection models.

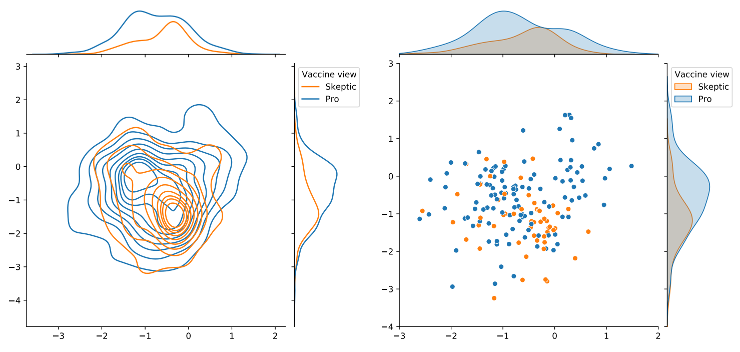


Fig. 5.5: Walklets clusters pro-vaxxer and vax-skeptic users well in the embedded space. **On the left** we show the kernel density estimation of the two groups for the whole test period, while **on the right** only active users between 5-13 May are visualized.

5.4 Conclusion

In this work, we quantitatively showed that social interactions play a major role in detecting vaccine skepticism. By deploying multiple node embedding models on a large Twitter reply network, we managed to discover pro-vaxxer and vax-skeptic communities. It was also interesting to see that Walklets successfully captured the topic hierarchy for different vaccine views on a very fine-grained level.

For reproducibility and future research purposes, we share our data on GitHub[1]. In order to comply with the data publication policy of Twitter, we only share the user ID, original and reply tweet IDs along with the encoded content vectors.

[1]https://github.com/ferencberes/covid-vaccine-network

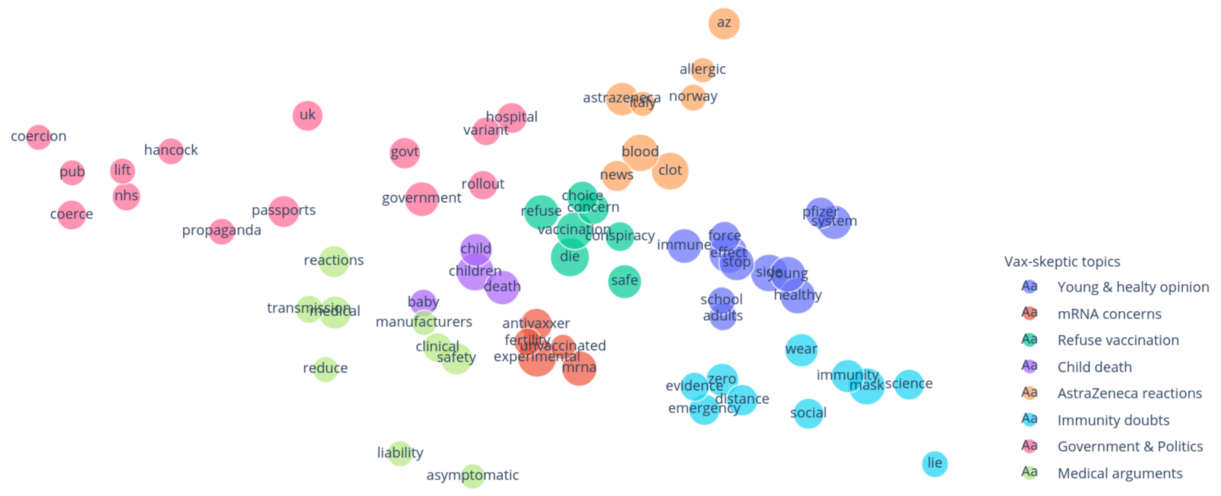


Fig. 5.6: Vax-skeptic topic space uncovered by Walklets embeddings.

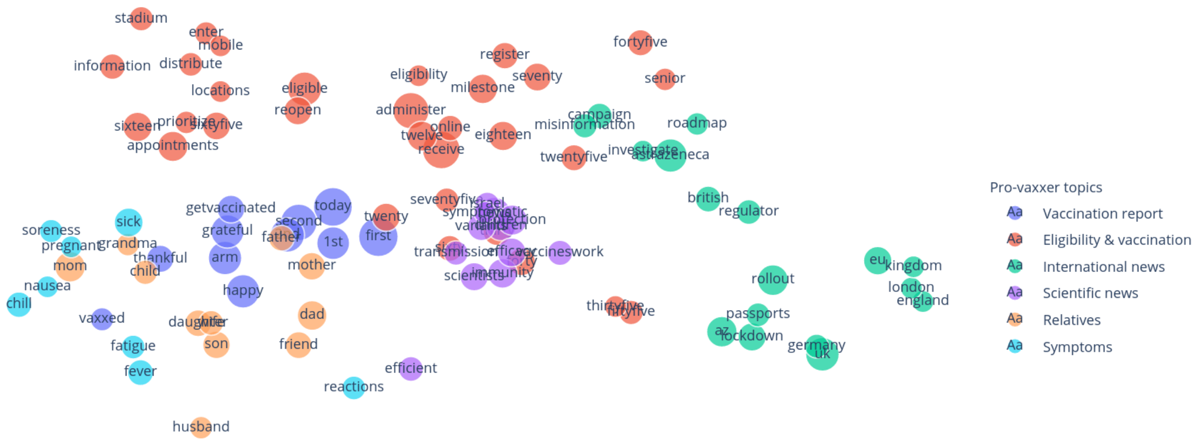


Fig. 5.7: Pro-vaxxer topic space uncovered by Walklets embeddings.

Chapter 6

Profiling and Deanonymizing Ethereum Users

The narrative around cryptocurrency privacy provisions has dramatically changed since the inception of Bitcoin [102]. Initially many, especially criminals, thought Bitcoin and other cryptocurrencies provide privacy to hide their illicit business activities [38]. The first extensive study about Bitcoin's privacy provisions was done by Meiklejohn et al. [92], in which they provide several powerful heuristics allowing one to cluster Bitcoin addresses. The revelation of Bitcoin's privacy shortcomings spurred the creation and implementation of many privacy-enhancing overlays for cryptocurrencies.

Ethereum is the largest public blockchain by usage. It is an account-based cryptocurrency where users store their assets in accounts rather than in unspent transaction outputs (UTXOs) as they do for Bitcoin. In an account-based cryptocurrency, native transactions can only move funds between a single sender and a single receiver, hence in a payment transaction, the change remains at the sender account. Thus, a subsequent transaction necessarily uses the same address again to spend the remaining change amount. Therefore, the account-based model essentially relies on address-reuse on the protocol level. This behavior practically renders account-based cryptocurrencies inferior to UTXO-based currencies from a privacy perspective.

In our privacy analysis of Ethereum's account-based model, we describe several patterns that characterize only a limited set of users and successfully apply these "quasi-identifiers" in address deanonymization tasks.

6.1 Our results

Prior to our work, we are aware of no empirical studies on account-based cryptocurrency privacy provisions. Therefore in this work, we put forth the problem of studying the privacy guarantees of Ethereum's account-based model.

In our first contribution, we identify and apply several quasi-identifiers stemming from address reuse, which allow us to profile and deanonymize Ethereum users.

In order to assess the performance of quasi-identifiers, we collected Ethereum related data from several sources, including Ethereum name service (ENS), Etherscan blockchain explorer, Tornado Cash mixer contracts, and Twitter. Using ENS identifiers as ground truth information, we quantitatively compare algorithms in a recent branch of machine learning, the so-called graph representation learning, as well as time-of-day activity and transaction fee based user profiling techniques. We are the first to quantitatively assess the performance of node embedding algorithms in the cryptocurrency domain.

Assessing and understanding the privacy guarantees of cryptocurrencies is essential as there are several companies or other entities, e.g. Chainalysis [106], performing large-scale deanonymization tasks on cryptocurrency users. In order to better showcase the loss of privacy caused by a quasi-identifier, we use three different performance metrics that we introduce in Section 6.5.

Finally, as an application, we explore the privacy guarantees of the Tornado Cash (TC) non-custodial mixer on Ethereum. In light of our results, we propose a few techniques that could potentially improve the privacy of the TC mixer.

We release the collected data (in anonymized form) as well as our source code for further research[1].

6.2 Background

In this section, we provide some background on Ethereum and a few related technologies and services.

[1]https://github.com/ferencberes/ethereum-privacy

6.2.1 Ethereum basics

Ethereum is a cryptocurrency built on top of a blockchain [157]. There are two types of accounts in Ethereum: externally owned accounts (EOAs) and contract accounts, also known as smart contracts. The global state of the system consists of the state of all different accounts. EOAs are controlled by an asymmetric cryptographic key pair, while smart contracts are controlled by their code stored in persistent, immutable storage. Contract code is executed in the Ethereum Virtual Machine (EVM). EOAs can issue transactions, which might alter the global state. Transactions can either create a new contract account or call existing accounts. Accounts have balances in ether (ETH), the native currency of Ethereum, and are denominated in wei where $1ETH = 10^{18}wei$.

A crucial aspect of the EVM is the **gas mechanism**. To every EVM opcode, there is a gas amount assigned, which is deemed to price the computational complexity of that opcode. Therefore, whenever one executes a smart contract code in the EVM, the execution consumes a certain amount of gas. At each transaction, the sender needs to define the maximum number of gas, called gas limit, they allow their transaction to consume. If a transaction does not consume all the gas assigned to it, then surplus gas is refunded to the caller, however, if a transaction runs out of gas, then all state changes are reverted and assigned gas is taken from the caller.

As of now, gas can only be purchased by Ethereum's native currency, ether, at a dynamically changing price, called gas price. Essentially, the gas price allows participants to prioritize their transactions as the transaction fee can be expressed as $txFee = gasPrice * gasLimit$. Therefore, miners are naturally incentivised to insert transactions with higher gas prices into their blocks to increase their collected transaction fees.

6.2.2 Ethereum Name Service

Ethereum Name Service (ENS) is a distributed, open, and extensible naming system based on the Ethereum blockchain. It is similar to the well-known Domain Name Service (DNS). However, in ENS, the registry is implemented in Ethereum smart contracts[2]. Hence, it is resistant to DoS attacks and data tampering. Like DNS, ENS operates on a system of dot-separated hierarchical names called domains, with the owner of a domain having full control over subdomains. ENS maps human-readable names like `alice.eth` to machine-readable identifiers, e.g., Ethereum addresses. Therefore, ENS provides a user-friendly way of moving assets on Ethereum, where users

[2]See: https://docs.ens.domains

can use ENS names (`alice.eth`) as recipient addresses instead of the error-prone hexadecimal Ethereum addresses.

6.2.3 Non-custodial mixers

Coin mixing is a prevalent technique to enhance the transaction privacy of cryptocurrency users. Coin mixers may be custodial or non-custodial. In case of custodial mixing, users send their "tainted" coins to a trusted party, who in return sends back "clean" coins after some timeout. This solution is not satisfactory as the users do not retain ownership of their coins during the course of mixing.

Motivated by these drawbacks, recently several non-custodial mixers have been proposed in the literature [91,135,137,156]. The recurring theme of non-custodial mixers is to replace the trusted mixing party with a publicly verifiable transparent smart contract or with secure multi-party computation (MPC). Non-custodial mixing is a two-step procedure, as illustrated in Figure 6.1. First, users deposit equal amounts of ether or other tokens into a mixer contract from an address $\mathcal{A}$. After some user-defined time interval, they can withdraw their deposited coins with a withdrawal transaction to a fresh address $\mathcal{B}$. In the withdrawal transaction, users can prove to the mixer contract that they deposited without revealing which deposit transaction was issued by them by using one of several available cryptographic techniques, including ring signatures [91], verifiable shuffles [135], threshold signatures [137], and zkSNARKs [156].

Cryptocurrency mixers typically provide k-anonymity to their users [132]. Generally speaking, a k-anonymized dataset has the property that each record is indistinguishable from at least $k-1$ others. Specifically, if a mixer contract holds n deposits out of which $n-k$ had already been withdrawn, then the next withdrawer will be indistinguishable among at least those k users who have not withdrawn from the mixer yet. Hence each withdrawer can enhance their transaction privacy and make their identity indistinguishable among at least k addresses. We call the set containing the k indistinguishable addresses the **anonymity set** of the user.

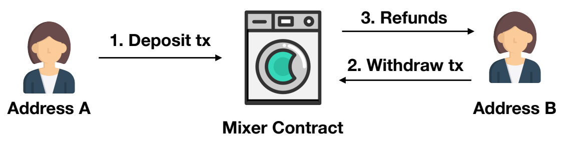


Fig. 6.1: Schematic depiction of non-custodial mixers on Ethereum

6.3 Review of related work

First results on Ethereum deanonymization [78] attempted to directly apply both on-chain and peer-to-peer (P2P) Bitcoin deanonymization techniques. The starting point of our work is the recognition that common deanonymization methods for Bitcoin *are not* applicable to Ethereum due to differences in Ethereum's P2P stack and account-based model.

The relevant body of more recent literature takes two different approaches. The first approach analyzes Ethereum smart contracts with unsupervised clustering techniques [110]. Kiffer et al. [75] assert a large degree of code reuse which might be problematic in the case of vulnerable and buggy contracts.

The second branch of literature assesses Ethereum addresses. A crude and initial analysis had been made by Payette et al., who clusters the Ethereum address space into only four different groups [117]. More interestingly, another work [148] proposes address clustering techniques based on participation in certain airdrops and ICOs. These techniques are indeed powerful. However, they do not generalize well as it assumes participation in certain on-chain events. Our techniques are more general and are applicable to all Ethereum addresses. Victor et al. gave a comprehensive measurement study of Ethereum's ERC-20 token networks, which further facilitates the deanonymization of ERC-20 token holders [149].

A completely different and unique approach is taken by [83], which uses stylometry to deanonymize smart contract authors and their respective accounts. The work had been used to identify scams on Ethereum.

We introduced node embedding algorithms as a class of network representation learning methods that map graph nodes to vectors in a low-dimensional vector space in Chapter 4. As feature mapping techniques are not limited to single-hop transaction neighbors, they have the potential to surpass intersection attacks [54]. For example, GraRep [35] can represent multi-hop neighborhood similarity, while Diff2Vec [129] can preserve distances between nodes by sampling diffusion trees during the training process. On the other hand, Role2Vec [3], a structural node embedding method assigns similar vectors to two addresses if they have similar motif structure in the transaction graph despite their distance in the network. To the best of our knowledge, we are the first to apply node embedding for Ethereum user profiling.

Table 6.1: Number of Ethereum accounts collected from three different sources. *In the Tornado Mixer, we consider account pairs identified by careless usage patterns as ground truth, see Section 6.7. Due to overlaps between the data sources, the total number of investigated addresses is less than the sum of the records in the top three rows.

Source	**Total**	**At least 5 sent txs**	**Used as ground truth pairs**
Twitter	1364	1260	129
Tornado Cash	2361	1618	*189
Humanity-Dao	695	602	n/a
All	4259	3321	318

6.4 Data collection

Our experiments were motivated by a Twitter movement that gained momentum in November 2019. Many ENS users posted their ENS name on their Twitter profile in order to facilitate transactions for those who want to interact with them on the Ethereum network. By discovering the Ethereum addresses linked to the published ENS names, we can quantitatively evaluate profiling techniques.

We collected addresses related to regular users and not automatic (trader or exchange) bots from the following publicly available data sources. **Twitter:** By using the Twitter API[3], we were able to collect 890 ENS names included in Twitter profile names or descriptions, and discover the connected Ethereum addresses, see Figure 6.2. **Humanity DAO:**[4] A human registry of Ethereum users, which can include a Twitter handle in addition to the Ethereum address. **Tornado Cash mixer contracts:** We collected all Ethereum addresses that issued or received transactions from Tornado Cash mixers up to 2020-04-04. Table 6.1 shows the total number of addresses collected from each data source as well as addresses with at least 5 sent transactions. We note that there are overlaps between the three address groups, see the last row of Table 6.1.

By using the Etherscan blockchain explorer API, we collected 1,155,188 transactions sent or

[3]Using the Twitter Search and People API endpoints, we collected tweets containing the following keywords {'@ensdomains','.eth','ENS name','ENS address', 'ethereum', '#ethereum'} as well as profiles with an ENS name in their displayed profile name or description. We also searched for ENS names in the name and description of every account in our data. Twitter data collection lasted from 2019-11-15 until 2020-03-05.

[4]See: https://www.humanitydao.org/humans

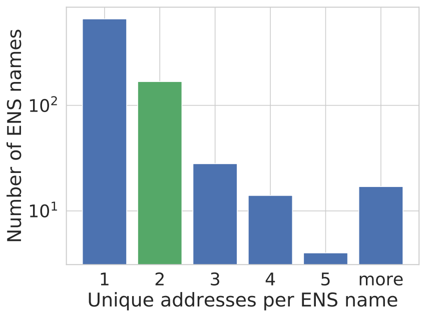


Fig. 6.2: Unique address count of ENS names collected from Twitter. Most of the ENS names in our collection are linked to a single Ethereum address, while some entities use multiple accounts. In Section 6.6, we use ENS names with exactly two unique addresses **(green)** to measure the performance of different profiling techniques.

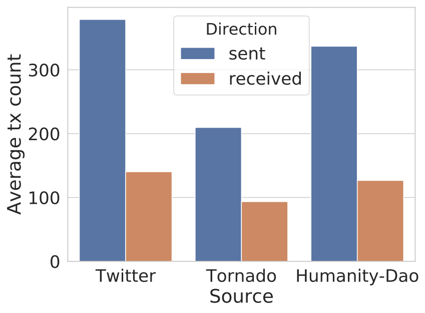


Fig. 6.3: Average number of transactions sent or received by the addresses of each data source. Tornado accounts have less transactions as the service has only recently been launched.

received by the 4259 regular user accounts in our collection (see Table 6.1). The final transaction graph contains 159,339 addresses, and the transactions span from 2015-07-30 until 2020-04-04. Figure 6.3 shows the average number of transactions sent and received only by the accounts in the three data sources. Addresses collected from Twitter and Humanity DAO have similar characteristics, while Tornado accounts have fewer transactions since Tornado Cash has only recently been launched at the time of our research.

Due to ethical considerations, we took several steps to anonymize our published dataset. For example, we transformed ENS names to hash values and we also removed links to real-world identities, e.g., Twitter accounts.

6.5 Evaluation measures

In this work, we propose deanonymization methods, i.e., pairing Etherum addresses of the same user (Section 6.6), Tornado deposits and withdrawals (Section 6.7). To establish an appropriate measure for evaluating our methods, we face the diversity and complexity of estimates of the adversary's success to breach privacy. In the literature, the adversary's output takes the form of a posterior probability distribution, see the survey [151].

The simplest metrics consider the success rate of a deanonymizing adversary. Metrics such as accuracy, coverage, the fraction of correctly identified nodes [103, 105] are applicable only when the attack has the potential to exactly identify a significant part of the network.

Exact identification is an overly ambitious goal in our experiments, which aim to use limited public information to rank candidate pairs and quantify the leaked information as the risk for a potential systematic deanonymization attack. For this reason, we quantify non-exact matches. Since even though our deanonymizing tools might not exactly find a mixing address, they can radically reduce the anonymity set, which is still harmful to privacy. We want to quantify the information leaked from network structure, time-of-day activity, and gas price usage to assess the implications for the *future privacy* [104] of the account owners.

In our deanonymization experiments, our algorithms will return a ranked list of candidate pairs for each account in our testing set. Based on the ranked list, we propose a simple metric, the **average rank** of the target in the output.

Recent results consider deanonymization as a classification task and use AUC for evaluation [86]. In our experiments, we will compute AUC by the following claim:

Lemma 6.5.1. Consider a set of accounts a, each with a set of candidate pairs $c(a)$ such that exactly one in $c(a)$ is the correct pair of a. Let an algorithm return a ranked list of all sets $c(a)$. The AUC of this algorithm is equal to the average of $r(a)/|c(a)|$ over all a, where $r(a)$ is the rank of the correct pair of a in the output.

Proof. Follows since AUC is the probability that a randomly selected correct record pair is ranked higher than another incorrect one [62]. □

Finally, we consider evaluation by variants of entropy, which quantify privacy loss by the number of bits of additional information needed to identify a node. Defining entropy is difficult in our case for two reasons. First, our algorithms provide a ranked list and not a probability distribution. Second, for the Tornado Cash mixer deanonymization, the anonymity set size is dynamic, as users can freely deposit anytime they wish, hence, increasing the anonymity set size.

In the literature, entropy based evaluation considers the a priori knowledge without a deanonymization method and the a posteriori knowledge after applying one [136]. Several papers compute the entropy of the a posteriori knowledge [44,104,136]. However, they assume that the deanonymizer outputs a probability distribution of the candidate records [104].

The information the attacker has learned with the attack can be expressed as the difference of the a priori and a posteriori entropy. We call this difference the **entropy gain**, denoted as $\text{gain}(n, p)$ where n and p are the anonymity set size and probability distribution, respectively. The a priori entropy of the target record is typically the base-2 logarithm of the a priori anonymity set size. The problem with varying a priori anonymity set size is that while correctly selecting ten candidate users from a pool of a million is a great achievement, the same entropy of $\log_2(10)$ is achieved without deanonymization if the initial pool size, for example in a low-utilization mixer, is only 10. We note that in [44], the authors also divide the entropy gain to normalize the value.

Next, we describe a new method to infer the a posteriori distribution given varying a priori knowledge and appropriately normalize with respect to the a priori entropy. More precisely, first we give a heuristic argument that the a priori anonymity set size has little effect on the entropy gain, and hence we can compare and average across different measurements. In the formula below, given an a priori anonymity set size $2n$ vs. n, we compare the entropy gain of the same distribution p, $\text{gain}(2n, p) - \text{gain}(n, p)$. In the formula below, p_i denotes the probability $p([(i-1)/(2n), i/(2n)])$ for $i = 1, \cdots, 2n$.

$$\begin{aligned}
\text{gain}(2n, p) &= \log_2(2n) + \sum_{i=1}^{2n} p_i \log_2(p_i); \\
\text{gain}(n, p) &= \log_2(n) + \sum_{j=1}^{n} (p_{2j-1} + p_{2j}) \log_2(p_{2j-1} + p_{2j}).
\end{aligned}$$

Since $\log_2(2n) - \log_2(n) = 1 = \sum_{i=1}^{2n} p_i = \sum_{j=1}^{n}(p_{2j-1} + p_{2j})$, we may group the terms to obtain the difference in the entropy gain as the sum for $1 \leq j \leq n$ of

$$\begin{aligned}(p_{2j-1}+p_{2j}) + p_{2j-1}\log_2(p_{2j-1}) + p_{2j}\log_2(p_{2j}) - (p_{2j-1}+p_{2j})\log_2(p_{2j-1}+p_{2j}) &= \\ p_{2j-1}\log_2(p_{2j-1}) + p_{2j}\log_2(p_{2j}) - (p_{2j-1}+p_{2j})\log_2\left(\frac{p_{2j-1}+p_{2j}}{2}\right) &= \\ p_{2j-1}\log_2\left(\frac{2p_{2j-1}}{p_{2j-1}+p_{2j}}\right) + p_{2j}\log_2\left(\frac{2p_{2j}}{p_{2j-1}+p_{2j}}\right)&,\end{aligned} \tag{6.1}$$

which can be bounded from above by using $\log x < x - 1$ as

$$\begin{aligned}p_{2j-1}\left(\frac{2p_{2j-1}}{p_{2j-1}+p_{2j}} - 1\right) + p_{2j}\left(\frac{2p_{2j}}{p_{2j-1}+p_{2j}} - 1\right) &= \\ \frac{2}{p_{2j-1}+p_{2j}}\left(p_{2j-1}^2 + p_{2j}^2\right) - (p_{2j-1}+p_{2j}) &= \\ \frac{(p_{2j-1}-p_{2j})^2}{p_{2j-1}+p_{2j}}&.\end{aligned} \tag{6.2}$$

If the probability distribution is smooth with little density changes in a neighborhood, the above value $\frac{(p_{2j-1}-p_{2j})^2}{p_{2j-1}+p_{2j}}$ is very small. For example, the value is small if p_i is monotonic in i, which at least approximately holds in our experiments.

Based on the above argument, we may infer an **empirical probability distribution** of the candidates ranked by an algorithm. For each a priori size n and rank r for the ground truth pair of a target record, we define the distribution $P(n, r)$ to be uniform in $[(r-1)/n, r/n]$, and 0 elsewhere, in accordance with formula (6.1). The empirical probability distribution of an algorithm will be the average of $P(n, r)$ over all the output of the algorithm. In the discussion, we will use the **entropy gain** of the above empirical probability distribution to quantify the deanonymization power of our algorithms.

6.6 Linking Ethereum accounts of the same user

In this section, we introduce our approach to identify pairs of Ethereum accounts that belong to the same user. In our measurements, we investigate three quasi-identifiers for each account: the time-of-day activity, the gas price selection as well as a low-dimensional vector space representation that represents the Ethereum transaction graph structure.

6.6.1 Ground truth data

We evaluate our methods by using the set of address pairs in our collection that belong to the same name in the Ethereum Name Service (ENS), see Figure 6.4. We consider 129 ENS names

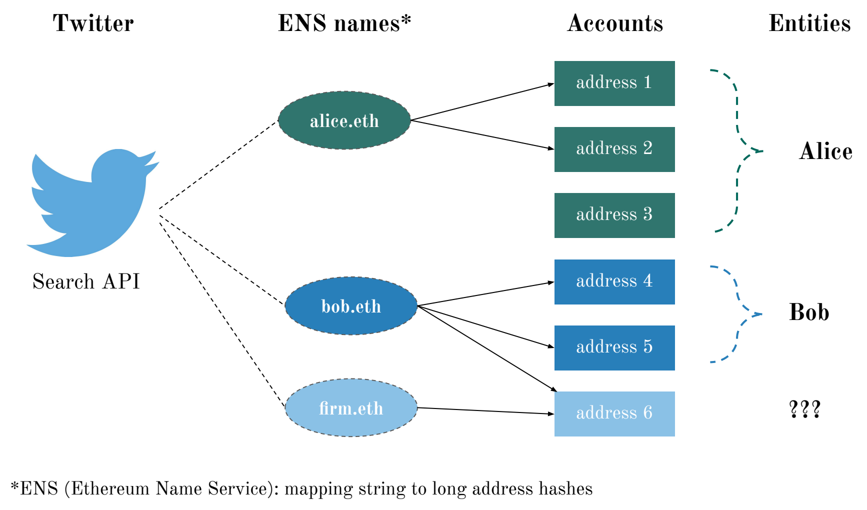


Fig. 6.4: We use the ENS names, collected from Twitter, as ground truth information in the Ethereum account linking tasks. For example, Address 1–2 and Address 4–5 belong to different entities, as suggested by the related ENS names.

with exactly two Ethereum addresses to avoid the possible validation bias caused by ENS names with more than two addresses, see Figure 6.2. We note that Ethereum addresses connected to multiple ENS names were excluded from our experiments.

6.6.2 Time-of-day transaction activity

Ethereum transaction timestamps reveal the daily activity patterns of the account owner. For example, Figure 6.5 shows that Ethereum users are not uniformly active in hours measured in GMT. Given the set of timestamps, an account is represented by the vector including the mean, median and standard deviation, as well as the time-of-day activity histogram divided into b_{hour} bins. In the top row of Figure 6.7, we show time-of-day profiles for two ENS names that are active in different time zones.

6.6.3 Gas price distribution

Ethereum transactions also contain the gas price, which is usually automatically set by the wallet software. Users rarely change this setting manually. Most wallet user interfaces offer three levels of gas prices, slow, average, and fast where the fast gas price guarantees almost immediate inclusion in the blockchain.

The changes in daily Ethereum traffic volume sometimes cause temporary network congestion,

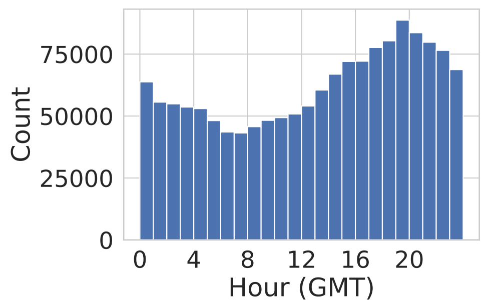


Fig. 6.5: Time-of-day distribution of Ethereum transactions

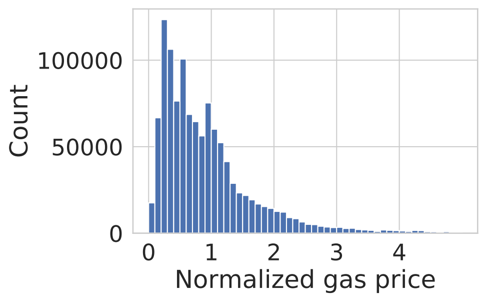


Fig. 6.6: Normalized gas price distribution of Ethereum transactions. Outliers above 5 are omitted.

which affects user gas prices. Hence, we normalized the gas price by the daily network average. In Figure 6.6, the two peaks of the normalized gas price around 0.5 and 1 correspond to the slow and average gas price options. On the other hand, users only occasionally charge more than three times the daily average gas price. The combination of the gas price levels forms the gas price profile for each Ethereum user. In the bottom row of Figure 6.7, we show normalized gas price profiles for two ENS names.

Given the normalized gas prices of the transactions sent, an address is represented by the vector including the mean, median and standard deviation, and the normalized gas price histogram divided into b_{gas} bins.

6.6.4 Graph representation learning

The set of addresses used in interactions characterize a user. Users with multiple accounts might interact with the same addresses or services from most of them. Furthermore, as users move funds between their personal addresses, they may unintentionally reveal their address clusters.

Our deanonymization experiments are conducted on a transaction graph with nodes as Ethereum addresses and edges as transactions. We deployed twelve node embedding methods from the library[5] of Rozenberczki et al. [128] to discover address pairs that might belong to the same user.

To compute the node embedding, certain graph preprocessing steps are required. First, we considered transactions as undirected edges and removed loops and multi-edges. We also excluded nodes outside the largest connected component. Due to the data collection techniques described

[5]https://github.com/benedekrozemberczki/karateclub

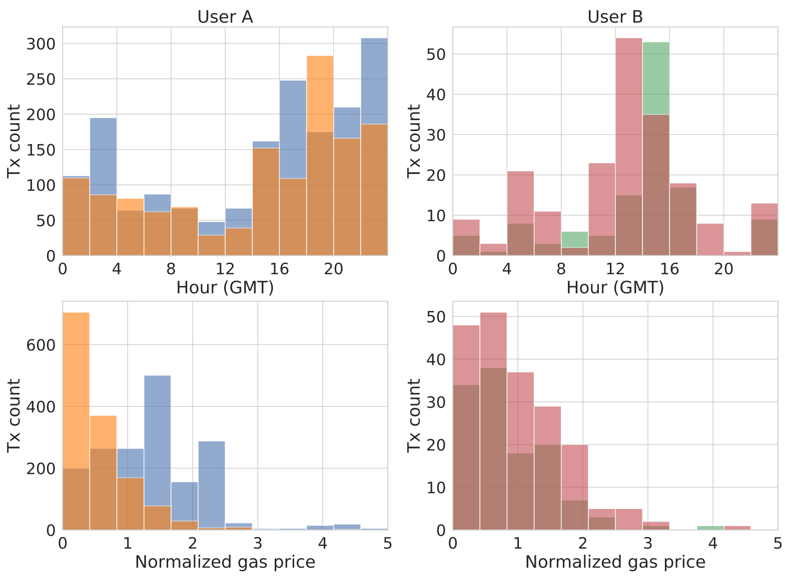


Fig. 6.7: Time-of-day (top row) and normalized gas price (bottom row) profiles for two ENS names (*User A*, *User B*) with a pair of addresses each. Both the time-of-day and gas price selection are similar in case of *User B* addresses (red, green) while the addresses of *User A* (blue, orange) have different gas price profiles. Addresses are denoted by different colors.

in Section 6.4, our transaction graph has two main components. The majority of the nodes have only one transaction, hence they reside on the periphery of the network, see Table 6.1. On the other hand, the core of the network contains the popular Ethereum services. Since the periphery adds little to the graph structure, we excluded nodes with degree one before training the node embedding models.

The resulting graph has 16,704 nodes and 132,231 edges. We generated 128-dimensional embeddings for the nodes (addresses), which is the standard setting for graph representation learning tasks [119, 142]. In order to compare with timestamp and gas price representations, we assign the overall average of the network embedding vectors to the removed nodes.

6.6.5 Evaluation

Based on timestamp, gas price distributions or network embedding, we generate Euclidean feature vectors for 3321 Ethereum addresses with each having at least five transactions sent, see Table 6.1. Given a target address, we order the remaining addresses by their Euclidean distance from the target with respect to their representations.

In the evaluation, we use 129 address pairs that belong to the same ENS name. The accuracy

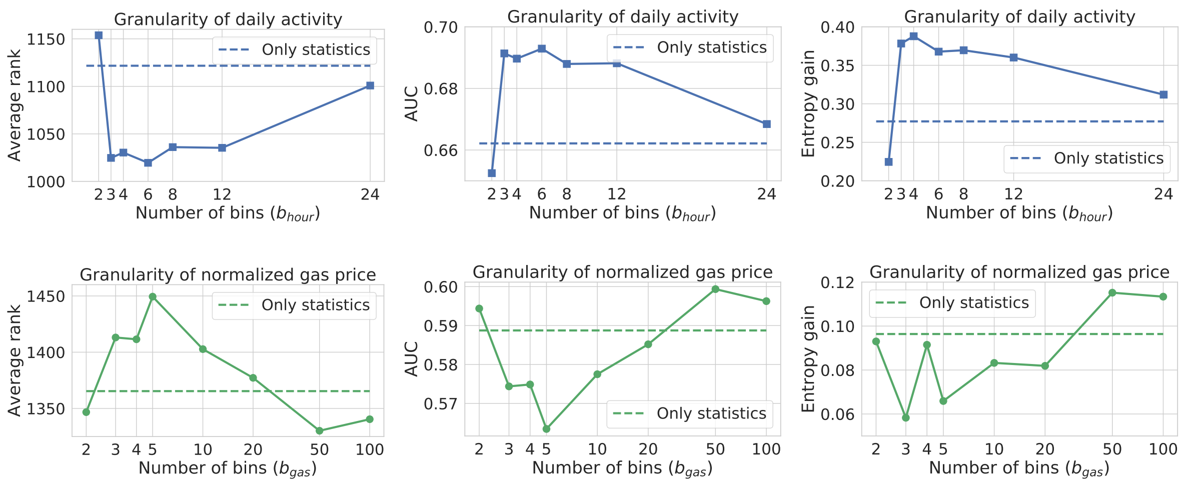


Fig. 6.8: Average rank, AUC and entropy gain at different granularity for daily activity textbf(top row) and normalized gas price **(bottom row)**. **Dashed lines** show performance with only mean, median and standard deviation used.

metrics of Section 6.5 for identifying accounts of the same user by using only time-of-day activity or normalized gas price is given in Figure 6.8. While time-of-day representation works best with $b_{\text{hour}} = 6$ (four-hour-long bins), normalized gas price representation performs weaker and the related histogram gives only a small improvement with $b_{\text{gas}} = 50$ over the case when the representation contains only the mean, median and standard deviation.

The average performance of the twelve different node embedding algorithms is shown in Figure 6.9 based on ten independent experiments. The two best performing methods are Diff2Vec [129] and Role2Vec [3]. Note that these algorithms capture different aspects of the same graph. Diff2Vec is a neighbourhood preserving model that performs strongly in community detection tasks [129]. In contrast, Role2Vec encodes the motif structure of the Ethereum addresses to successfully infer their function or role in the transaction graph. We achieved the best Ethereum address linking performance by the combination of Diff2Vec and Role2Vec, taking the harmonic average of the ranks for each account in the two candidate lists. Thus the combination of Diff2Vec and Role2Vec showed that different addresses of the same entity are usually in the same cluster or community performing similar roles.

In Figure 6.10, we show the fraction of pairs where the rank of the ground truth pair is not more than a given value. Surprisingly, Diff2Vec and Role2Vec find the corresponding ENS address pairs within 100 closest representations by almost 20% more likely than time-of-day activity and gas price statistics. Our combination based approach further improves the performance.

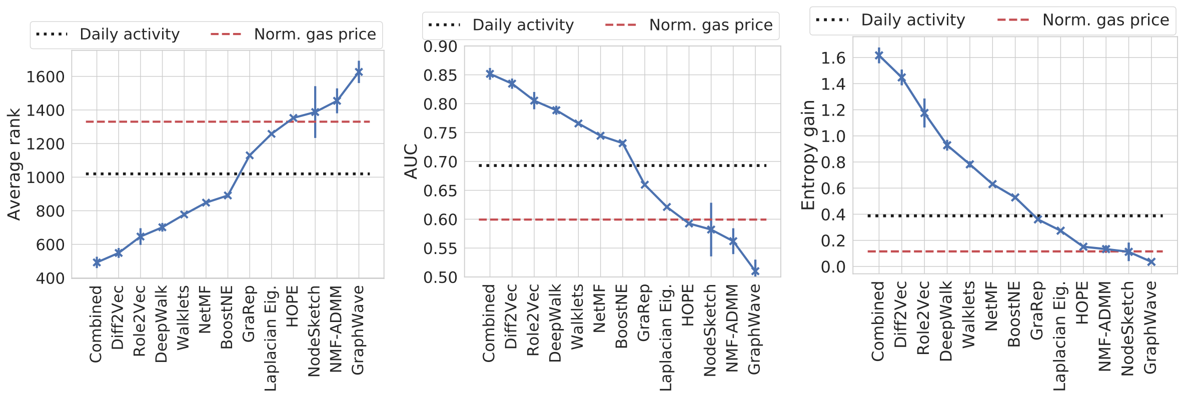


Fig. 6.9: Average rank, AUC and entropy gain for node embedding methods. Vertical lines show standard deviation in 10 independent experiments. Reciprocal rank combination of Diff2Vec and Role2Vec gives the best performance.

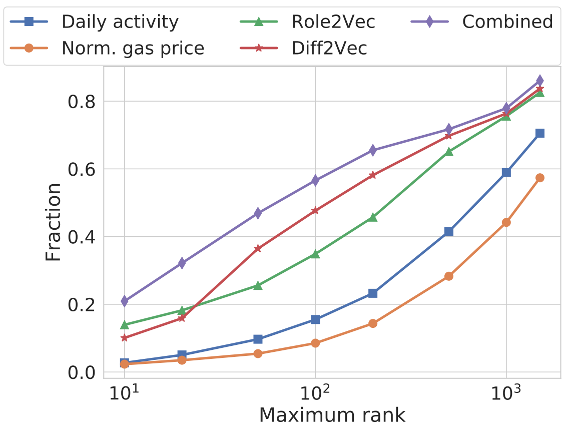


Fig. 6.10: Fraction of ENS address pairs correctly identified within a given maximum rank, for different embedding methods.

Our results show that the proposed profiling techniques link Ethereum addresses of the same user significantly better than random guessing. More precisely, the combination of Diff2Vec and Role2Vec yield 1.6 bits of additional information on account owners, see entropy gain results in Figure 6.9. In other words, we can reduce the anonymity set of a particular address by a factor of $2^{1.6} \approx 3.0314$.

In general, node embedding algorithms turn out to be more powerful for the Ethereum address deanonymization task than less elaborate time-of-day and gas price profiles. The latter techniques are only reliable if there are not many users in the same time zone or gas price profile. In contrast, graph representation learning can extract more complex descriptors that identify entities better.

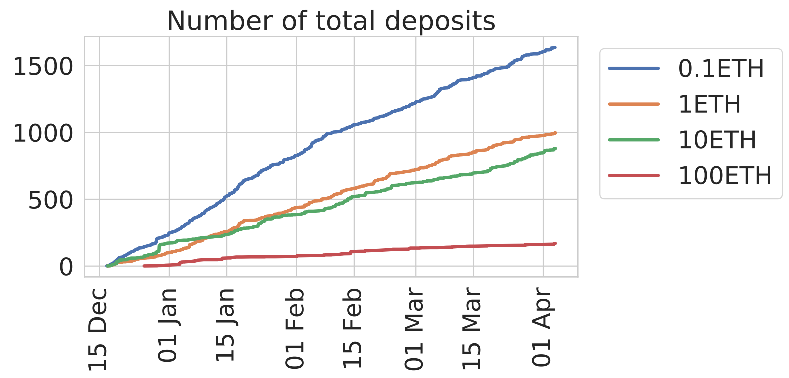


Fig. 6.11: The number of total deposits in each TC mixer from December 2019 to April 2020. This is an upper bound for the achievable anonymity set size when a withdraw transaction is executed. The popularity of the 0.1ETH mixer is superior compared to higher value mixers.

6.7 Deanonymizing trustless mixing services

Privacy-enhancing tools became crucially important gadgets in the Ethereum ecosystem. The most popular is Tornado Cash (TC), a non-custodial zkSNARK-based mixer. The TC mixers are sets of Ethereum smart contracts allowing users to enhance their anonymity. Each TC mixer contract holds equal amounts of funds from a set of depositors. In Figure 6.11, we show the changes in the anonymity set size over time for four TC mixer contracts (0.1 ETH, 1 ETH, 10 ETH, 100 ETH) respectively. Since TC was launched in December 2019, hundreds of deposits were placed in the mixers as more and more user interacted with this service. In general, we observe orders of magnitude lower activity for the 100ETH mixer, thus it does not provide as much anonymity as mixers with lower values (0.1ETH, 1ETH, 10ETH).

Unfortunately, careless usage easily reveals links between deposits and withdrawals and also impact the anonymity of other users, since if a deposit can be linked to a withdraw, it can be excluded from the anonymity set of other withdraws. The simplest careless usage is applying the same address for deposit and withdrawal transactions as well:

Pattern 1. *If there is an address from where a deposit and also a withdrawal has been made, then we consider these deposits and withdrawals linked.*

6.7.1 Ground truth data

Next, we define two more patterns we use for defining ground truth data for our machine learning methods that find linked pairs of accounts. Note that Pattern 1 finds a deposit-withdraw pair of the same address rather than an address pair. Hence, the anonymity guarantees are already broken without the need of machine learning methods.

The next pattern consists of the use of a salient gas price, which can be used to define linked address pairs in a ground truth set to evaluate our machine learning methods. Most wallet software, e.g. Metamask or My Ether Wallet automatically sets gas prices as multiples of Gwei (10^9 wei, i.e. Giga wei). However, one can observe gas prices whose last 9 digits are non-zero, hence those gas prices are likely set by the transaction issuer manually. These custom-set gas prices can be used to link deposits and withdraw transactions. For instance, one might observe the deposit transaction[6] at block height $9,418,956$ with 5.130909091 Gwei gas price. Later on, there is a withdraw transaction[7] at block height $9,419,096$ with exactly the same custom-set gas price.

Pattern 2. *If there is a deposit-withdraw pair with* unique gas prices*, then we consider them as linked.*

In a third pattern, users reveal links between their deposit and withdraw addresses if they sent transactions from one of their addresses to another address owned by them. We conjecture that users falsely expect that withdraw addresses are clean. Therefore, they send transactions from any address to their clean withdraw addresses. However, if the withdraw address can be linked to one of their deposit addresses, then they effectively lose all privacy guarantee accomplished by the fresh withdraw address. Express differently, if users run out of clean funds at their fresh addresses, they might feel tempted to move "dirty" assets to their "clean" addresses. Again, such a transaction links "clean" and "dirty" addresses as follows.

Pattern 3. Let d be a deposit and w a withdraw address in a TC mixer. If there is a transaction between d and w (or vice versa), we consider the addresses linked.

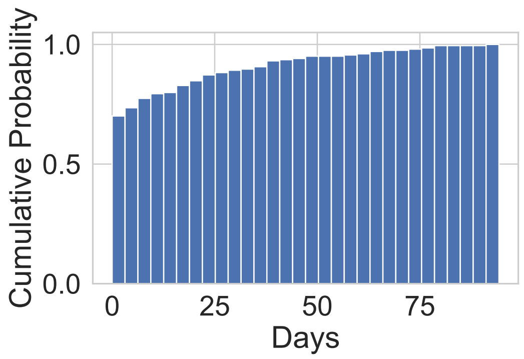
We found 218, 110, 60, and 7 withdrawals linked by Patterns 1–3 in the four mixer contracts (0.1 ETH, 1 ETH, 10 ETH, 100 ETH), respectively, up to April 4th 2020, see Table 6.2. We note that withdrawals identified by Pattern 2 can overlap with other withdrawals identified by Pattern 1 or 3. Hence the number of total linked withdrawals are less than the sum of all withdrawals individually identified by each pattern.

[6]Depositor: *0x074a3e9451fe3fb47be47786cf2dc4e84e797a6f*

[7]Withdrawer: *0x0f2437ff38e032596f2226873038230dcb22c485*

Table 6.2: Number of all withdrawals and deanonymized withdrawals using the corresponding patterns in each mixer contract.

	Deanonymized withdrawals				**All**
Mixer	**Pattern 1**	**Pattern 2**	**Pattern 3**	**Total**	**withdrawals**
0.1ETH	95 (7.5%)	80 (6.2%)	113 (8.8%)	218 (17.1%)	1272
1ETH	21 (2.5%)	40 (4.8%)	75 (9%)	110 (13.2%)	833
10ETH	8 (1.1%)	9 (1.2%)	46 (6.2%)	60 (8.1%)	738
100ETH	2 (1.5%)	5 (3.8%)	3 (2.3%)	7 (5.3%)	132



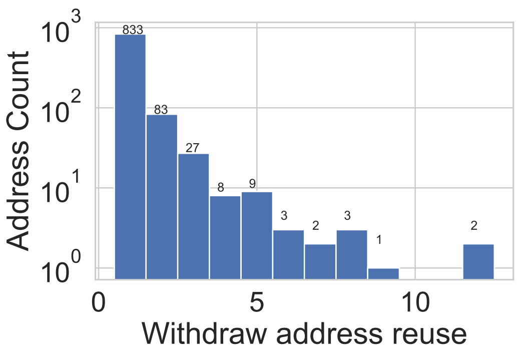


Fig. 6.12: Elapsed time in days between linked deposit and withdraw transactions for the 0.1 ETH mixer contract. Vast majority of users do not wait more than one day to withdraw their deposits.

Fig. 6.13: Withdrawal address reuse in the 0.1 ETH mixer contract. Many users withdraw multiple deposits to the same address, which eases deanonymization and reduces the privacy properties of the mixer.

6.7.2 Elapsed time between deposits and withdrawals, withdraw address reuse

In Figure 6.12, we observe that most users of the linked deposit-withdraw pairs leave their deposit for less than a day in the mixer contract. This user behavior can be exploited for deanonymization by assuming that the vast majority of the deposits are always withdrawn after one or two days.

Even worse, in Figure 6.13, we observe several addresses receiving multiple withdrawals from the 0.1 ETH mixer contract. For instance, there are 83 addresses that have withdrawn twice and 27 addresses with 3 withdrawals each. This phenomenon causes privacy risk not just for the owner of these addresses but also reduces the privacy properties of the mixer. Proper usage always requires a withdraw to a fresh address.

Table 6.3: Withdraw linking performance for the 0.1ETH mixer contract. Entropy gain and the average rank of the deposit address in the candidate list of our algorithms are shown for the three different ground truth sets (past, week, day), cf. Section 6.7.3.

Evaluation metric:	Average rank			Entropy gain		
Withdraw within:	past	week	day	past	week	day
Norm. gas price	168.784483	28.224719	5.861111	0.145438	0.134766	0.095546
Daily activity	116.146552	18.213483	3.444444	0.526595	0.570671	0.560121
Diff2Vec	89.222414	17.968539	4.268056	0.899869	0.743705	0.491611
Concatenated	64.012069	13.241573	3.362500	1.238389	1.151222	0.840526
Avg. anonymity set size	400.672414	70.247191	12.291667			

6.7.3 Deanonymization performance

Next, we measure how well the techniques of Section 6.6 identify the linked withdraw-deposit address pairs. We build ground truth by using careless Tornado usage Patterns 2–3. We define three different **ground truth sets**, one when the deposit is within the past day of the withdraw, another when within the past week, and the unfiltered full set. For example, for the 0.1 ETH mixer contract, we found 100, 80 and 67 deposit-withdraw pairs for these ground truth sets, respectively. Experiments on the unfiltered full set are labelled *past* in Figures 6.14-6.15 and Table 6.3.

Note that in Pattern 2, we used gas prices. Hence, in this section, we include measurements for gas price based linking performance only as reference. Similarly, Pattern 3 relies on an edge (transaction) between the two addresses, hence we discard these edges (transactions) in the network analysis algorithms. As we see, gas price distribution performs weakly for finding the account pairs, despite that Pattern 2 is based on gas price. On the other hand, adding the edges between accounts identified by Patterns 3 would yield misleadingly strong performance, since the same information is used for defining the ground truth and for testing.

Table 6.3 shows that an address with withdraw within a day or week has a significantly smaller anonymity set size, on average, since we only search for the corresponding deposit in a smaller set. In the 0.1ETH mixer, the original average anonymity set size of 400 can be reduced to almost 12 by assuming (e.g., an adversary might obtain this information by any means as background knowledge) that the deposit occurred within one day of the withdraw.

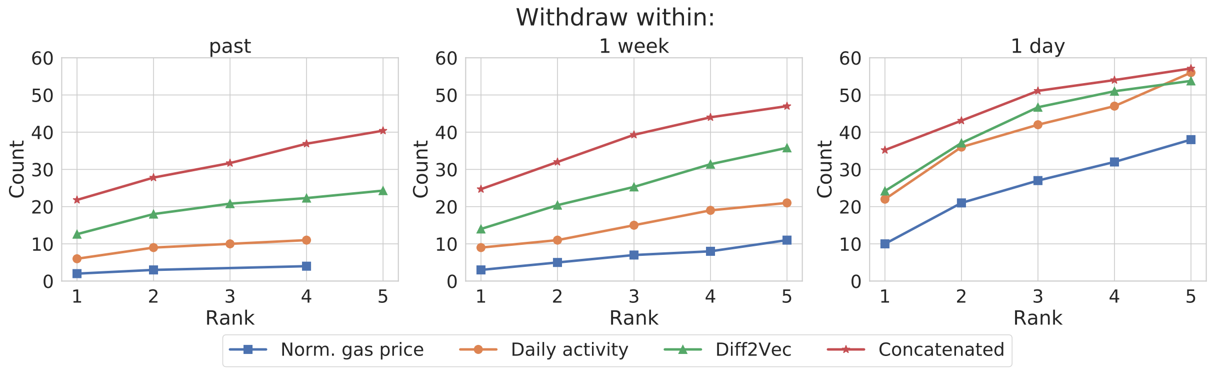


Fig. 6.14: Number of withdraw addresses in the 0.1ETH mixer contract such that the corresponding deposit is identified within the given rank in the candidate list of each deanonymization technique, separate for the three ground truth sets described in Section 6.7.3.

Daily activity and Diff2Vec perform much better in the TC withdraw linking task than gas price based user profiles. For the smaller week and day ground truth sets, they identify related deposit addresses within the 20 and 5 closest representations on average. Withdraw linking performance is further improved by concatenating the representations of daily activity and Diff2Vec. The number of withdrawals linked to deposits within a given rank of the candidate list for different methods are in Figure 6.14. The concatenated model identifies almost twice as many withdraw deposit pairs than Diff2Vec for the unfiltered ground truth set.

In Figure 6.15, we show the withdraw linking performance over time. As the number of active deposits increases, it becomes harder to link withdrawals to any of the past deposits. However withdrawals that follow the deposit after a few days are still much easier to deanonymize.

6.8 Maintaining privacy

In this section, we propose countermeasures to mitigate the effect of the main privacy leaks. First, we address problems related to non-custodial mixers, which is followed by a more general discussion on user behavior reflecting our results in Sections 6.6.5 and 6.7.3.

Randomized mixing intervals

Participants in mixer contracts greatly decrease anonymity if they withdraw funds after short time intervals, cf. Figure 6.12 and Table 6.3. A possible workaround is the use of randomized mixing intervals. Such a delay in withdrawal cannot be enforced by the mixing contract itself,

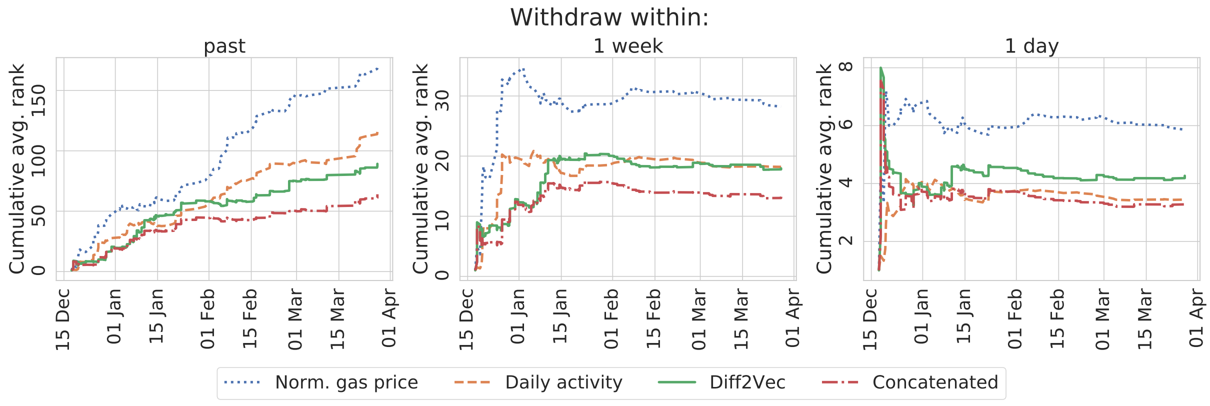


Fig. 6.15: Change of average rank in time, cumulated from the beginning of our data (December 2019), for the 0.1 ETH Tornado mixer by using our best deanonymization methods. Results are showed separately for the three ground truth sets described in Section 6.7.3.

since withdrawals cannot be linked to the deposits. Thus, delaying should be accomplished by the user wallet software.

Fresh withdraw addresses

Currently, many users apply the same withdraw addresses across several withdrawals, see Figure 6.13. This greatly decreases the complexity of linking deposits and withdrawals. Therefore, users must use fresh withdraw addresses for each of their withdrawals.

Mixer usage and user behaviors

Mixers mainly attempt to break the link between accounts associated with the same entity. As such, users need to ensure that their on-chain behaviors are unlinkable between uses of the TC mixers. Therefore, to ensure maximal privacy, users should use the TC mixers after every transaction. However, this decreases the user experience and ability to use applications on Ethereum.

Making deposits and withdrawals always during the same time of the day can also decrease the anonymity set for an account. An option in the user wallet software to schedule transactions for the future with random time delay could easily hide the time zone for the given entity.

Unfortunately, fooling complex graph representation learning algorithms is harder and requires constant monitoring of the whole Ethereum transaction graph, which the users themselves will unlikely be able to perform. For example, fooling Diff2Vec representations can require carefully chosen transactions to other address clusters. Similarly, several transactions are needed to change the motif profile to mislead Role2Vec. Companies or other large entities could probably act to obfuscate the network structure, but eventually, the end users will have to pay the price for the additional transactions. Finally, we note that using payments with negligibly small amounts to artificially link unrelated addresses are insufficient for obfuscation, as such transactions can be easily ignored in a privacy attack.

6.9 Conclusion

In this chapter, we studied how graph representation learning, time-of-day activity and gas price profile can be used to link Ethereum addresses owned by the same user. The Ethereum Name Service (ENS) relations in our data set provided ground truth information to quantitatively compare and analyze the performance of these quasi-identifiers. Our results showed that recent node embedding methods had superior performance compared to user activity based profiling techniques.

Our most interesting finding is that user activity for account-based cryptocurrency networks needs to be extremely well planned, otherwise the analysis of the transaction graph can easily reveal the account owner. Through the combination of Diff2Vec and Role2Vec, two recent node embedding models, we showed that different addresses of the same entity are usually in the same cluster or community performing similar roles.

As an application, we applied these profiling techniques on recently deployed privacy-enhancing overlays, such as Tornado Cash (TC) mixers. By our measurements, their decreased usability and immature user behavior prevent them from reaching their highest attainable privacy guarantees. Evaluation on different ground truth sets compiled from careless usage patterns of the TC mixer showed that profiling techniques, especially novel node embedding algorithms, can significantly reduce the anonymity set sizes of the mixing parties.

We release the collected data as well as our source code to facilitate further research[8].

[8] https://github.com/ferencberes/ethereum-privacy

Chapter 7

Cryptoeconomic traffic analysis of Bitcoin's Lightning network

Bitcoin is a peer-to-peer, decentralized cryptographic currency [101]. It is a censorship-resistant, permissionless, digital payment system. Anyone can join and leave the network whenever they would like to. Participants can issue payments, which are inserted into a distributed, replicated ledger called blockchain. Since there is no trusted central party to issue money and guard this financial system, payment validity is checked by all network participants. The necessity of full validation severely limits the scalability of decentralized cryptocurrencies: Bitcoin could theoretically process 27 transactions per second (tps) [51]; however, in practice its average transaction throughput is 7 tps [41]. This is in stark contrast with the throughput of mainstream payment providers; for example, in peak hours Visa is able to achieve 47,000 tps on its network [147].

To alleviate scalability issues, the cryptocurrency community is continuously inventing new protocols and technologies. A major line of research is focused on amending existing currencies without modifying the consensus layer by introducing a new layer, i.e., off-chain transactions [45, 89, 95]. These proposals are called Layer-2 protocols: they allow parties to exchange transactions locally, without broadcasting them to the blockchain network, updating a local balance sheet instead and only utilizing the blockchain as a recourse for disputes. For an exhaustive review of off-chain protocols, refer to [59].

Among these proposals, the most prominent ones are payment channel networks (PCN), in which nodes have several open payment channels, being able to connect to all nodes, possibly through multiple hops. The most popular instantiation of a PCN is Bitcoin's Lightning Network (LN) [122], a public, permissionless PCN, which allows anyone to issue Bitcoin transactions without the need to wait for several blocks for payment confirmation and currently with transaction fees orders of magnitude lower than on-chain fees. LN is suitable for several application scenarios, for instance, micropayments or e-commerce, with the intent to make everyday Bitcoin usage more convenient and frictionless. LN's core value proposition is that Bitcoin users can send low-value payments instantly in a privacy-preserving manner with negligible fees, which has led to quite a widespread adoption of LN among Bitcoin users.

The main difficulty with analyzing how LN operates is that the exact transaction routes are cryptographically hidden from eavesdroppers due to onion routing [71]. LN can only be observed through public information on nodes and channel openings, closings, and capacity changes. The actual amount of Bitcoins circulated in LN is unknown, although in blog posts, some node owners publish high-level statistics, such as their revenue [22, 84], which can be used as grounds for estimation.

7.1 Our results

To analyze LN efficiency and profitability, we designed a **traffic simulator** for LN to analyze the routing costs and potential revenue at different nodes. We assigned roles to nodes by collecting external data[1], labeling nodes as wallet services, shops, and other merchants. Using node labels, we simulated the flow of Bitcoin transactions from ordinary users towards merchants over time, based on the natural assumption that transactions are routed through the path that charges the minimum total transaction fee. By taking the dynamically changing transaction fees of the LN nodes into account, we designed a method to predict the optimal fee pricing policy for individual nodes in case of the cheapest path routing.

To the best of our knowledge, there has been no previous **empirical study on LN transaction fees**. Our traffic simulator hence opens the possibility for addressing questions of transaction routes, amounts, fees, and other measures otherwise depending upon strictly private information, based solely on the observable network structure. By releasing the source code of our tool[2], we allow node owners to fit various parameters to their private observation on LN traffic.

[1] Source: https://1ml.com

[2] See: https://github.com/ferencberes/LNTrafficSimulator

In particular, in this work the simulator enables us to draw two major conclusions:

Economic incentives. Currently, LN provides little to no financial incentive for payment routing. Low routing fees do not sufficiently compensate the routing nodes that essentially hold the network together. Our results show that in general, transaction fees are underpriced, since for many possible payments there is no alternative path to execute the transaction. We also give estimates of how the current network and fee structure responds to increase in traffic and decrease in channel capacities, thus assessing the income potential in different strategies. We provide an open source tool for nodes to experimentally design their channels, capacities, and fees by incorporating all possible information that they privately infer from the traffic over their channels.

Privacy. We quantitatively analyze the privacy provisions of LN. Despite onion routing, we observe that strong statistical evidence can be gathered about the sender and receiver of LN payments, since a substantial portion of payments involve only a single routing intermediary, who can easily de-anonymize participants. We find that using deliberately suboptimal, longer routing paths can potentially restore privacy while only marginally increasing the cost of an average transaction, as it is partially already incorporated in other implementations of the Lightning protocol [58].

7.2 Background

7.2.1 Payment channel networks

A **payment channel** allows users to make multiple cryptocurrency transactions without committing all of the transactions to the blockchain. In a typical payment channel, only two transactions are added to the blockchain, but theoretically, an unlimited number of payments can be made between the participants. Parties can open a payment channel by escrowing funds on the blockchain for subsequent use only between those two parties. The sum of the individual balances on the two sides of the channel is usually referred to as the **capacity**.

We illustrate the operation of a payment channel by an example. Let Alice and Bob escrow 1 and 2 tokens respectively, by committing a transaction to the blockchain that sets up a new channel. Once the channel is finalized, Alice and Bob can send escrowed funds back and forth by revoking the previous state of the channel and digitally signing the new state updated by the transacted tokens. For example, Alice can send 0.1 of her 1 token to Bob, so that the new channel state is

(Alice=0.9, Bob=2.1). Once the parties decide to close the channel, they can commit its final state through another blockchain transaction.

Maintaining a payment channel has an opportunity cost since users must lock up their funds while the channel is open, and funds are not redeemable until the channel is closed. Hence, it is not practical to expect users to maintain a channel with every individual with whom they may ever need to transact.

In a **payment channel network** (PCN), nodes have several open payment channels between each other; however, not necessarily with all other nodes. The network of bidirectional payment channels allows two parties to exchange funds even if they do not have a direct payment channel. For example, if Alice has a balance of 1 token with Ingrid, and Ingrid has a balance of 2 tokens with Bob locked in a payment channel, then Alice can route payments to Bob through Ingrid up to the maximum of the balances of Alice and Ingrid. Assuming that Alice sends 0.2 tokens to Bob, after routing we have the following channel balances: Alice=0.8, Ingrid=0.2 on the first channel and Ingrid=1.8, Bob=0.2 on the second channel.

In a payment channel, cryptographic protections are used to ensure that channel updates in both directions are executed atomically, i.e., either both or neither of them are performed [59]. In addition, incentive-based protections are also implemented to prevent users from stealing funds in a channel, e.g., by committing a revoked state. Similar techniques allow payment routing for longer paths. Furthermore, payment router intermediaries are financially motivated to relay payments as they are entitled to claim transaction fees after each successfully routed payment.

LN as a PCN consists of nodes representing users and undirected, weighted edges representing payment channels. Users can open and close bidirectional payment channels between each other and route payments through these connections. Therefore, LN can be modeled as an undirected, weighted multigraph since nodes can have multiple channels between each other. The weights on the edges correspond to the capacity of the payment channels.

In LN only capacities of payment channels are known publicly, individual balances are kept secret. This is because if individual balances are known, balance updates would reveal successful transactions, hence preventing transaction privacy.

7.2.2 Routing in LN and Fee Mechanism

LN applies source routing, meaning that it is always the sender who decides the payment route towards the intended recipient. Packets are onion routed, which means that intermediary nodes only know the identity of their immediate predecessor and successor in the route. Therefore, from a privacy perspective, nodes are incentivized to avoid single-intermediary paths, as in those cases intermediaries are potentially able to identify both the sender and the receiver.

LN provides financial incentives for intermediaries to route payments. In LN there are two types of fees that a sender pays to the intermediaries in case the transaction involves more than one payment channels. Nodes can set and charge the following fees after each routed payments:

Base fee: a fixed fee denoted as baseFee, charged each time a payment is routed through the channel.

Fee rate: a percentage fee denoted as feeRate, charged on the value txValue of the payment.

Therefore, the total transaction fee txFee to an intermediary can be obtained as:

$$\text{txFee} = \text{baseFee} + \text{feeRate} \cdot \text{txValue}. \tag{7.1}$$

We note that the base fee and fee rate is set by individual users, thus forming a fee market for payment routing. Furthermore, we remark that Equation 7.1 does not hold for all routing algorithms. However, we do not consider other fee structures in our simulator, as currently alternative routing algorithms are not widely adopted throughout the network.

7.3 Review of related work

To the best of our knowledge, we have conducted the first empirical analysis on LN transaction fees, similar to the way empirical and theoretical studies on on-chain transaction fees have been conducted during the early adoption of cryptocurrencies. Möser and Böhme conducted a longitudinal study on Bitcoin's nascent transaction fee market [98]. Kaskaloglu asserted that near-zero transaction fees cannot last long as block rewards diminish [70]. Easley et al. developed a game-theoretic model to explain the factors leading to the emergence of transactions fees, and provided empirical evidence on the model predictions [46]. Recently, BitMEX, a single LN node, has experimented with setting different transaction fees to measure the effect on routing revenue [22], which shows a similar pattern to our simulation experiments.

Unlike on-chain transactions, the LN transaction fee market is not yet consolidated. Some actors behave financially rationally, while the vast majority exhibit altruistic behavior, which parallels the early days of Bitcoin [98]. Similarly to on-chain fees, we expect to see more maturity and a similar evolution in the LN transaction fee market in the future.

Even before the launch of LN, many works studied the theoretical aspects of PCNs. Branzei et al. studied the impact of LN on Bitcoin transaction costs [26]. They conjectured a lower miner income from on-chain transaction fees as users tend to use and issue transactions on LN. In [74], the transaction fees of various payment channels are compared, however, without reference to the underlying network dynamics.

Depleted payment channels account for many efficiency issues in PCNs. Khalil and Gervais devised a handy algorithm to revive imbalanced payment channels without opening new ones [73].

PCNs can also be considered to be creation games. A user might decide to create a payment channel to a destination node or just route the payment in the already existing PCN. The former is more expensive; however, repeated payments can amortize the on-chain cost of opening a payment channel. Avarikioti et al. found that given a free routing fee policy, the star graph constitutes a Nash equilibrium [8]. In a similar game-theoretic work, the effect of routing fees was analyzed [7]. It was again found that the star graph is a near-optimal solution to the network design problem.

Even though transactions in LN are not recorded on the blockchain, they do not provide privacy guarantees. As early as 2016, Herrera et al. anticipated the privacy issues emerging in a PCN [64]. Single-intermediary payments do not provide privacy, although they have higher utility. Tang et al. asserts that a PCN either operates in a low-privacy or a low-utility regime [144]. Although a recently devised cryptographic protocol solves the privacy issues of single-intermediary routed payments [139], the protocol is not yet in use due to its complexity of implementation.

After the launch of LN, several studies have investigated the graph properties of LN [88,126,134]. They described the topology of LN at an arbitrarily chosen point in time and found that LN exhibits a hub and spoke topology, and its degree distribution can be well approximated with a scale-free distribution [126, 134]. Furthermore, these works assessed the robustness of the network against various types of attack strategies: they showed that LN is susceptible to both node [88, 134] and channel [126] removal based attacks. These works are restricted to a static snapshot of LN. The lack of temporal data has largely limited the insights and results of these contributions.

In a Youtube video [121], an estimate of the routing income is given based on the assumption that the payment probability between any node pair is the same. As it is easy to see, under this assumption the routing income of a node is proportional to its betweenness centrality. In our simulation experiments, we will explicitly compare our prediction with the one based on betweenness centrality and show how the finer structure of our estimation procedure yields more plausible results.

At the time of writing, four research groups published results on payment channel network simulators, each serving purposes very different from ours. Out of them, the simulator of Branzei et al. [26] is the only one that has pointers to publicly available resources. Their simulator only considers single bidirectional channels or a star topology, and its main goal is to analyze channel opening costs and depletion. This simulator is extended in [47] to generate and analyze Barabási-Albert graphs as underlying networks. CLoTH [40] is able to provide performance statistics (e.g., probability of payment failure on a given PCN graph); however, it does not analyze transaction fees, profitability, optimal fee policy, and privacy provisions of LN. In contrast, our LN traffic simulator can produce insights in those areas as well. Finally, the simulator in [161] is a distributed method to minimize the transaction fee of a payment path, subject to the timeliness and feasibility constraints for the success ratio and the average accepted value of the transactions.

7.4 Data collection

Throughout our work, we analyze two main data sources that are both available online[3]. Originally, these data sets contained only structural information related to LN that we augmented with node labels. Relying on the tags provided by the node owners[4], we distinguish between ordinary users and **merchants**. Merchants are assumed to receive payments more often than regular users. This is essential in understanding how popular payment channels are depleted throughout LN by repeated use in one direction.

We note that according to some estimates, 28% of all channels are private [125], meaning that their existence can only be recognized by the two ends. In our analysis, we have no information about private payment channels; however, the same holds for all the other network participants

[3]See: https://github.com/ferencberes/LNTrafficSimulator

[4]Source: https://1ml.com

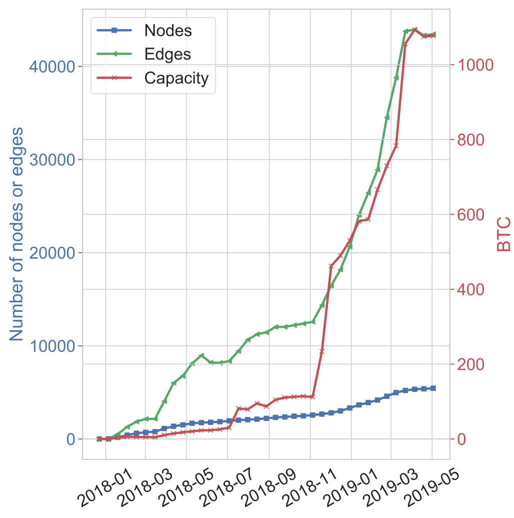


Fig. 7.1: LN's increasing popularity and adoption in its first 17 months.

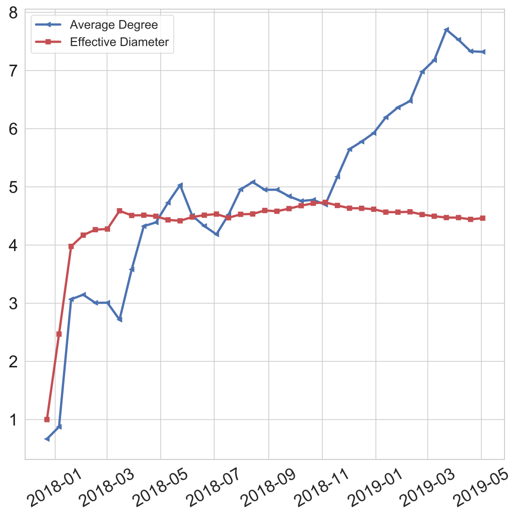


Fig. 7.2: Average degree and effective diameter in LN, as the function of time.

as well. Hence, we do not expect a significant bias in our results, as presumably those channels have private use and do not participate in carrying the global network traffic.

7.4.1 Dynamics of LN

Our first source is an edge stream data that we received from Antoine Le Calvez (Coin Metrics). It describes every payment channel opening and closure from block height 501,337 (in December 28, 2017) to 576,140 (in May 15, 2019). In this subsection, we summarize the main observations on the dynamics of LN that helped us to lay down the fundamental design choices for the LN traffic simulator that we describe in Section 7.5.

Ever since LN was launched, its popularity has grown steadily (Figure 7.1). This growth in popularity has caused the average degree increasing and the diameter decreasing over time, a "densification" phenomenon observed for a wide class of general networks in [81]. The average degree steadily increases, while the effective diameter decreases only after a first initial expansion phase (Figure 7.2).

LN payment channels adjacent to merchants have a shorter average lifetime (5198 blocks) than the average channel lifetime (5474 blocks), see the difference of the full distribution in Figure 7.4. We suspect that subsequent payments deplete the channels of the merchants, who then close these channels, collect their funds, and open new channels.

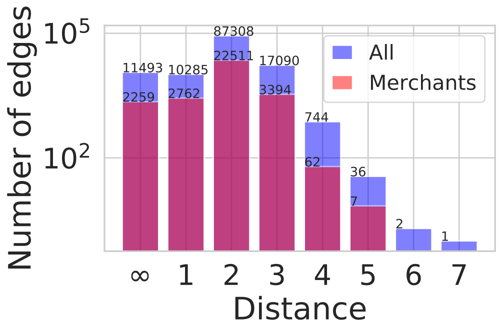


Fig. 7.3: The distance of LN nodes in the network at the time before a payment channel is established between them. ∞ denotes the case when they were in different connected components.

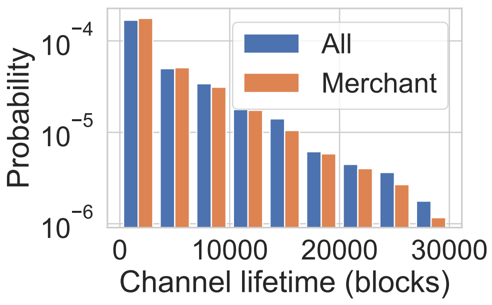


Fig. 7.4: Channel lifetime distribution of merchants and others (merchant average: 5198; overall average: 5474).

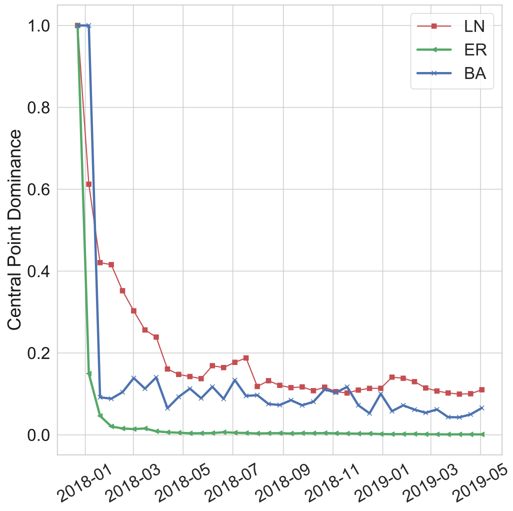


Fig. 7.5: Central Point Dominance of LN as the function of time, compared to that of an Erdős-Rényi (ER) and a Barabási-Albert (BA) graph of equal size at the given time.

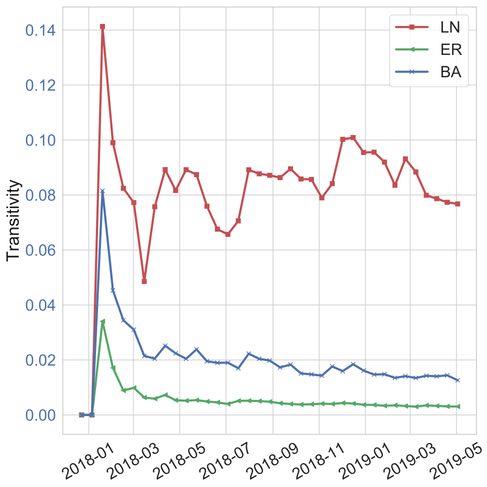


Fig. 7.6: Transitivity of LN, compared to that of an Erdős-Rényi (ER) and a Barabási-Albert (BA) graph of equal size at the given time.

We observe strong central point dominance in LN (Figure 7.5), which indicates that LN is more centralized than a Barabási-Albert or an Erdős-Rényi graph of equal size. This is in line with the predictions of [7, 8], affirming that PCNs lean to form a star graph like topology to achieve Nash equilibrium.

Counterintuitively, LN also exhibits high transitivity, also known as global clustering coefficient, see Figure 7.6. One would expect that nodes have no incentive to close triangles, as they might as well just route payments along already existing payment channels. However, we observe that the vast majority (68.76%) of all created payment channels connect nodes only 1 hop (distance 2) away from each other, see Figure 7.3. We believe that in most cases this is caused by replacing depleted payment channels. The high transitivity in LN is especially striking when it is compared to other social graphs. LN has roughly the same clustering coefficient as the YouTube social network [96].

7.4.2 LN snapshots with routing fees

We also collected snapshots of *the public graph* using the *lnd* client and utilized snapshots taken by Rohrer et al [126] as well.

In this section, we describe the graphs defined based on the 40 consecutive LN graph snapshots in our data set that span from 2019 February and March. The number of merchant nodes in the union of these snapshots is 169. We consider a minimum meaningful capacity $\alpha = 60\,000$ SAT (approximately USD 5 in 2019, at the time of writing our paper) and exclude edges with capacity less than α in G as they cannot be used in payments with value α.[5] Although LN channels are bidirectional, in our experiments we consider two directed edges, so that we can use channels in one direction if the capacity is exhausted in the other direction. We also ignore edges in the direction where they are flagged as disabled in the data. The properties of the LN network, averaged over the 40 daily snapshots, is as follows:

- Number of the union of all nodes: 4 787;
- Average number of nodes in a day: 3 358;

[5]Note that at the time of writing our paper, atomic multipath payments (AMPs) are not implemented. AMPs would allow one to split a payment value into multiple smaller amounts and subsequently send those payments to the receiver via multiple payment paths through different intermediaries. The AMP protocol will guarantee that either all sub-payments are executed or none of them.

- Non-isolated nodes after filtering disabled edge directions and edges with capacity less than 60 000 SAT: 3 132;
- Size of the largest strongly connected component: 2 206;

We highlight that it is only our snapshot data set that contains transaction fee information essential to simulate LN payments. Thus, the experiments in Sections 7.5-7.8 are based on the 40 consecutive LN graph snapshots only.

7.5 Lightning Network Traffic Simulator

In this section, we introduce our main contribution, the LN Traffic Simulator, which we designed for daily routing income and traffic estimation of network entities. Simulation is necessary to analyze the fine-grained structure, since the key concept of LN is privacy: data will never include transaction amounts, sources, and targets in any form, and it is very unlikely that it will give information on the capacity distribution over the channels, since that would leak information on the actual transactions. Hence we need a simulator to understand the capabilities and limitations of the network to route transactions.

By simulating transactions at different traffic volumes and transaction amounts, we shed light on the fee pricing policies of major router entities as well as on privacy considerations, as we will describe in Sections 7.6–7.8.

In our simulator, we make the assumption that the sender nodes always choose the cheapest route to execute their transactions. Due to the source routing nature of LN, nodes are expected to possess the knowledge of network structure and current transaction fees to make price optimal decisions. Note that in the LN client[6], the source node selects the routing for their transactions. For example, the sender node may choose the shortest instead of the cheapest path to the target if speed is more important than the transaction cost, and our simulator can be modified accordingly.

The main goal of our traffic simulator is to generate a certain number of transactions, given as an input parameter, by using only the information on the edges and their capacities in a given LN snapshot. To generate transaction sources and targets, we predefine the fraction of the transactions that lead to merchants based on the assumption that the majority of the transactions

[6]See https://github.com/lightningnetwork/lnd and https://github.com/ElementsProject/lightning.

correspond to money spent at shops and service providers. We fix the amount as constant to reduce the complexity of the simulation model.

We acknowledge that using constant payment amounts is a strong assumption. One could consider various distributions such as Pareto, power law, Poisson, as in previous works [144]. However, assumptions on the distributions as well as their parameter settings greatly increase the complexity of the experimentation, and cannot be empirically validated, since payment values are not public. We found the necessity to incorporate correlations of the amounts with node sizes and roles particularly troublesome. We note that constant amounts are also capable of capturing larger values by repeated payments from the same node. Finally, any time some entities obtain reliable estimates on the payment value distribution, they can conduct the corresponding experiments with our open source simulator.

Formally, we use the following notation:

- G, a daily graph snapshot of the LN with channels represented by pairs of edges in both directions; disabled directions and too low capacity edges are excluded;
- M, the set of merchant nodes defined in Section 7.4;
- τ, the number of random transactions to sample;
- α, the (constant) value of each transaction, in satoshis[7];
- ϵ, the ratio of merchants in the endpoints of the random transactions.

The available data only includes the total channel capacity but not its distribution between the endpoints. Thus, before simulation we randomly initialize the capacity between the channel endpoints. For example, if Γ is the total capacity of the channel between nodes u and v, we let $0 \leq \gamma(uv) \leq \Gamma$ and $0 \leq \gamma(vu) \leq \Gamma$ denote the maximum value in satoshis, which can be routed from u to v and vice versa. Both $\gamma(uv)$ and $\gamma(vu)$ change after each transaction that uses this channel while maintaining $\gamma(uv) + \gamma(vu) = \Gamma$ at all times.

If an edge has capacity less than α in a direction, that is $\gamma(uv) < \alpha$, the edge direction uv is *depleted*. In the simulation, a depleted edge uv cannot be used before a payment is made in the opposite direction vu, in which case $\gamma(uv) \geq \alpha$ will hold. Optionally, in Section 7.7, we

[7]Each Bitcoin (BTC) is divisible to the 8th decimal place, so each BTC can be split into 100,000,000 units. Each unit of Bitcoin, or 0.00000001 Bitcoin, is called a Satoshi. A satoshi is the smallest unit of Bitcoin, see https://satoshitobitcoin.co/.

will also investigate the effect of removing this constraint and allow the simulation to use an edge direction without limits. We also note that routers can balance payment channels without closing and reopening existing ones by finding cycles containing a depleted channel and route funds on a circular payment path [73], however, this option is not implemented in the current version of our simulator.

We start the simulation by first sampling τ transactions, each of amount α. First we select τ senders uniformly at random from all nodes. Recipients are selected by putting emphasis on merchants M: we choose $\epsilon \cdot \tau$ merchants with probability proportional to their degree in addition to $(1 - \epsilon) \cdot \tau$ recipients that are selected uniformly at random from all nodes including both merchants and non-merchants. Finally, we randomly match senders and recipients.

Given the transactions, we are ready to simulate traffic by finding the cheapest paths $P = (s = u_0, u_1, u_2, \dots, u_k = t)$ from sender s to recipient t with the capacity constraint $\gamma(u_i u_{i+1}) \geq \alpha$ for $i = 0 \dots k-1$. Then, node statistics (e.g., routing income, number of routed transactions) are updated for each intermediary node $\{u_1, u_2, \dots, u_{k-1}\}$ with respect to the latest transaction. Finally, for $i = 0 \dots k-1$ the value of $\gamma(u_i u_{i+1})$ is decreased while $\gamma(u_{i+1} u_i)$ is increased by the transaction amount α in order to keep available node capacities up to date. As we work with daily graph snapshots, the simulation mimics the daily traffic on LN.

The simulated routing income of a node will arise as the sum of the payment costs of its inbound channels. The cost of a payment can be obtained by substituting $\text{txValue} = \alpha$ in the transaction fee Equation (7.1), we obtain the transaction fee of an edge as $\text{baseFee} + \text{feeRate} \cdot \alpha$. We note that in this work we give no estimate on the cost of opening the channels, instead, we stop using depleted edges as long as a payment in the opposite direction reactivates them. We will assess the effect of channel depletion on routing income in Section 7.7, where we will allow the simulation to use an edge direction without capacity limits.

Due to several random factors in the simulation, including source and target sampling and capacity distribution initialization, we run the traffic simulator ten times. We use 40 consecutive daily snapshots in our data. We always report the mean node statistics (e.g., node routing income, daily traffic) of LN entities over our sets of 400 simulations for each parameter setting.

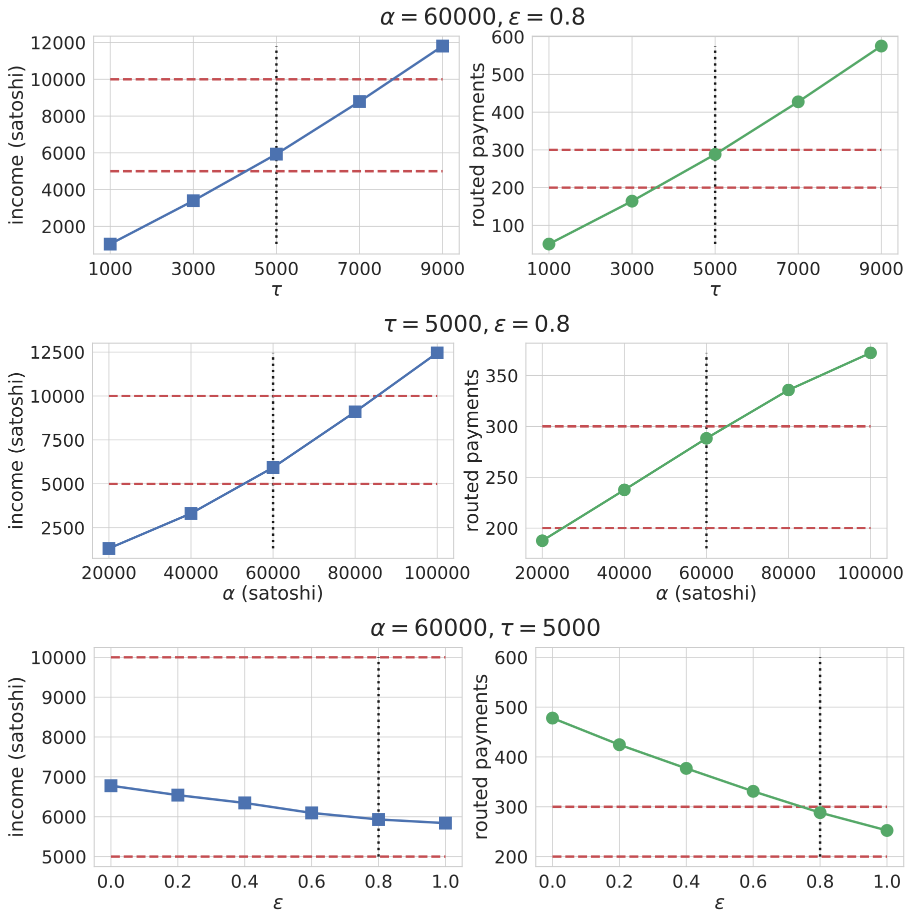


Fig. 7.7: Mean estimated routing income and number of routed payments of LNBIG.com entity with respect to traffic simulator parameters. The default parameter setting (daily transaction count $\tau = 5000$, single transaction amount $\alpha = 60,000$ satoshis, and merchant endpoint ratio $\epsilon = 0.8$) is marked by vertical black dotted lines. The daily income and traffic ranges stated by LNBIG.com [84] are marked by horizontal red dashed lines.

7.5.1 Feasibility Validation and Choice of Parameters

We validate our simulation model by comparing published information with our estimates for the income and traffic of the most relevant LN router entities. These nodes are responsible for keeping the network operational by routing most of the transactions. Our key source of information is the blog post [84] on LNBIG.com, the most relevant routing entity who owns several nodes on LN as well as approximately half of the total network capacity:

- In a typical day, LNBIG.com serves 200–300 transactions through all of its nodes, rarely exceeding 600 in a single day.

- On routing commissions, LNBIG.com earns $5,000$–$10,000$ satoshis per day.

We managed to reproduce daily traffic and routing income similar to LNBIG.com by sampling $\tau = 5,000$ transactions with $\alpha = 60,000$ satoshis (approximately 5 U.S. dollars) and merchant ratio $\epsilon = 0.8$. The estimated revenue, as the function of the parameters, is shown in Figure 7.7, also showing the target daily income and traffic ranges stated by LNBIG.com [84].

To summarize, simulating a few thousand micro-payments with mostly merchant recipients resulted in similar traffic and revenue as described over the nodes of LNBIG.com. We choose $\tau = 5,000$, $\alpha = 60,000$, and $\epsilon = 0.8$ as default parameters of our traffic simulator in order to draw some conclusions on LN node profitability and transaction privacy in Sections 7.6–7.8.

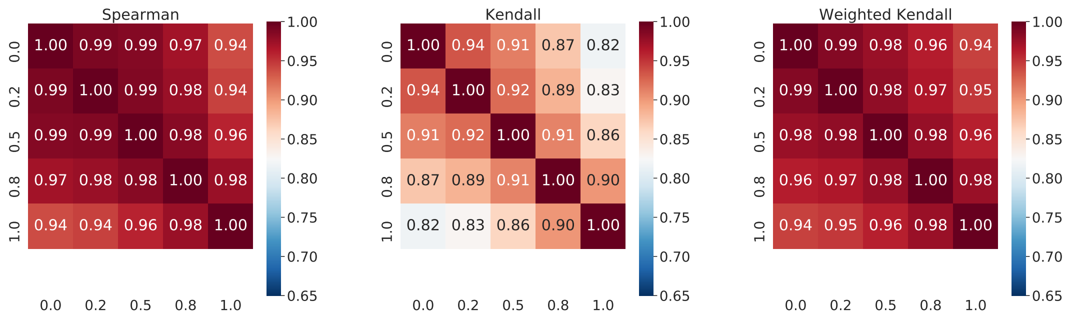
7.5.2 Traffic Simulator Response to Parameter Changes

Next, we examine the stability of our traffic simulator for different ratios of merchant endpoints ϵ. We note that the set of transaction recipients can be sampled uniformly at random by choosing $\epsilon = 0.0$, while in case $\epsilon = 1.0$, every sampled transaction has merchant endpoints. Thus, by increasing the value of ϵ the traffic can be centralized towards LN service providers. As determined in the previous subsection, we set the remaining parameters $\tau = 5,000$ and $\alpha = 60,000$.

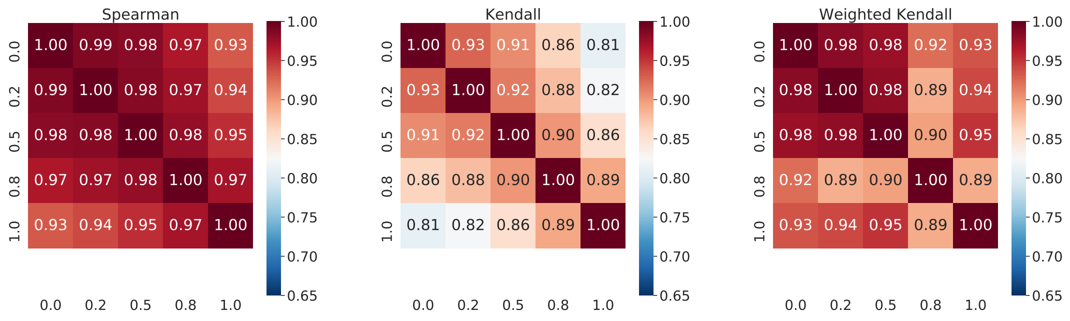
Our goal is to observe stable traffic characteristics throughout a sequence of days, measured as the correlation of node statistics across days. Towards this end, we measure the following node level summaries of the simulated traffic every day:

- *Routing traffic*: the number of transactions that are forwarded by a given node;
- *Routing income*: the sum of all transaction fees that a given node charges for payment routing;
- *Sender traffic*: the number of transactions that are initiated by a given node;
- *Sender fee*: the sum of all transaction fees that a given node has to pay for his transactions to be forwarded by intermediary nodes.

In Figure 7.8, the Spearman, Kendall, unweighted and weighted Kendall-tau correlations of routing traffic and income are shown for $\epsilon = 0.0$, 0.2, 0.5, 0.8, and 1.0. For the definitions, see [150].



(a) Correlation of routing traffic for $\epsilon \in \{0.0, 0.2, 0.5, 0.8, 1.0\}$.



(b) Correlation of routing income for $\epsilon \in \{0.0, 0.2, 0.5, 0.8, 1.0\}$.

Fig. 7.8: Correlation of simulated daily node routing traffic **(top three)** and income **(bottom three)** with respect to different ratio of merchants among transaction endpoints ϵ.

We observe high weighted Kendall-tau correlation, which means that the set of nodes with the highest routing income and traffic are very similar regardless of the ratio of merchants ϵ among transaction recipients.

By contrast, we observe low values of (unweighted) Kendall-tau. Since the set of nodes is dominated by low-traffic ones, the Kendall-tau value also depends mostly on the simulated traffic amount of these nodes. Hence, low Kendall-tau implies that nodes with low traffic and income fluctuate as transaction endpoints are selected at random. Most of these nodes have probably no traffic when transactions are centralized towards service providers ($\epsilon = 1.0$).

In Figure 7.9, we assess the stability of the simulation by showing the mean correlation of four different node statistics over 10 independent simulations for each snapshot. Two of the statistics, routing income and routing traffic, show high correlation for all values of ϵ, which means that nodes with high daily routing income and traffic are stable across independent experiments. By

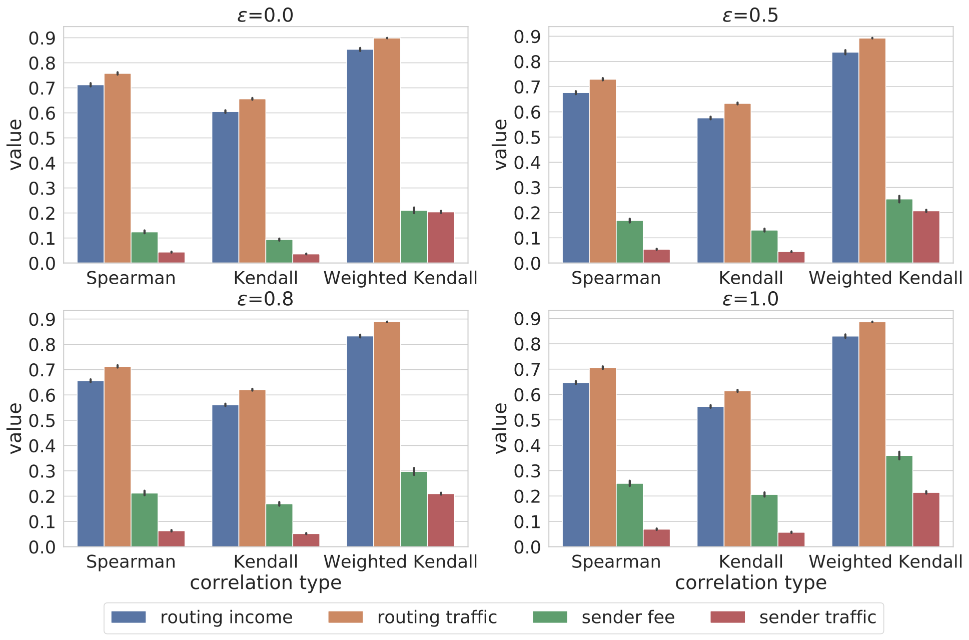


Fig. 7.9: Mean Spearman, unweighted and weighted Kendall-tau cross correlation of node statistics over the 10 independent simulations with respect to the ratio of merchants as transaction endpoints ($\epsilon \in \{0.0, 0.5, 0.8, 1.0\}$).

contrast, sender transaction fees and sender traffic especially vary highly, which is a natural consequence of uniform random sampling for source selection. By our measurements, ratio ϵ only affects the sender transaction fee. By increasing the value of ϵ, more and more transactions are centralized towards merchants. Thus, sender nodes pay the transaction fees to more or less the same set of intermediary nodes, which results in higher sender transaction fee correlations.

Finally, we compare our simulated routing income with simple estimates based on the properties of the nodes in LN as a graph. In a Youtube video, Pickhardt [121] shows the routing income of a node is proportional to its betweenness centrality in case the payment probability between any node pair is the same. In Figure 7.10, we observe that our simulated routing income with parameters $\alpha = 60,000$, $\tau = 5000$, $\epsilon \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ is well correlated with the betweenness centrality of a node. However, the Spearman correlation decreases with larger ϵ, which means that since payment endpoints are biased towards merchants, we need a more accurate estimation method. In Figure 7.11, we show two more node statistics, degree and total node capacity, both correlating much weaker to our prediction than betweenness centrality.

In summary, the set of nodes with high routing income and traffic are consistent across independent simulations regardless of the ratio of merchants among sampled transaction endpoints,

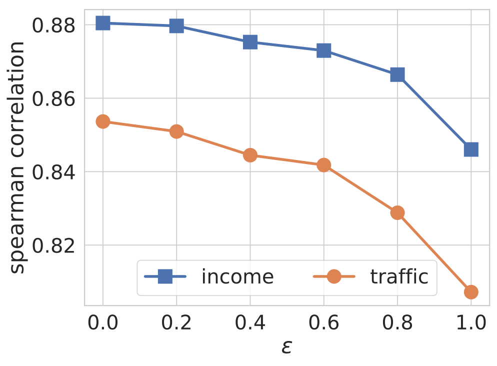


Fig. 7.10: Spearman correlation of predicted daily routing income (or traffic) and Betweeness centrality of LN nodes. The correlation decreases in case of high simulated merchant ratio ϵ.

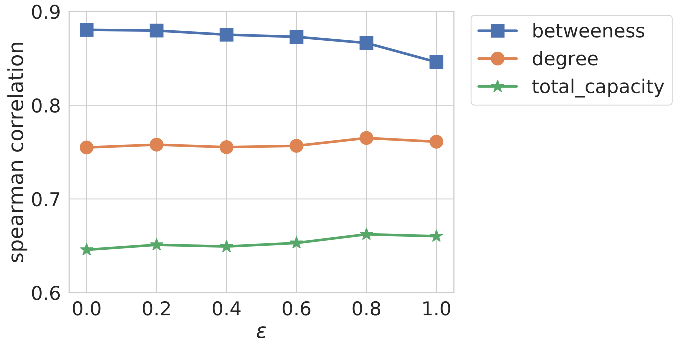


Fig. 7.11: Spearman correlation of predicted daily routing income and graph centrality measures with regard to the merchant ratio ϵ among payment end-points.

while randomization naturally has a big influence on the low traffic end of the network. The low traffic end can be estimated by incorporating the role of a node in the simulation, as we do in a very simple way by controlling traffic towards merchants with the parameter ϵ.

7.6 Transaction Fee Competition

Our first analysis addresses the observed and potential profitability of LN, which is questioned in several blog posts [22, 84]. A core value proposition of LN is that Bitcoin users can execute payments with negligible transaction fees. This feature may be cherished by payment initiators, but in case of insufficiently low network traffic, it could be unprofitable for router entities.

Our goal is to assess how transaction costs depend on topology and to what extent they are targets to competition. To measure transaction fee price competition, we use our traffic simulator to estimate daily node routing income and traffic volume for the 40 consecutive LN snapshots in our data. Our findings on how revenue from routing depend on transaction fees shows a similar shape as experimented for BitMEX, a single LN node [22].

We use the parameters of the simulator that we calibrated based on published information on the income of certain nodes [84] in Section 7.5.1. Our analysis in this section confirms that transaction fees are indeed very low, and they are potentially underpriced for relevant router

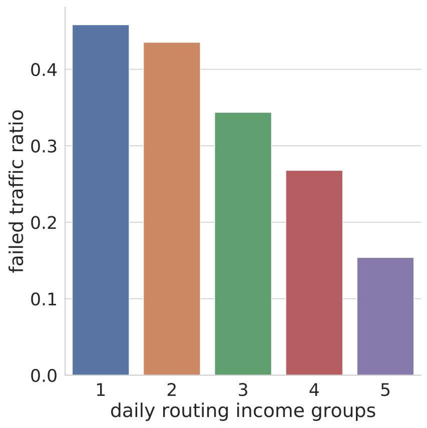


Fig. 7.12: The average failure ratio of individual node traffic for five income groups defined as the top $1-10$, $11-20$, $21-50$, $50-100$, and $101-$ router nodes with highest simulated income.

nodes.

To analyze the competition that a node x faces in the network, we compare the simulated traffic in a daily LN snapshot G and in the graph G_x that we obtain by removing node x from G. By attempting to route the same set of τ transactions on G and G_x, first of all we measure the number of failed payments $\varphi(x)$ that were originally routed through x but are incapable of reaching destination when x is out of service. For each node x, the failure ratio of individual node traffic is $\frac{\varphi(x)}{\tau(x)}$ where $\tau(x)$ denotes the number of transactions through x in the original simulation.

In Figure 7.12, we show the average ratio of the traffic of a node that has no alternate routing path, for five income groups defined as the top $1-10$, $11-20$, $21-50$, $50-100$, and $101-$ router nodes with highest simulated income. For each group, the average is taken over its nodes x, considering the fraction of transactions $\frac{\varphi(x)}{\tau(x)}$ that cannot be routed anymore after removing x. It is interesting to observe that for the first three groups, the average ratio of traffic with no alternate path is at least 0.3. This means that even if the 100 routers with highest simulated traffic increased their transaction fees close to on-chain fees, the majority of payment sources would have no less expensive option to route their payments.

In the next experiment, we estimate the extent transaction prices are potentially limited by the competition among alternate routes in LN. We take a highly pessimistic view by assuming that a transaction that can only be routed by relying on an intermediary node x will select a payment method outside LN immediately if x increases its transaction fees. For other transactions, we

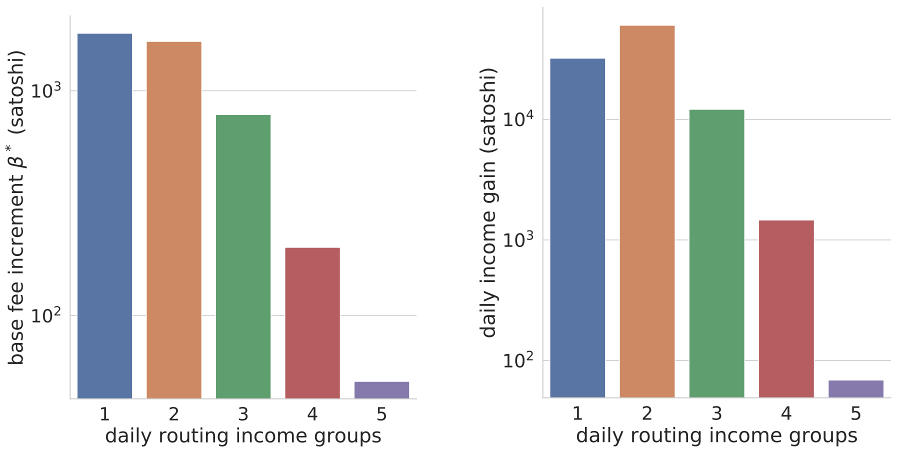


Fig. 7.13: The maximal possible base fee increment (β^*, **left**), and the corresponding income gain **(right)** in satoshis, given the price competition assumptions in Section 7.2.2. Income groups are defined as the top $1-10$, $11-20$, $21-50$, $50-100$, and $101-$ router nodes with highest simulated income.

search for the next cheapest route that avoids x and assume that x could increase its fees to match the second cheapest option. In other words, our analysis ignores the failed transactions $\varphi(x)$ and is based on the remaining $\tau(x) - \varphi(x)$ where payment routing avoiding node x being available. For each of these transactions, the difference of the total fee δ can be calculated from the fees of the original path in G and the alternative route in G_x.

Our assumption is that if node x increases its base fee by β, transactions with $\delta \geq \beta$ are still willing to pay for the additional costs, while for $\delta < \beta$, payments will be routed on the cheaper alternative path, where δ is the fee difference to the cheapest path avoiding x. Thus, by observing $\beta \geq 0$ at different thresholds, we propose an optimal β^* base fee increment for each router node.

We estimate the optimal fee increase β^* for each node over multiple snapshots and independent simulations. For the five node income groups that we previously defined in Figure 7.12, we show the average optimal base fee increment as well as the corresponding routing income gain in Figure 7.13.

The transaction fee data shows that the current LN fee market is still immature, as the majority of all channels apply default base fee (1 SAT) and fee rate (10^{-6} SAT), while the capacities are usually set higher than the default value (100000 SAT) in the *lnd* client, see Figure 7.14.

In our measurements, we find that nodes with high routing income could still increase their base

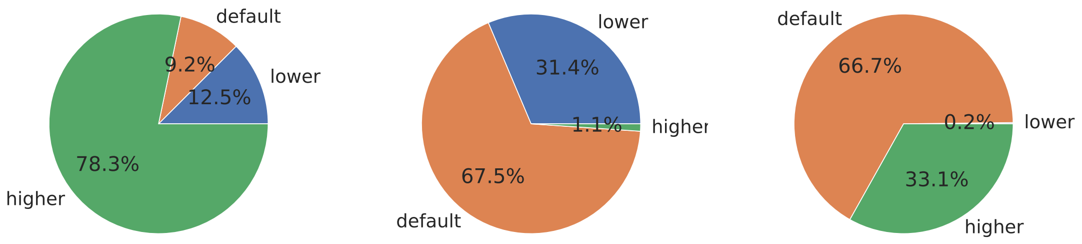


Fig. 7.14: Distribution of channel capacities **(left)**, base fees **(center)** and fee rates **(right)** with regard to their default values in the *lnd* client (100000 SAT, 1 SAT, and 10^{-6} SAT), respectively.

fee by a few hundred satoshis, thus generating an average gain of more than 10,000 satoshis in their daily income. Despite the low gain, our assumption is that it could get orders of magnitude higher if router nodes increased their base fee in succession, which could have a major impact on the competition for transaction costs.

7.7 Profitability Estimation of Central Routers

Router entities are an essential part of LN. They are responsible for keeping the network operational by forwarding payments. In this section, we estimate the current routing revenue of these central nodes, and give predictions how their income will change if the traffic over the current network increase. Note that our technique can also be used for node owners to predict the effect of opening and closing channels as well as changing capacities and transaction fees.

Central routing nodes are binding a huge amount of financial resources in the form of channel capacity, which enables them to serve high volumes of traffic. In general, router entities consist of a single node, but sometimes they have multiple LN nodes. For example, LNBIG.com owns 25 nodes in our dataset. One of our main motivations was to estimate the annual return of investment (RoI) for entities by simulating daily traffic over several snapshots. In our measurements we calculate annual RoI as follows:

$$RoI = \frac{\text{estimated daily routing income in satoshis} \times 365}{\text{total amount of satoshis bound by channel capacities}}. \tag{7.2}$$

By simulating traffic with parameters $\tau = 5,000$, $\alpha = 60,000$, and $\epsilon = 0.8$, we estimated the daily average income and traffic for each router. From these statistics and additional entity capacity data downloaded from 1ML.com, we estimate annual RoI in Table 7.1. We present all router entities with at least 50 satoshis of simulated income and 10 forwarded transactions per day on average. For each of these nodes, the following statistics are presented:

- *Entity capacity* as downloaded from 1ML.com. *Capacity fraction* is the fraction of entity capacity and total network capacity. Remarkably, half of the total network capacity is bound by the nodes of LNBIG.com.
- Average *transaction fee*, *daily income*, and *daily traffic*, based on the simulated mean cost in satoshis that a given entity charges for each payment routing over his channels during the observed 40 snapshots, in ten random simulations, as explained in Section 7.2.2.
- *Annual RoI* calculated from simulated daily income and entity capacity by Formula 7.2.
- *Economical fee* in satoshis is the amount required on average to reach an annual 5% RoI. *Fee ratio* is the ratio of the economical and the actual transaction fees. Higher values mean lower profitability.
- Three columns show the *rank* of the nodes in decreasing order of annual RoI, total fee, and traffic.

Based on our findings, the annual RoI is way below 5% for almost all relevant entities. Only rompert.com achieved a comparable amount of annual RoI (3.45%), who indeed applies orders of magnitude higher fees than others. It is interesting to see that despite its high transaction fees, it has the highest daily traffic in the simulation. Note that rompert.com applies base fees close to onchain fees, which may invalidate the assumptions of our simulator if participants fall back to onchain rather than paying rompert.com routing fees.

Compared to the most profitable node rompert.com, the total estimated traffic of LNBIG.com through its 25 nodes is only one third. The main reason behind low annual RoI is low transaction fees. Table 7.1 shows that for forwarding $\alpha = 60,000$ satoshis, most of these entities ask for less then 100 satoshis, which is less than 0.2% of the payment value. Very low fees may uphold LN's core value proposition, but they are economically irrational for the central routers holding the network together. Based on our simulations, for several routers (e.g., LNBIG.com, yalls.org, ln1.satoshilabs.com, etc.), fees should be in the range of a few thousand satoshis to reach a 5% annual RoI, which was approximately the magnitude of on-chain transaction fees (1,000-2,000

Table 7.1: Estimated daily income, traffic and annual RoI for relevant router entities. Columns are explained in Section 7.7. Note that on-chain transaction fees for a regular transaction (2 inputs, 2 outputs) was in the range of 1000-2000 satoshis at the time of writing our paper.

name	Capacity fraction (%)	Capacity (millions)	Fee for 60000SAT	Daily income	Daily traffic	Annual RoI (%)	Fee ratio	Economical fee (SAT)	RoI rank	Fee rank	Traffic rank
rompert.com	0.958	969	4371.9	91831.4	835.2	3.458924	1.4	6319.7	1	1	1
zigzag.io	0.926	950	601.0	6716.2	11.2	0.258036	19.4	11645.7	2	2	23
LNBIG.com	52.309	53686	32.4	5932.0	288.2	0.004033	1239.8	40181.0	17	8	4
yalls.org	1.728	1772	151.0	2001.2	13.3	0.041199	121.4	18325.8	4	4	22
ln1.satoshilabs.com	2.597	2665	60.0	1180.5	19.7	0.016167	309.3	18556.9	9	6	20
OpenNode	1.364	1400	87.5	825.6	29.6	0.021519	232.4	20335.2	6	5	17
tippin.me	1.054	1081	55.1	474.4	45.3	0.016009	312.3	17207.9	10	7	12
BlueWallet	1.523	1562	276.0	453.4	16.4	0.010588	472.2	130329.2	13	3	21
LightningPowerUsers.com	2.413	2440	1.4	368.7	305.1	0.005514	906.7	1278.1	16	19	3
ACINQ	3.367	3455	7.0	281.7	40.2	0.002975	1680.5	11763.3	18	13	14
btc.lnetwork.tokyo	0.240	245	2.8	224.0	107.6	0.033247	150.4	426.8	5	16	7
Sagittarius A	0.528	541	14.9	218.3	30.9	0.014715	339.8	5047.4	11	11	16
1ML.com node ALPHA	0.688	706	1.1	174.5	164.6	0.009021	554.3	587.5	15	20	5
tady je slushovo	0.413	423	1.1	123.2	116.3	0.010622	470.7	499.0	12	21	6
Electrophorus [W_C_B]	0.384	393	19.5	101.7	80.8	0.009425	530.5	10346.2	14	10	9
There be dragons here	0.172	176	25.5	87.5	7.1	0.018050	277.0	7068.2	7	9	25
LightningTo.Me	1.202	1233	0.4	78.3	725.5	0.002317	2158.3	919.1	20	25	2
fairly.cheap	1.365	1401	0.7	74.9	103.0	0.001950	2563.7	1863.8	24	24	8
Bitrefill.com	2.323	2383	4.4	73.2	31.4	0.001121	4460.8	19713.9	25	15	15
BOLTENING.club	0.898	921	6.7	54.7	43.3	0.002166	2308.8	15362.1	22	14	13
ORANGESQUIRREL	0.018	18	7.0	52.7	7.5	0.104882	47.7	333.8	3	12	24
lightning-roulette.com	0.728	747	1.1	49.6	46.8	0.002421	2065.7	2189.6	19	23	10
CoinGate	0.821	842	1.1	49.2	46.4	0.002129	2348.6	2489.5	23	22	11
ln.BitSoapBox.com	0.590	605	1.6	37.8	23.9	0.002276	2196.6	3503.3	21	18	18
Blockstream Store	0.076	78	1.6	37.1	23.2	0.017364	287.9	460.7	8	17	19

satoshis[8] at the time of writing our paper).

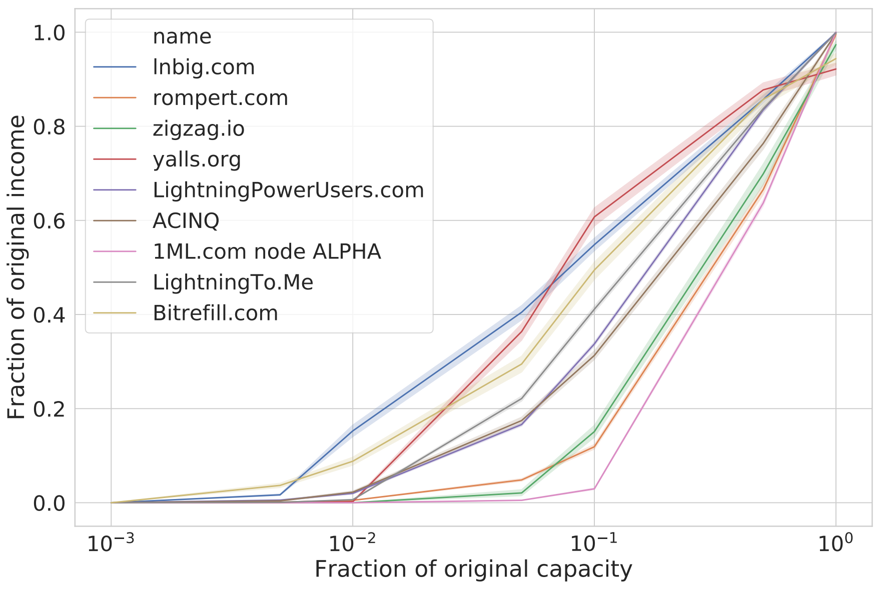


Fig. 7.15: The remaining fraction of the original estimated daily routing income, after reducing node capacities to the given fractions.

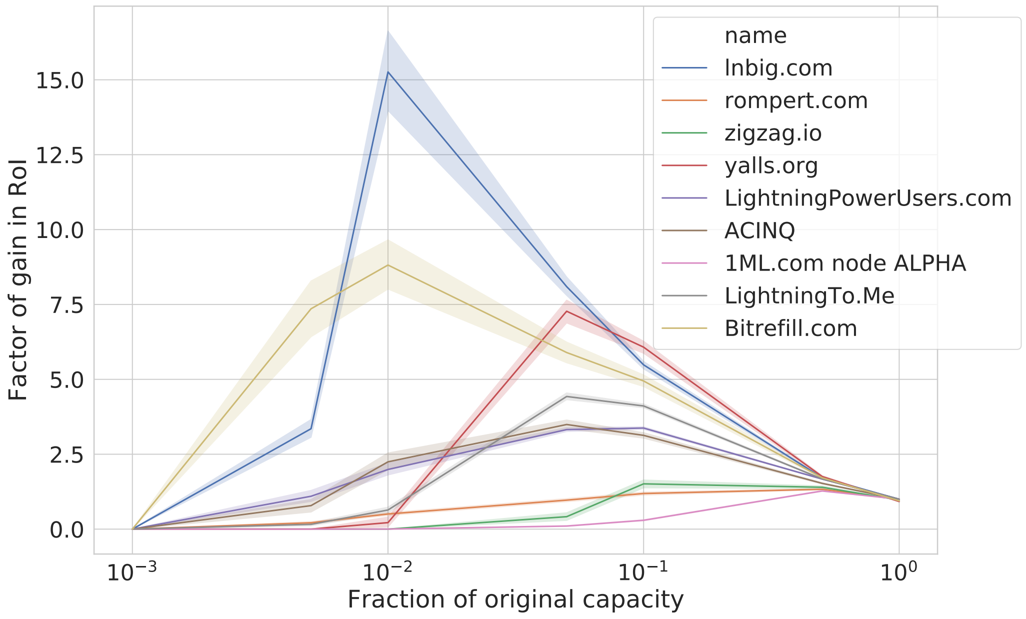


Fig. 7.16: RoI gain after reducing node capacities to the given fractions.

[8]See https://bitcoinfees.info/

Capacity overprovisioning also causes low RoI. For example, extremely large LNBIG.com capacities result in low RoI, despite the reasonable daily income reported. By using our traffic simulator, we observed that the router entities of Table 7.1 can increase their RoI by reducing their channel capacities. For each of these routers, we estimated the changes in revenue (Figure 7.15) and RoI (Figure 7.16), after reducing all of its edge capacities to $50, 10, 5, 1, 0.5, 0.1\%$ of the original value, with the assumption that all other routers keep their capacities. In our measurements, LNBIG.com can significantly improve its RoI by bounding only 1% of its original capacity values. In Table 7.3, we compute the estimated optimal RoI for the central routers.

To estimate whether routers can be more profitable with an increase in traffic volume or transaction values, we ran simulations with different values of τ and α and measured the fraction of unsuccessful payments as well as the average length of completed payment paths.

First we vary the transaction value α with a fixed number of daily transactions $\tau = 5,000$. In Figures 7.17 and 7.18, we present statistics for ten central entities based on their service profiles. For example, zigzag.io is a cryptocurrency exchange service, while ACINQ provides solutions for Bitcoin scalability. Additional entity profiles can be found in Table 7.2. In Figure 7.17, the income for most of the nodes significantly increases with transaction value, while this effect is almost negligible for rompert.com, LightningPowerUsers.com, and 1ML.com node ALPHA, whose behavior can be explained by charging almost only a base fee and applying a fee rate close

Table 7.2: LN network entities with related service profiles.

Entity name	Service profile
rompert.com	Provider of some Lightning Network related information
LNBIG.com	Half of the total network capacity in bound by this entity
zigzag.io	Exchange Top Cryptocurrencies in seconds with low fees
yalls.org	Read and write articles, with Lightning Network micropayments.
ln1.satoshilabs.com	Cryptocurrency solution developers
tippin.me	Send and receive Bitcoin tips on Twitter
ACINQ	One of the leading companies working on Bitcoin scalability
1ML.com node ALPHA	Lightning Network Search and Analysis Engine
LightningTo.Me	Helping to resolve routing and capacity issues
LightningPowerUsers.com	Request Inbound Capacity

Table 7.3: Estimated optimal channel capacity reduction for maximal RoI of the routers of Table 7.1. Capacity fraction is the estimated optimal fraction of the original channel capacities and income fraction is the estimated fraction of the original income by using reduced channel capacities.

Entity name	RoI gain (times)	Capacity fraction	Income fraction	Original RoI (%)	Optimal RoI (%)	Optimal RoI rank	Original RoI rank
lnbig.com	15.263039	0.01	0.152630	0.004033	0.061557	5.0	17.0
Bitrefill.com	8.815776	0.01	0.088158	0.001121	0.009881	21.0	25.0
yalls.org	7.274128	0.05	0.363706	0.041199	0.299685	3.0	4.0
fairly.cheap	5.527895	0.01	0.055279	0.001950	0.010781	18.0	24.0
LightningTo.Me	4.428039	0.05	0.221402	0.002317	0.010258	20.0	20.0
ln.BitSoapBox.com	4.270262	0.10	0.427026	0.002276	0.009720	22.0	21.0
ACINQ	3.492428	0.05	0.174621	0.002975	0.010391	19.0	18.0
LightningPowerUsers.com	3.374553	0.10	0.337455	0.005514	0.018608	13.0	16.0
Blockstream Store	3.211826	0.10	0.321183	0.017364	0.055771	6.0	8.0
ln1.satoshilabs.com	3.165573	0.05	0.158279	0.016167	0.051176	8.0	9.0
There be dragons here	3.064046	0.05	0.153202	0.018050	0.055307	7.0	7.0
Electrophorus [W_C_B]	2.229242	0.10	0.222924	0.009425	0.021010	11.0	14.0
lightning-roulette.com	1.724278	0.10	0.172428	0.002421	0.004174	23.0	19.0
BlueWallet	1.672075	0.05	0.083604	0.010588	0.017704	15.0	13.0
zigzag.io	1.511212	0.10	0.151121	0.258036	0.389947	2.0	2.0
OpenNode	1.458822	0.50	0.729411	0.021519	0.031392	10.0	6.0
tady je slushovo	1.445600	0.50	0.722800	0.010622	0.015355	16.0	12.0
BOLTENING.club	1.430646	0.10	0.143065	0.002166	0.003098	24.0	22.0
btc.lnetwork.tokyo	1.422923	0.50	0.711462	0.033247	0.047307	9.0	5.0
CoinGate	1.400418	0.50	0.700209	0.002129	0.002981	25.0	23.0
rompert.com	1.330968	0.50	0.665484	3.458924	4.603716	1.0	1.0
ORANGESQUIRREL	1.313521	0.50	0.656760	0.104882	0.137764	4.0	3.0
tippin.me	1.297128	0.50	0.648564	0.016009	0.020766	12.0	10.0
1ML.com node ALPHA	1.273918	0.50	0.636959	0.009021	0.011491	17.0	15.0
Sagittarius A	1.216862	0.50	0.608431	0.014715	0.017906	14.0	11.0

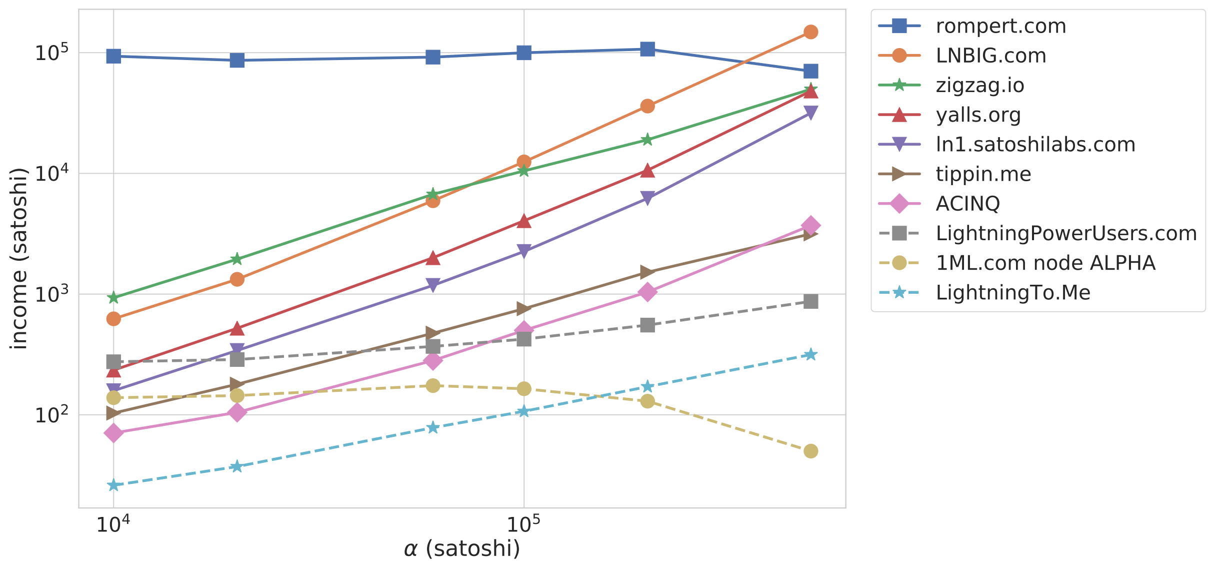


Fig. 7.17: Average simulated daily routing income of some LN router entities as the function of the transaction value α.

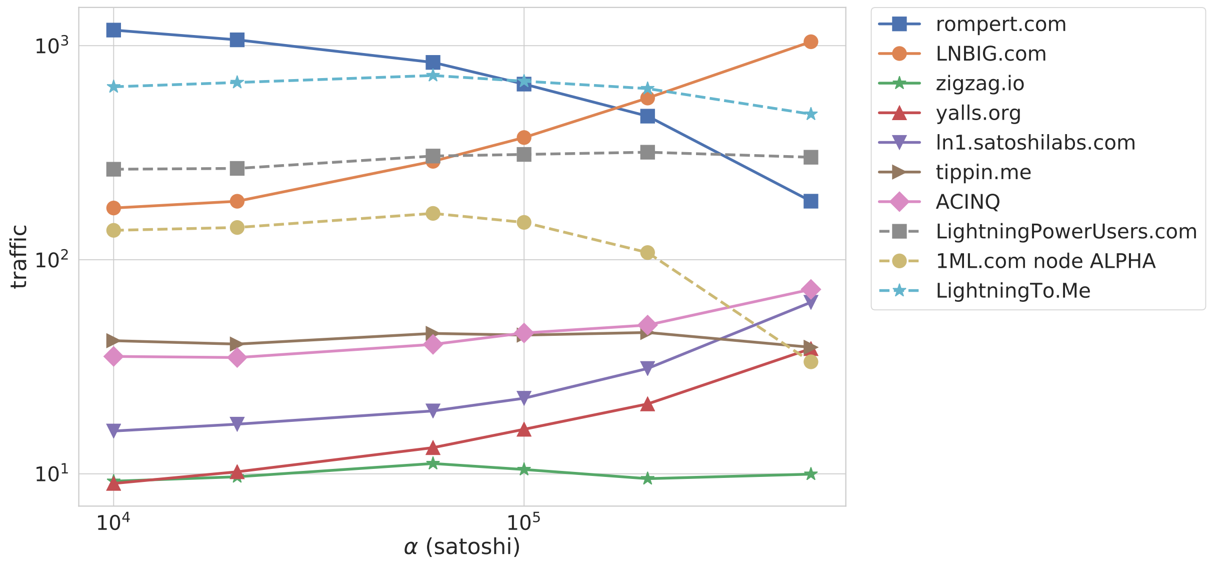


Fig. 7.18: Average simulated daily routing traffic of some LN router entities as the function of the transaction value α.

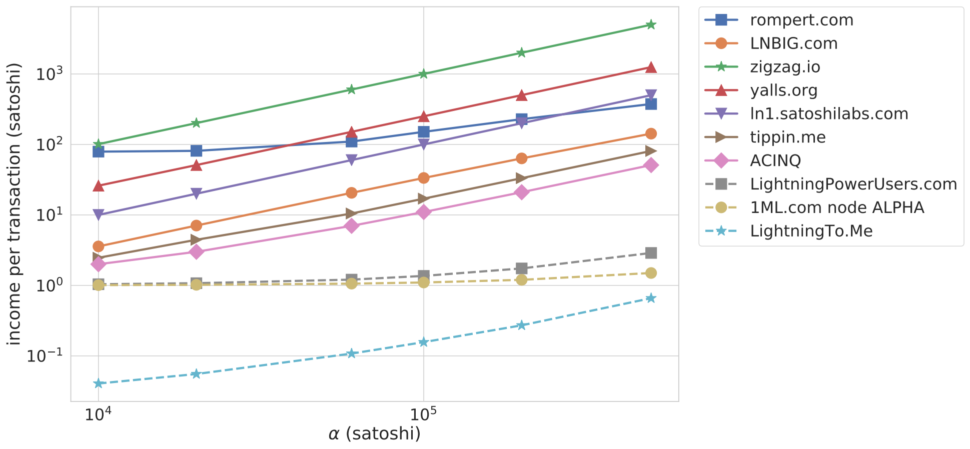


Fig. 7.19: Average simulated daily routing income per transaction for some LN router entities as the function of the transaction value α.

to zero.

The simulated amount of daily traffic for the ten central nodes is shown in Figure 7.18. We observe that scalability and capacity providers LightningTo.Me, LightningPowerUsers.com, and 1ML.com node ALPHA are responsible for forwarding a significant amount of payments irrespective of α. Probably due to the lack of high capacity channels, the traffic of rompert.com and 1ML.com node ALPHA drop at $\alpha = 500,000$ satoshis (≈ 41 USD). By contrast, the number of payments routed by LNBIG.com increases with payment value due to the fact that this entity owns approximately half of all network capacity, as seen Table 7.1. In Figure 7.19, we provide an efficiency metric for each entity by dividing estimated income by traffic volume. The efficiency of rompert.com and LNBIG.com are surpassed by zigzag.io and yalls.org for $\alpha \geq 60,000$ satoshis, as these service providers have reasonable routing income relative to the number of daily forwarded transactions. On the other hand, LightningPowerUsers.com, 1ML.com node ALPHA, and LightningTo.Me have orders of magnitude lower efficiency than other relevant entities. They are likely not considering routing profitability, as their transaction fees are negligible.

Next, we estimate the effect of channel depletion, which can be a side-effect of increasing the traffic without increasing channel capacities. In a highly simplistic experiment, we compare traffic with simulated channel depletion with the case when we allow the simulator to use channel directions without limits. We take depletion into account by suspending depleted channels until a reverse payment reopens them. On the top of Figure 7.20, we show the routing income estimate with depletion taken into account for the top ten router nodes, as the function of τ. And on

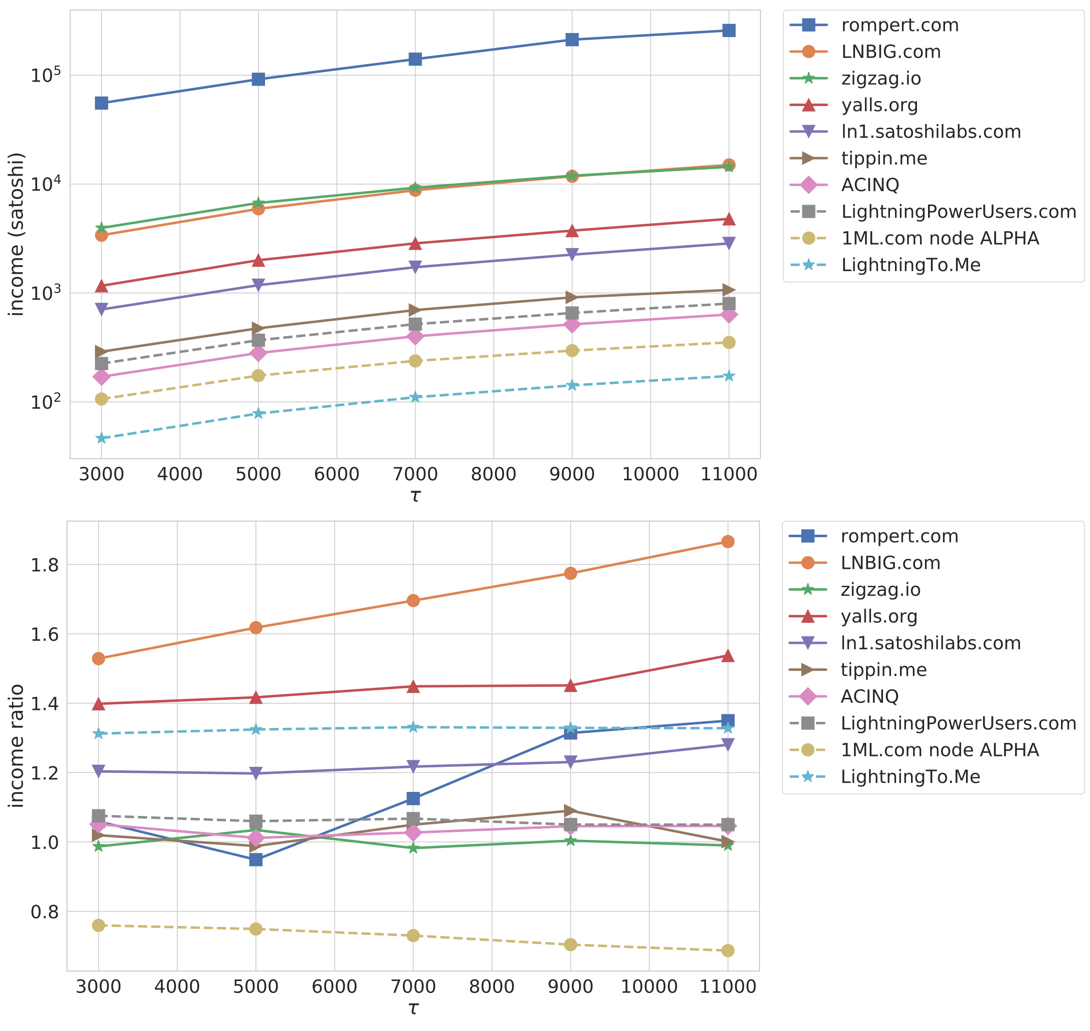


Fig. 7.20: Average simulated daily routing income **(top)** and the income divided by the optimistic income when channel depletion is ignored **(bottom)** for some LN router entities as the function of the simulated transaction count τ. Note that the ratio is above 1 for most nodes as they can take over routing for depleted channels.

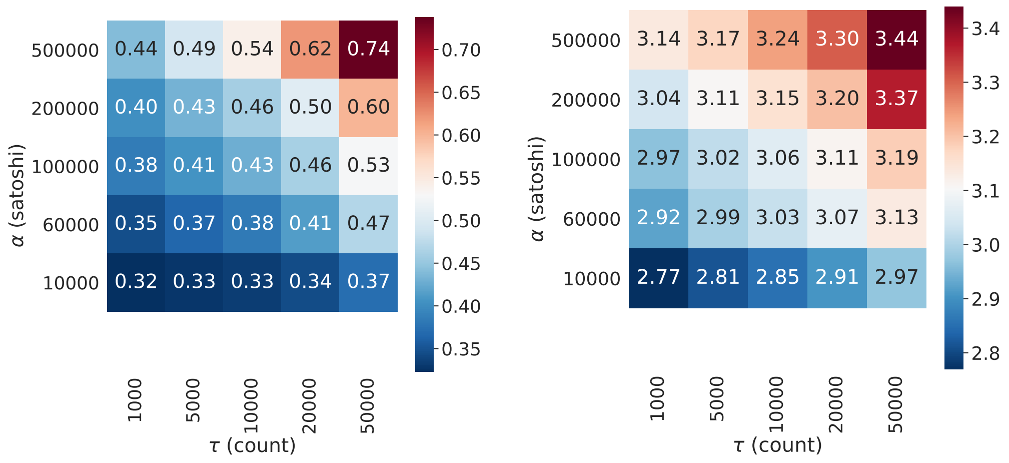


Fig. 7.21: Fraction of failed transactions **(left)** and average length of completed payment paths **(right)** with respect to the simulated transaction value α and the number of sampled transactions τ.

the bottom of Figure 7.20, we show the ratio of the routing income with and without depletion taken into account. At first glance, it is surprising that the fraction is above 1 for most of the router nodes. To explain, observe that channels with low routing fees are used and depleted first, and these channels will loose revenue compared to the optimistic case. However, if there is an alternate routing path with more expensive transaction fees, the owners of these channels will observe an increase in revenue due to the depletion of low cost channels.

As we simulate more traffic or execute more expensive payments, both the fraction of unsuccessful payments and the average length of completed payment paths increase, as we show in Figure 7.21. Transactions can fail in the simulation when there is no path from the source to the recipient such that the channels have at least α available capacity. If α is too high, then only a fraction of all channels can be used for payment routing, while in the case of an extremely large number of transactions, the available capacity of several channel directions becomes depleted. For example, channels leading to popular merchants could become blocked in case of heavy one-directional traffic. The growth in completed payment path length is in agreement with this scenario.

In Figure 7.21, we also observe that lower payment amounts do not significantly decrease the probability of a payment being successfully routed. Hence, we do not expect that Atomic Multi-path payments (AMP)[9] that allow a sender to atomically split a payment flow amongst several individual payment flows can significantly increase the success rate of the transactions.

[9]See: https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html

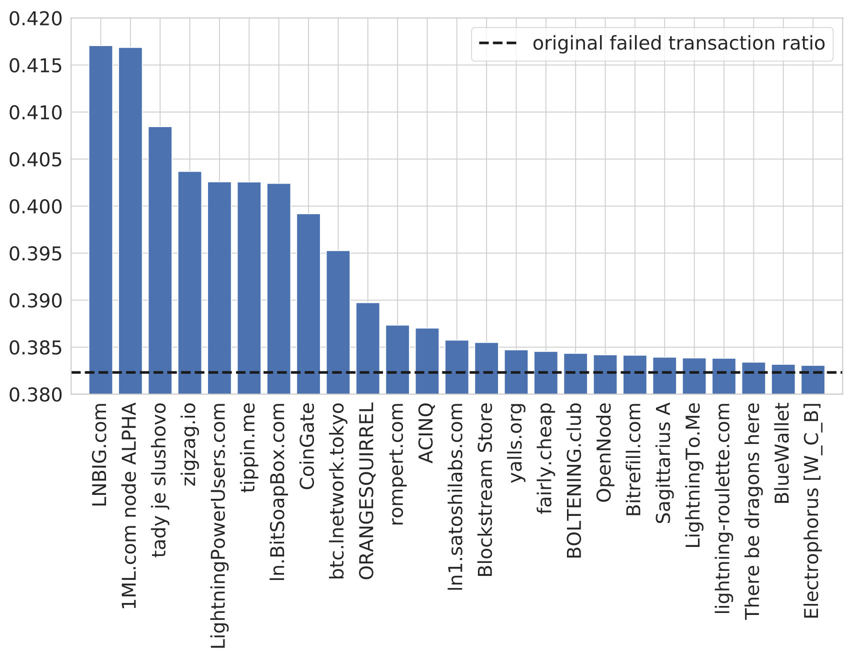


Fig. 7.22: The fraction of incomplete payments, out of the simulated $\tau = 5000$ transactions, after removing the given entity from LN. The original fraction of failed transactions 0.3823 is marked by the dashed line.

A final relevant metric is the number of payments that fail if the given entity becomes unavailable. In Figure 7.22, we show the fraction of unsuccessful payments after removing the given entity. For example, after removing the 25 nodes of LNBIG.com from LN, the rate of failed transactions increases to 0.417 from the original level of 0.382. Recall from Section 7.4 that a large fraction of the payments cannot be routed, since several nodes have only disabled or no outbound channels with capacity over the simulated payment value α.

In this section, we estimated the income of the central router nodes under various settings. Although our experiments confirm that at the present structure and level of usage, the participation for most routing nodes is not economical, we also foresee a potential in LN to make routing profitable with little adjustments in pricing and capacity policies if the traffic volume will increase.

7.8 Payment Privacy

While LN is often considered a privacy solution for Bitcoin as it does not record every transaction in a public ledger, the fundamentally different privacy implications of LN are often misunderstood [59,64]. LN provides little to no privacy for payments with a path of length two, since the single intermediary can de-anonymize both sender and receiver if it knows that the payment path is indeed of length two. The onion routing technique [71] used in LN provides a weaker notion of privacy called *plausible deniability*. By onion routing, an intermediary has no information on its position in the path and the sender node can claim that the payment was routed from one of its neighbors. In this sense, the privacy guarantees of LN payment routing are quite similar in spirit to that of TOR.

We remark that plausible deniability is also achieved for on-chain transactions by coin mixing techniques. In wallets supporting coin-mixing one can regularly observe privacy-enhanced transactions with large anonymity sets, where the identity of a sender is hidden by mixing with as many as 100 other transaction senders [85]. Hence for LN to provide privacy guarantees stronger than on-chain transactions, offering plausible deniability in itself can be insufficient.

Next, we assess the strength of privacy for simulated LN payments. By our discussion, high node degrees and long payment paths are compulsory for privacy. First, payments from low degree nodes are vulnerable, as the immediate predecessor or successor set is too small and can allow privacy attacks for example by investigating possible channel balances. Second, the majority of payments should be long, otherwise an intermediary has strong statistical evidence for the source or the destination of a large number its routed payments.

In Figure 7.23, we plot the fraction of nodes with sufficiently high degree to plausibly hide its payment as to be originating from one of its neighbors. We observe that half of the nodes have five or less neighbors, which makes their transactions vulnerable for attacks based on information either directly obtained from its neighbors, or inferred through investigating channel capacities. Furthermore, privacy guarantees are worsened as the value of the payment increases, since we can exclude payment channels from payment source candidates with capacity less than the payment value.

Next, we investigate the possible length of payment paths and the trade-off between length and cost. Note that the source has control over the payment path, hence it can deliberately select long paths to maintain its privacy, however this can result in increased costs.

The topological properties of LN, namely, its small-world nature, allow for very short payment path lengths. The average shortest path length of LN is around 2.8 [134], meaning that most payment routes involve one or two intermediaries. This phenomenon is further exacerbated by

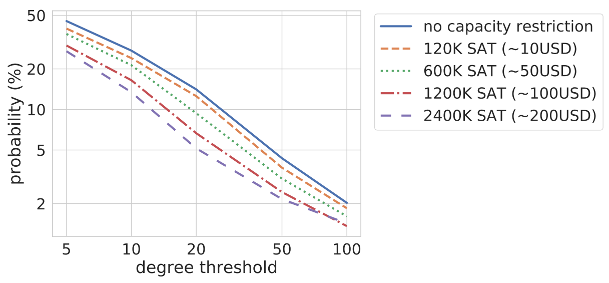


Fig. 7.23: The probability that a node has more channels with at least the given capacity than the degree threshold. Observe that larger payment amounts increase the risk of yielding more statistical evidence for tracing the source or destination of a payment.

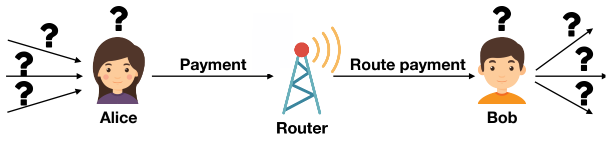


Fig. 7.24: Plausible deniability in LN. Alice can plausible deny being the source of a payment. Similarly, router cannot be sure whether Bob is the recipient of the payment or one of Bob's neighbors.

the client software, which prefers choosing shortest paths[10], resulting in a considerable fraction of single-hop transactions. However, we note that newer advancements in LN client softwares, e.g. c-lightning, incorporate solutions to decrease the portion of single-hop payments [11]

Loosely connecting to merchants and paying them only via routing facilitated by intermediaries is advantageous not just for privacy considerations but also for reducing the required number of payment channels, and thus limiting the amount that needs to be committed. By contrast, our measurements in Figure 7.3 showed that nodes seem to prefer opening direct links to other nodes and especially to merchant nodes. The figure is obtained by computing the shortest path length between u and v for each new edge (u, v) immediately before the new edge was created. If there is no such path, i.e., u and v lie in different connected components, we assign ∞ to the edge.

Simulations reveal that on average 16% of the payments are single-hop payments, see Figure 7.25. By increasing the fraction of merchants among receivers, this fraction increases to 34%, meaning that strong statistical evidence can be gathered on the payment source and destination through the router node for more than one third of the LN payments. We note that in practice, the ratio of de-anonymizable transactions might be even larger, since payments with longer routes can also be de-anonymized if all intermediary router nodes correspond to the same company.

In our final experiment, we estimate the payment fee increase by using longer paths in the existing

[10]Source: https://github.com/lightningnetwork/lnd/blob/40d63d5b4e317a4acca2818f4d5257271d4ac2c7/routing/pathfind.go

[11]Source: https://github.com/ElementsProject/lightning/commit/d23650d2edbfe16a21d0e637e507531a60dd2ddd.

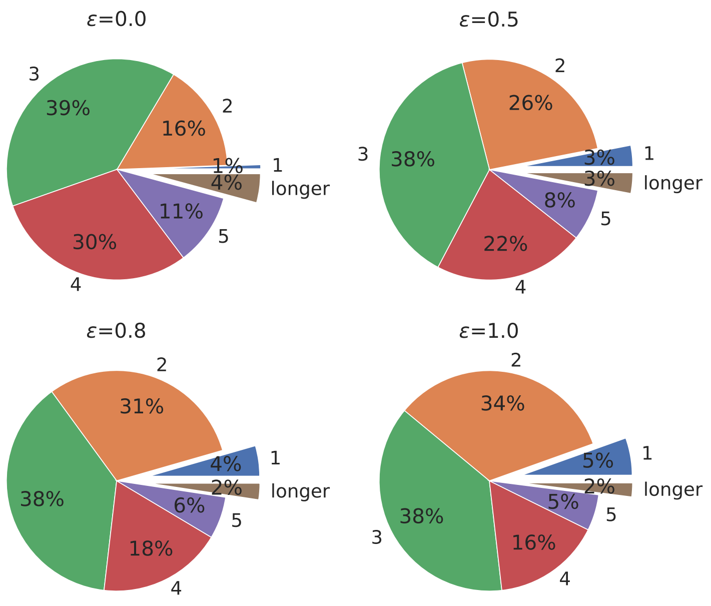


Fig. 7.25: Distribution of simulated path length with respect to the ratio of merchants as transaction endpoints ($\epsilon \in \{0.0, 0.5, 0.8, 1.0\}$).

network, based on the assumption that privacy-enhanced routed payments could be achieved by deliberately selecting longer payment routes. While paths of length more than a predefined number can be found in polynominal time [23], the algorithm is quite complex and in our case needs enhancements to use the edge costs.

We implemented a heuristic solution to find cheap but longer routing paths starting out with the existing known lowest cost one or ones. Note that one can come up with several variants of genetic search algorithms to generate candidate long routing paths similar to ours, and more refined methods can potentially incur even lower increase in the routing cost.

Our algorithm has one or more initial routing paths and a target length ℓ as input. In the first step, to increase the length of the routing paths, we attempt to replace edges by paths of length two. We iterate over the edges of the candidate routing paths until we obtain $k = 100$ paths of length ℓ, where k is a parameter. And in the second step, we generate more routing paths by shuffling the nodes at the same distance from the source and the target across the existing candidates. Finally, we select the lowest cost length ℓ routing path from the set produced by the algorithm.

As the conclusion of our experiment with searching for longer low cost routing paths, in Figure 7.26 we observe that we can find routing paths that only marginally increase the median cost of the transactions by selecting paths of length up to six.

In summary, we observed the very small world nature of LN, which is in contrast to the fact that privacy-aware payment routing could be achieved by deliberately selecting longer payment routes. The fact that many channel openings are triangle closing could suggest the unreliability of payment routing in LN. Another reason for the creation of triangle-closing payment channels can also be the possibility to inject additional hops to preserve transaction privacy, which, by our simulation, is a low additional cost solution to enhancing privacy.

Overall, we raised questions about the popular belief of the LN community that LN payments provide superior privacy than on-chain transactions. We believe that deliberately longer payment paths are required to maintain payment privacy, which does not drastically increase costs at the current level of transaction fees.

7.9 Conclusion

In this work, we analyzed Lightning Network, Bitcoin's payment channel network from a network scientific and cryptoeconomic point of view. Past results on the Lightning Network were unable to analyze the fee and revenue structure, as the data on the actual payments and amounts is strictly private. Our main contribution is an open-source LN traffic simulator that enables research on the cryptoeconomic consequences of the network topology without requiring information on the actual financial flow over the network. The simulator can incorporate the assumption that the payments are mostly targeted towards the merchants identified by using the tags provided by node owners. We validated some key parameters of the simulator such as traffic volume and amount by simulating the revenue of central router nodes and comparing the results with information published by certain node owners. By using our open source tool, we encourage node owners to build more accurate estimates of LN properties by incorporating their private

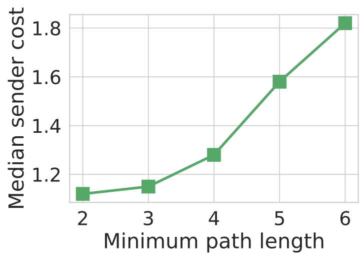


Fig. 7.26: Median sender costs in satoshis for fixed path length routing.

knowledge on usage patterns.

Our simulator provided us with two main insights. First, the participation of most router nodes in LN is economically irrational with the present fee structure; however, signs of sustainability are seen with increased overall traffic volume over the network. By contrast, at the present level of usage, if routers start acting rationally, payment fees will rise significantly, which might harm one of LN's core value propositions, namely, negligible fees. Second, the topological properties of LN make a considerable fraction of payments easily de-anonymizable. However, with the present fee structure, paths can be obfuscated by injecting extra hops with low cost to enhance payment privacy. We release the source code of our simulator for further research on GitHub[12].

[12]https://github.com/ferencberes/LNTrafficSimulator

Summary

In this work, we proposed several techniques to mine social and cryptocurrency networks. First, we developed an online network centrality measure, temporal Katz centrality based on the notion of time-respecting walks. By definition, the centrality score for a given node at time t is the weighted sum of time-respecting walks that reach this node up to time t. Our model is an online updateable extension of the well-known Katz-index for dynamically evolving networks. We conducted several experiments on large social networks and found that (1) temporal Katz centrality outperforms Temporal PageRank and static centrality metrics, as well as (2) adapts well to concepts drifts in the data.

Second, we developed two online node embedding algorithms for graph streams. Our StreamWalk algorithm is built upon the notion of time-respecting walks similar to temporal Katz centrality, while our online second order similarity algorithm directly learns the neighborhood overlap of node pairs as new links arrive from the edge stream. We deployed our online node embedding algorithms for a dynamic node similarity search task and found that they outperform static baselines such as LINE, Node2Vec, and DeepWalk.

Our results related to static node embedding applications on social and cryptocurrency networks are twofold.

- By collecting Covid-19 vaccine-related discussions from Twitter, we were able to deploy static node embedding models on the underlying reply network between users. Our results show that representations learned by node embedding models are useful features for the task of vaccine skepticism detection. Furthermore, the representation space even managed to capture the topic hierarchy for vax-skeptic and pro-vaxxer users.
- We deployed static node embeddings to link Ethereum addresses that belong to the same user. Our supervised evaluation shows that node embedding models are very powerful tools

for Ethereum address deanonymization, and it requires a carefully planned presence on the blockchain to fool these models.

Finally, we analyzed the Bitcoin Lightning Network from a profitability and privacy perspective. We designed a payment traffic simulator that we fit for profitability reports of network participants. Our simulated results reveal that central router nodes have low RoI in this payment channel network. On the other hand, they have very strong statistical evidence on payment sender and receiver nodes as payments are usually realized through short paths. In this work, we also propose a genetic algorithm to improve payment security with only a negligible increase in the general transaction fee paid by payment senders.

Összefoglalás

Kutatásom során számos technikát javasoltunk a közösségi és kriptopénz hálózatok elemzésére. Először is kidolgoztunk egy valós időben frissíthető gráf központiság mértéket, a dinamikus Katz központiságot, amely az időrendezett séták fogalmára épül. Definíciónk szerint a gráf egy adott csúcsának központiság mérőszáma egy adott időpontban a csúcsot elérő időrendezett séták súlyozott összege. Az általunk javasolt módszer a jól ismert Katz-index online frissíthető kiterjesztése időben változó gráfokra. Nagyméretű közösségi hálózatokon végzett méréseink igazolták, hogy (1) a dinamikus Katz központiság jobban teljesít, mint a szintén időrendezett utakra épülő dinamikus PageRank, illetve a hagyományos statikus központiság mértékek. Emellett (2) a módszer jól alkalmazkodik az adatfolyam/élfolyam eloszlásában bekövetkezett változásokhoz.

Disszertációm ezután részletesen beszámol az általunk kifejlesztett két online gráf csomópont-beágyazó algoritmusról. A StreamWalk módszerünk szintén az időrendezett útvonalak koncepciójára épít, míg az online másodrendű hasonlóság módszerünk közvetlenül megtanulja a csúcspárok szomszédsági hasonlóságát, amint új élek érkeznek az élfolyamból. A javasolt algoritmusokat egy dinamikus csomópont-hasonlóság-keresési feladaton értékeltük ki, és azt találtuk, hogy módszereink felülmúlják a statikus gráfokra kifejlesztett csúcs-beágyazó eljárásokat, például a LINE-t, a node2vec-et vagy a DeepWalk-ot.

Munkám során a statikus csomópont-beágyazó algoritmusok számos új alkalmazási lehetőségét is megvizsgáltam. Főbb eredményeim a következők:

- Covid-19 védőoltásokhoz kapcsolódó nagyméretű tweet adathalmazunkból kinyert gráfon tanítottam statikus csomópont-beágyazó algoritmusokat. Gépi tanulás segítségével igazoltam, hogy a beágyazó módszerek által megtanult felhasználó reprezentációk jelentősen javítják az oltásszkepticizmus detektálhatóságát a Twitter üzenetekben. Méréseink továbbá igazolták a vakcina támogató és ellenző felhasználók, illetve az őket érdeklő altémák el-

szeparálódását a reprezentációs térben.

- Egy másik kutatás során elsőként vizsgáltuk meg csomópont-beágyazó módszerek hatékonyságát kriptopénz hálózatokon. Esetünkben az Ethereum tranzakciós gráfon elemeztük az azonos felhasználóhoz tartozó Ethereum fiókok felfedhetőségét. Méréseink kimutatták, hogy a csomópont-beágyazási modellek nagyon hatékony eszközök az Ethereum fiókok deanonimizálásához, és gondosan megtervezett felhasználói jelenlét szükséges a blokkláncon ahhoz, ha el akarjuk kerülni, hogy a vizsgált algoritmusok könnyen ránk tanuljanak és ezzel felfedjék az általunk kezelt Ethereum fiókokat.

A kriptopénz hálózatok elemzését folytatva, megvizsgáltuk a Bitcoin Lightning Networköt (LN) az adatvédelem és a részvevők befektetett pénzmennyiségének megtérülése szempontjából. Egy olyan LN forgalom szimulátort terveztünk, amely képes megbecsülni a hálózat résztvevőinek tranzakciós költségekből származó jövedelmét. Szimulált eredményeink azt mutatják, hogy a hálózat működése szempontjából elengedhetetlen központi csúcsok is csak rendkívül alacsony megtérülés mellett üzemelnek. Az adatvédelem szempontjából viszont kedvezőtlen módon, ezek a csúcsok nagy valószínűséggel ismerik a hálózatban lebonyolított kifizetések feladó és címzett csúcsait, mely információknak rejtve kellene maradnia. A probléma kiküszöbölésére egy genetikus algoritmust javasoltunk, amely képes biztonságosabb útvonalon realizálni a kifizetéseket, és ezzel csak elhanyagolható többlet költséget ruházva a feladó csúcsokra.

List of abbreviations

Data sets or events	UO17	US Open 2017 Twitter data set (see Section 2.2)
	RG17	Roland-Garros 2017 Twitter data set (see Section 2.2)
	TREC	Text Retrieval Conference
Performance metrics	DCG	Discounted Cumulative Gain
	NDCG	Normalized Discounted Cumulative Gain
	AUC	Area under the ROC curve
	RoI	Return of interest
Models	GF	Graph factorization [2]
	DW	DeepWalk [118]
	SW	StreamWalk (see Section 4.3.1)
	SO	Online second order similarity (see Section 4.3.2)
	SW+SO	Combination of SW and SO (see Section 4.4.3)
Cryptocurrency networks	PCN	Payment channel network
	LN	Lightning network
	AMP	atomic multipath payments
	P2P	Peer-to-peer
	UTXO	unspent transaction output
	ENS	Ethereum Name Service
	ETH	Ether
	TC	Tornado Cash
	EAO	Externally owned account
	EVM	Ethereum Virtual Machine
	ICO	Initial Coin Offering
	ERC-20	A technical standard for Ethereum-based tokens
	zkSNARK	Zero-Knowledge Succinct Non-Interactive Argument of Knowledge

Bibliography

[1] Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.

[2] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48. ACM, 2013.

[3] Nesreen Ahmed, Ryan Rossi, John Lee, Xiangnan Kong, Theodore Willke, Rong Zhou, and Hoda Eldardiry. Learning role-based graph embeddings. In *StarAI workshop, IJCAI 2018*, pages 1–8, 2018.

[4] Azzah Al-Maskari, Mark Sanderson, and Paul Clough. The relationship between ir effectiveness measures and user satisfaction. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 773–774. ACM, 2007.

[5] Ahmad Alsayed and Desmond J Higham. Betweenness in time dependent networks. *Chaos, Solitons & Fractals*, 72:35–48, 2015.

[6] Pablo Aragón, Karolin Eva Kappler, Andreas Kaltenbrunner, David Laniado, and Yana Volkovich. Communication dynamics in twitter during political campaigns: The case of the 2011 spanish national election. *Policy & Internet*, 5(2):183–206, 2013.

[7] Georgia Avarikioti, Gerrit Janssen, Yuyi Wang, and Roger Wattenhofer. Payment network design with fees. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 76–84. Springer, 2018.

[8] Georgia Avarikioti, Rolf Scheuner, and Roger Wattenhofer. Payment networks as creation games, 2019.

[9] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[10] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. In *Proceedings of the 3rd Annual ACM Web Science Conference*, 2012.

[11] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3):173–184, 2010.

[12] Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Pagerank on an evolving graph. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 24–32. ACM, 2012.

[13] LEI BAI, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17804–17815. Curran Associates, Inc., 2020.

[14] Eytan Bakshy, Jake M Hofman, Winter A Mason, and Duncan J Watts. Everyone's an influencer: quantifying influence on twitter. In *Proceedings of the 4th international conference on Web search and data mining (WSDM)*, 2011.

[15] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, volume 2, pages 623–632, 2002.

[16] Albert-László Barabási. Scale-free networks: A decade and beyond. *Science*, 325(5939):412–413, 2009.

[17] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, pages 585–591, 2002.

[18] András Benczúr, Ferenc Béres, Domokos Kelen, and Róbert Pálovics. Tutorial on graph stream analytics. DEBS '21, page 168–171, New York, NY, USA, 2021. Association for Computing Machinery.

[19] Ferenc Béres, Domokos M. Kelen, Róbert Pálovics, and András A Benczúr. Node embeddings in dynamic graphs. *Applied Network Science*, 4(64):25, 2019.

[20] Ferenc Béres, Róbert Pálovics, Anna Oláh, and András A Benczúr. Temporal walk based centrality metric for graph streams. *Applied Network Science*, 3(32):26, 2018.

[21] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010.

[22] BitMEX. The lightning network (part 2) – routing fee economics. https://blog.bitmex.com/the-lightning-network-part-2-routing-fee-economics/.

[23] Hans L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1):1–23, 1993.

[24] Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014.

[25] Dan Braha and Yaneer Bar-Yam. From centrality to temporary fame: Dynamic centrality in complex networks. *Complexity*, 12(2):59–63, 2006.

[26] Simina Brânzei, Erel Segal-Halevi, and Aviv Zohar. How to charge lightning. *arXiv preprint arXiv:1712.10222*, 2017.

[27] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[28] Ferenc Béres and András A. Benczúr. Online centrality in temporally evolving networks. In *Book of Abstracts of the 6th International Conference on Complex Networks and Their Applications*, pages 184–186, 2017.

[29] Ferenc Béres, Rita Csoma, Tamás Vilmos Michaletzky, and András A. Benczúr. Vaccine skepticism detection by network embedding. In *Book of Abstracts of the 10th International Conference on Complex Networks and Their Applications*, pages 241–243, 2021.

[30] Ferenc Béres, Róbert Pálovics, and András A. Benczúr. Temporal walk based centrality metric for graph streams. In *14th International Workshop on Mining and Learning with Graphs, held in conjunction with KDD'18*, 2018.

[31] Ferenc Béres, Róbert Pálovics, Domokos M. Kelen, Dávid Szabó, and András A. Benczúr. Node embeddings in dynamic graphs. In *Book of Abstracts of the 7th International Conference on Complex Networks and Their Applications*, pages 178–180, 2018.

[32] Ferenc Béres, István András Seres, and András A Benczúr. A cryptoeconomic traffic analysis of bitcoin's lightning network. *Cryptoeconomic Systems*, 1(1), 2021.

[33] Ferenc Béres, István András Seres, András A Benczúr, and Mikerah Quintyne-Collins. Blockchain is watching you: Profiling and deanonymizing ethereum users. In *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 69–78, 2021.

[34] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 891–900. ACM, 2015.

[35] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 891–900, New York, NY, USA, 2015. Association for Computing Machinery.

[36] Yi Chang, Xuanhui Wang, Qiaozhu Mei, and Yan Liu. Towards twitter context summarization with user influence models. In *Proceedings of the 6th international conference on Web search and data mining (WSDM)*, 2013.

[37] Hassan Nazeer Chaudhry. Flowgraph: Distributed temporal pattern detection over dynamically evolving graphs. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, pages 272–275, 2019.

[38] Nicolas Christin. Traveling the silk road: A measurement analysis of a large anonymous online marketplace. In *Proceedings of the 22nd international conference on World Wide Web*, pages 213–224, 2013.

[39] Charles LA Clarke, Nick Craswell, and Ian Soboroff. Overview of the trec 2004 terabyte track. In *TREC*, volume 4, page 74, 2004.

[40] Marco Conoscenti, Antonio Vetrò, Juan De Martin, and Federico Spini. The cloth simulator for htlc payment networks with introductory lightning network performance results. *Information*, 9(9):223, 2018.

[41] Kyle Croman, Christian Decke, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, and Emin Gun Sirer. Ds an, and r. wattenhofer.

on scaling decentralized blockchains (a position paper). In *3rd Workshop on Bitcoin and Blockchain Research*, 2016.

[42] Gianmarco De Francisci Morales, Albert Bifet, Latifur Khan, Joao Gama, and Wei Fan. Iot big data stream mining. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2119–2120. ACM, 2016.

[43] Nicholas Diakopoulos, Munmun De Choudhury, and Mor Naaman. Finding and assessing social media information sources in the context of journalism. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2012.

[44] Claudia Diaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *International Workshop on Privacy Enhancing Technologies*, pages 54–68. Springer, 2002.

[45] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment channels over cryptographic currencies. *IACR Cryptology ePrint Archive*, 2017:635, 2017.

[46] David Easley, Maureen O'Hara, and Soumya Basu. From mining to markets: The evolution of bitcoin transaction fees. *Journal of Financial Economics*, 2019.

[47] Felix Engelmann, Henning Kopp, Frank Kargl, Florian Glaser, and Christof Weinhardt. Towards an economic analysis of routing in payment channel networks. In *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, page 2. ACM, 2017.

[48] Dániel Fogaras and Balázs Rácz. Scaling link-based similarity search. In *Proceedings of the 14th World Wide Web Conference*, pages 641–650, Chiba, Japan, 2005.

[49] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments. *Internet Mathematics*, 2(3):333–358, 2005. Preliminary version from the first two authors appeared in WAW 2004.

[50] Daniel Gayo-Avello. A meta-analysis of state-of-the-art electoral prediction from twitter data. *Social Science Computer Review*, 2013.

[51] Evangelos Georgiadis. How many transactions per second can bitcoin really handle? theoretically. *IACR Cryptology ePrint Archive*, 2019:416, 2019.

[52] Marwan Ghanem, Florent Coriat, and Lionel Tabourier. Ego-betweenness centrality in link streams. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, ASONAM '17, page 667–674, New York, NY, USA, 2017. Association for Computing Machinery.

[53] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, 2010.

[54] Steven Goldfeder, Harry A. Kalodner, Dillon Reisman, and Arvind Narayanan. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *CoRR*, abs/1708.04748, 2017.

[55] Maoguo Gong, Chuanyu Yao, Yu Xie, and Mingliang Xu. Semi-supervised network embedding with text information. *Pattern Recognition*, 104:107347, 2020.

[56] Peter Grindrod and Desmond J Higham. A dynamical systems view of network centrality. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 470, 2014.

[57] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[58] Cyril Grunspan and Ricardo Pérez-Marco. Ant routing algorithm for the lightning network. *arXiv preprint arXiv:1807.00151*, 2018.

[59] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Off the chain transactions. *IACR Cryptology ePrint Archive*, 2019:360, 2019.

[60] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[61] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[62] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[63] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

[64] Jordi Herrera-Joancomartí and Cristina Pérez-Solà. Privacy in bitcoin transactions: new challenges from blockchain scalability solutions. In *International Conference on Modeling Decisions for Artificial Intelligence*, pages 26–44. Springer, 2016.

[65] Scott A Hill and Dan Braha. Dynamic model of time-dependent complex networks. *Physical Review E*, 82(4):046105, 2010.

[66] Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

[67] Weishu Hu, Haitao Zou, and Zhiguo Gong. Temporal pagerank on social networks. In *International Conference on Web Information Systems Engineering*, pages 262–276. Springer, 2015.

[68] Chia-Feng Juang and Chin-Teng Lin. An online self-constructing neural fuzzy inference network and its applications. *IEEE transactions on Fuzzy Systems*, 6(1):12–32, 1998.

[69] Nobuhiro Kaji and Hayato Kobayashi. Incremental skip-gram model with negative sampling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 363–371, 2017.

[70] Kerem Kaskaloglu. Near zero bitcoin transaction fees cannot last forever. In *Proceedings of the International Conference on Digital Security and Forensics (DigitalSec2014)*, 06 2014.

[71] Aniket Kate and Ian Goldberg. Using sphinx to improve onion routing circuit construction. In *International Conference on Financial Cryptography and Data Security*, pages 359–366. Springer, 2010.

[72] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[73] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 439–453. ACM, 2017.

[74] Nida Khan et al. Lightning network: A comparative review of transaction fees and data analysis. In *International Congress on Blockchain and Applications*, pages 11–18. Springer, 2019.

[75] Lucianna Kiffer, Dave Levin, and Alan Mislove. Analyzing ethereum's contract topology. In *Proceedings of the Internet Measurement Conference 2018*, pages 494–499, 2018.

[76] Hyoungshick Kim and Ross Anderson. Temporal node centrality in complex networks. *Physical Review E*, 85(2):026107, 2012.

[77] Kyung Soo Kim and Yong Suk Choi. Incremental iteration method for fast pagerank computation. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, page 80. ACM, 2015.

[78] Robin Klusman. Deanonymisation in ethereum using existing methods for bitcoin. 2018.

[79] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *Link mining: models, algorithms, and applications*, pages 337–357. Springer, 2010.

[80] Kristina Lerman, Rumi Ghosh, and Jeon Hyung Kang. Centrality metric for dynamic networks. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pages 70–77. ACM, 2010.

[81] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[82] Jia Li, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, and Lujia Pan. Predicting path failure in time-evolving graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '19, page 1279–1289, New York, NY, USA, 2019. Association for Computing Machinery.

[83] Shlomi Linoy, Natalia Stakhanova, and Alina Matyukhina. Exploring ethereum's blockchain anonymity using smart contract code attribution. 10 2019.

[84] LNBig. Guy makes $20 a month from locking $5 million bitcoin on the lightning network. https://www.trustnodes.com/2019/08/20/

guy-makes-20-a-month-for-locking-5-million-worth-of-bitcoin-on-the-lightning-network?fbclid=IwAR2-p8nWdg0ayO9S0Uz7qg3wmh_A8Wy6ueX8r3dLQvDTyJaj1ReSbYalnWI.

[85] ltcadmin. 100 bitcoin (btc) community members of wasabi wallet make the biggest coinjoin payment ever. https://icowarz.com/100-bitcoin-btc-community-members-of-wasabi-wallet-make-the-biggest-coinjoin-payment-ever/.

[86] Jiangtao Ma, Yaqiong Qiao, Guangwu Hu, Yongzhong Huang, Arun Kumar Sangaiah, Chaoqin Zhang, Yanjun Wang, and Rui Zhang. De-anonymizing social networks with random forest classifier. *IEEE Access*, 6:10139–10150, 2017.

[87] Bundit Manaskasemsak, Pramote Teerasetmanakul, Kankamol Tongtip, Athasit Surarerks, and Arnon Rungsawang. Computing personalized pagerank based on temporal-biased proximity. In *Information Technology Convergence*, pages 375–385. Springer, 2013.

[88] Stefano Martinazzi. The evolution of lightning network's topology during its first year and the influence over its core values. *arXiv preprint arXiv:1902.07307*, 2019.

[89] Patrick McCorry, Malte Möser, Siamak F Shahandasti, and Feng Hao. Towards bitcoin payment networks. In *Australasian Conference on Information Security and Privacy*, pages 57–76. Springer, 2016.

[90] Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

[91] Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(2):105–121, 2018.

[92] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140, 2013.

[93] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[94] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[95] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, Christopher Cordi, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. *arXiv preprint arXiv:1702.05812*, 2017.

[96] Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42. ACM, 2007.

[97] Michael Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet mathematics*, 1(2):226–251, 2004.

[98] Malte Möser and Rainer Böhme. Trends, tips, tolls: A longitudinal study of bitcoin transaction fees. In *International Conference on Financial Cryptography and Data Security*, pages 19–33. Springer, 2015.

[99] Goran Muric, Yusong Wu, and Emilio Ferrara. COVID-19 vaccine hesitancy on social media: Building a public twitter dataset of anti-vaccine content, vaccine misinformation and conspiracies. *CoRR*, abs/2105.05134, 2021.

[100] Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

[101] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, 2008.

[102] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[103] Arvind Narayanan, Elaine Shi, and Benjamin IP Rubinstein. Link prediction by de-anonymization: How we won the kaggle social network challenge. In *The 2011 International Joint Conference on Neural Networks*, pages 1825–1834. IEEE, 2011.

[104] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125. IEEE, 2008.

[105] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *2009 30th IEEE symposium on security and privacy*, pages 173–187. IEEE, 2009.

[106] Danny Nelson. Inside chainalysis' multimillion-dollar relationship with the us government. coindesk, 2020. https://www.coindesk.com/inside-chainalysis-multimillion-dollar-relationship-\with-the-us-government.

[107] Lynnette Hui Xian Ng and Kathleen M. Carley. Flipping stance: Social influence on bot's and non bot's COVID vaccine stance. *CoRR*, abs/2106.11076, 2021.

[108] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *3rd International Workshop on Learning Representations for Big Networks*, 2018.

[109] Feiping Nie, Wei Zhu, and Xuelong Li. Unsupervised large graph embedding. 2017.

[110] Robert Norvill, Beltran Borja Fiz Pontiveros, Radu State, Irfan Awan, and Andrea Cullen. Automated labeling of unknown contracts in ethereum. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6. IEEE, 2017.

[111] Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. Efficient pagerank tracking in evolving networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 875–884. ACM, 2015.

[112] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114. ACM, 2016.

[113] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, 1998.

[114] Aditya Pal and Scott Counts. Identifying topical authorities in microblogs. In *Proceedings of the 4th international conference on Web search and data mining (WSDM)*, 2011.

[115] Róbert Pálovics, András A Benczúr, Levente Kocsis, Tamás Kiss, and Erzsébet Frigó. Exploiting temporal influence in online recommendation. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 273–280. ACM, 2014.

[116] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, and Charles E Leisersen. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. *AAAI*, 2020.

[117] James Payette, Samuel Schwager, and Joseph Murphy. Characterizing the ethereum address space, 2017.

[118] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[119] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710, New York, NY, USA, 2014. ACM.

[120] Bryan Perozzi, Vivek Kulkarni, and Steven Skiena. Walklets: Multiscale graph embeddings for interpretable network classification. *CoRR*, abs/1605.02115, 2016.

[121] Rene Pickhardt. Earn bitcoin with lightning network routing fees and a little data science. https://www.youtube.com/watch?v=L39IvFqTZk8.

[122] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *See https://lightning. network/lightning-network-paper. pdf*, 2016.

[123] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *CoRR*, abs/1608.05859, 2016.

[124] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 459–467. ACM, 2018.

[125] BitMEX Research. Lightning network (part 7) – proportion of public vs private channels. https://blog.bitmex.com/lightning-network-part-7-proportion-of-public-vs-private-channels/.

[126] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. *arXiv preprint arXiv:1904.10253*, 2019.

[127] Martin Rosvall and Carl T Bergstrom. Mapping change in large networks. *PloS one*, 5(1), 2010.

[128] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *Proc. CIKM*, page 3125–3132. ACM, 2020.

[129] Benedek Rozemberczki and Rik Sarkar. Fast sequence based embedding with diffusion graphs. In *International Conference on Complex Networks*, pages 99–107, 2018.

[130] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Béres, Guzmán López, Nicolas Collignon, and Rik Sarkar. Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. CIKM '21, page 4564–4573, New York, NY, USA, 2021. Association for Computing Machinery.

[131] Polina Rozenshtein and Aristides Gionis. Temporal pagerank. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 674–689. Springer, 2016.

[132] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. 1998.

[133] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. *Journal of the ACM (JACM)*, 58(3):13, 2011.

[134] István András Seres, László Gulyás, Dániel A Nagy, and Péter Burcsi. Topological analysis of bitcoin's lightning network. *arXiv preprint arXiv:1901.04972*, 2019.

[135] István András Seres, Dániel A Nagy, Chris Buckland, and Péter Burcsi. Mixeth: efficient, trustless coin mixing service for ethereum. In *International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[136] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In *International Workshop on Privacy Enhancing Technologies*, pages 41–53. Springer, 2002.

[137] Omer Shlomovits and István András Seres. Sharelock: Mixing for cryptocurrencies from multiparty ecdsa. *Cryptol. ePrint Arch., Tech. Rep*, 563:2019, 2019.

[138] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. Graphscope: Parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining (KDD)*, 2007.

[139] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A 2 l: Anonymous atomic locks for scalability and interoperability in payment channel hubs. In *IACR Cryptology ePrint Archive*, 2019.

[140] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proc. ACL*, pages 1555–1565, 2014.

[141] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[142] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[143] John Tang, Mirco Musolesi, Cecilia Mascolo, Vito Latora, and Vincenzo Nicosia. Analysing information flows and key mediators through temporal centrality metrics. In *Proceedings of the 3rd Workshop on Social Network Systems*, page 3. ACM, 2010.

[144] Weizhao Tang, Weina Wang, Giulia Fanti, and Sewoong Oh. Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks, 2019.

[145] Dane Taylor, Sean A Myers, Aaron Clauset, Mason A Porter, and Peter J Mucha. Eigenvector-based centrality measures for temporal networks. *Multiscale Modeling & Simulation*, 15(1):537–574, 2017.

[146] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. Lemp: Fast retrieval of large entries in a matrix product. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 107–122. ACM, 2015.

[147] Manny Trillo. Stress test prepares visanet for the most wonderful time of the year. *URl: http://www. visa. com/blogarchives/us/2013/10/10/stress-testprepares-visanet-for-the-most-wonderful-time-of-the-year/index. html*, 2013.

[148] Friedhelm Victor. Address clustering heuristics for ethereum. In *Proceedings of the 24th Financial Cryptography Conference*, page 617–633, Berlin, Heidelberg, 2020. Springer-Verlag.

[149] Friedhelm Victor and Bianca Katharina Lüders. Measuring ethereum-based erc20 token networks. In *International Conference on Financial Cryptography and Data Security*, pages 113–129. Springer, 2019.

[150] Sebastiano Vigna. A weighted correlation index for rankings with ties. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*, 2015.

[151] Isabel Wagner and David Eckhoff. Technical privacy metrics: a systematic survey. *ACM Computing Surveys (CSUR)*, 51(3):1–38, 2018.

[152] Peilu Wang, Yao Qian, Frank K Soong, Lei He, and Hai Zhao. A unified tagging solution: Bidirectional lstm recurrent neural network with word embedding. *arXiv preprint arXiv:1511.00215*, 2015.

[153] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. AAAI'17, page 203–209. AAAI Press, 2017.

[154] Xiaokai Wei, Linchuan Xu, Bokai Cao, and Philip S. Yu. Cross view link prediction by learning noise-resilient representation consensus. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 1611–1619. International World Wide Web Conferences Steering Committee, 2017.

[155] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. Twitterrank: finding topic-sensitive influential twitterers. In *Proceedings of the 3rd international conference on Web search and data mining (WSDM)*, 2010.

[156] Barry Whitehat. Miximus: zksnark-based trustless mixing for ethereum. github, 2018. https://github.com/barryWhiteHat/miximus.

[157] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[158] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[159] Shuang Yang and Bo Yang. Enhanced network embedding with text information. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 326–331. IEEE, 2018.

[160] Yanwei Yu, Huaxiu Yao, Hongjian Wang, Xianfeng Tang, and Zhenhui Li. Representation learning for large-scale dynamic networks. In *International Conference on Database Systems for Advanced Applications*, pages 526–541. Springer, 2018.

[161] Yuhui Zhang, Dejun Yang, and Guoliang Xue. Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[162] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, 2020.

[163] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. Scalable graph embedding for asymmetric proximity. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017.

[164] Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 358–369. VLDB Endowment, 2002.

[165] Wei Zhuo, Qianyi Zhan, Yuan Liu, Zhenping Xie, and Jing Lu. Context attention heterogeneous network embedding. *Computational intelligence and neuroscience*, 2019, 2019.

[166] Indre Žliobaite, Albert Bifet, Mohamed Gaber, Bogdan Gabrys, Joao Gama, Leandro Minku, and Katarzyna Musial. Next challenges for adaptive learning systems. *ACM SIGKDD Explorations Newsletter*, 14(1):48–55, 2012.

[167] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2857–2866. ACM, 2018.