TGLib: An Open-Source Library for Temporal Graph Analysis

Lutz Oettershagen  
University of Bonn  
Bonn, Germany  
lutz.oettershagen@cs.uni-bonn.de

Petra Mutzel  
University of Bonn  
Bonn, Germany  
petra.mutzel@cs.uni-bonn.de

Abstract—We initiate an open-source library for the efficient analysis of temporal graphs. We consider one of the standard models of dynamic networks in which each edge has a discrete timestamp and transition time. Recently there has been a massive interest in analyzing such temporal graphs. Common computational data mining and analysis tasks include the computation of temporal distances, centrality measures, and network statistics like topological overlap, burstiness, or temporal diameter. To fulfill the increasing demand for efficient and easy-to-use implementations of temporal graph algorithms, we introduce the open-source library TGLib, which integrates efficient data structures and algorithms for temporal graph analysis. TGLib is highly efficient and versatile, providing simple and convenient C++ and Python interfaces, targeting computer scientists, practitioners, students, and the (temporal) network research community.

Index Terms—temporal graph, data mining, centrality, open-source library

I. INTRODUCTION

TGLib is an open-source C++ template library with an easy-to-use Python front-end focusing on efficient temporal graph analysis tasks. Network data often originates from dynamic systems that change over time: Links are formed or broken, such that the topology of the network changes over time. Temporal graphs capture these changes. A temporal graph is a graph that changes over time, i.e., each edge has a time stamp that determines when the edge exists in the graph. Hence, the topology of the graph changes in discrete time steps. Temporal graphs are often good models for real-life scenarios due to the inherently dynamic nature of most real-world activities and processes. In many situations, events, e.g., communication in social networks, are a prime example of the application of temporal graphs. Email (or text message) networks model the (almost) instantaneous communication between the participants, and have been used to identify different dynamics of communication as well as properties of the participants [1]–[3]. Vertices of the network represent the participants and temporal edges represent each communication.

Communication networks are a prime example of the application of temporal graphs. Email (or text message) networks model the (almost) instantaneous communication between the participants, and have been used to identify different dynamics of communication as well as properties of the participants [1]–[3]. Vertices of the network represent the participants and temporal edges represent each communication.

Proximity and contact networks record the contacts between individuals by measuring their proximity. For example, modern smartphones are ubiquitous and can record the proximity of users to identify contacts or build opportunistic networks [4], [5]. Several works discuss the spreading of diseases in contact networks, e.g., [6], [7].

Social networks, formal or informal, are a fundamental part of human life. Nowadays, online social networks, like Facebook or WeChat, host billions of users. In online and offline social networks, participants join and leave the network over time and form or end relations with each other. Recent works discuss the importance of temporal properties, e.g., [8], [9].

There are many more prominent use-cases for temporal graphs, like modeling transportation networks [10], [11] or applications in biology, e.g., modeling dynamic protein-protein interactions [12], [13], and neural brain networks [14].

Contributions: We introduce TGLib, an open-source temporal graph library under the permissive MIT license. Our library focuses on temporal distance and centrality computations and other local and global temporal graph statistics. TGLib is designed for performance and usability by an efficient and modular C++ implementation of the core data structures and algorithms and an easy-to-use Python front-end allowing users and researchers without in-depth (C++) programming...
experience to use our new library.

Additionally, we offer the first implementations of new algorithms for the earliest arrival distance using the temporal graph data structure introduced in [15], a variant of the fastest path algorithm from [16] for shortest temporal paths, a graph data structure introduced in [15], a variant of the algorithms for the earliest arrival distance using the temporal transition time is given as pair \((t, \lambda)\).

Fig. 2. Example of a temporal graph \(G\). At each edge the availability and transition time is given as pair \((t, \lambda)\).

Fig. 3. Example for the static time-respecting representation introduced in [15] of the temporal graphs shown in Figure 2.

II. PRELIMINARIES

A temporal graph \(G = (V, E)\) consists of a finite set of vertices \(V\) and a finite set of directed temporal edges \(E\). A temporal edge \(e = (u, v, t, \lambda)\) consists of the vertices \(u, v \in V\), availability time (or time stamp) \(t \in \mathbb{N}\) and transition time \(\lambda \in \mathbb{N}\), i.e., \(e = (u, v, t, \lambda) \in V \times V \times \mathbb{N} \times \mathbb{N}^2\). We model an undirected temporal graph by a directed temporal graph using a forward- and a backward-directed edge with equal time stamps and traversal times for each undirected edge.

We use \(n = |V|\) and \(m = |E|\) to denote the numbers of vertices and temporal edges, respectively. The arrival time of an edge \(e = (u, v, t, \lambda)\) (at vertex \(v\)) is \(t + \lambda\). We use \(V(G)\) to denote the set of vertices of \(G\), the minimal number of incoming (outgoing) temporal edges over all vertices by \(\delta_{\text{min}}^+(\delta_{\text{min}}^-)\). Similarly, we define \(\delta_{\text{max}}^+(\delta_{\text{max}}^-)\) for the maximal incoming (outgoing) degree. Furthermore, we denote with \(N(u) = \{v \mid (u, v, t, \lambda) \in E\}\) the neighborhood of \(u\). With \(T(G)\), we denote the set of all availability times of edges in \(G\), i.e., \(T(G) = \{t \mid (u, v, t, \lambda) \in E\}\). Given a temporal graph \(G = (V, E)\), it is common to restrict algorithms and computations on \(G\) to a given restrictive time interval \(I = [a, b]\) with \(a, b \in \mathbb{N}\), such that only the temporal subgraph \(G' = (V, E')\) with \(E' = \{(u, v, t, \lambda) \in E \mid t \geq a \text{ and } t + \lambda \leq b\}\) needs to be considered.

III. DESIGN GOALS, ARCHITECTURE, AND TEMPORAL GRAPH DATA STRUCTURES

In the following, we will discuss the design goals and motivations for the high-level architecture of TG LIB. Furthermore, we introduce the data structures used for efficiently representing temporal graphs.

A. General Architecture

Figure 1 shows a high-level view of the general architecture of TG LIB. It consists of two main components: 1) the C++ template library and 2) the Python binding. The C++ library provides generic temporal graph data structures and contains the implementations of temporal graph algorithms. Furthermore, the C++ library offers functionality for IO, filtering, and data cleaning temporal graphs, which are common tasks for real-world temporal graph data set processing.

The Python interface allows non-C++-experts to use the efficient algorithms provided by TG LIB. We used Pybind11\(^1\) for generating the Python binding.

B. Design Goals

We developed TG LIB under the following design goals:

1) Performance and Efficiency: All running-time critical code is written in a modern C++ template library focusing on efficiency. Furthermore, many of the algorithms benefit from straightforward shared-memory parallelization; hence, when possible, we use OpenMP for loop parallelization and take advantage of modern shared-memory parallel processing capabilities. The lean temporal graph data structures are template based and designed for running time and memory efficiency.

2) Usability and Integration: The C++ template library is provided as a platform-independent header-only library that can be easily integrated into existing or new C++ projects. Furthermore, we provide an easy-to-use Python interface with clear workflows for temporal graph algorithms and analysis to make TG LIB widely available to researchers and practitioners.

3) Extendability, Reuseability, and Sustainability: The C++ source code of TG LIB is based on a generic object-oriented modular designed to be easily extendable and reusable. To this end, we use class templates for the temporal graph data structures to allow extension with customized data. For example, weighted temporal graphs or temporal graphs with time-dependent node labels can be realized by providing corresponding edge or node data types. Our implementation is fully documented, and we use unit tests and static code verification to ensure correctness and sustainability [18].

C. Temporal Graph Data Structures

We implemented the temporal graph data structures listed in Table I. The temporal graph data structures are provided as generic class templates. Table I shows the worst-case sizes of the data structures using the implemented default classes. In the following, we discuss the implemented data structures.

\(^1\)https://github.com/pybind/pybind11
### TABLE I
OVERVIEW OF THE TEMPORAL GRAPH DATA STRUCTURES.

| Name                               | Edge Type | Size           | Reference |
|------------------------------------|-----------|----------------|-----------|
| Temporal Edge Stream (STREAM)      | Temporal  | $O(m)$         | [19]      |
| Edge Incidence Lists (ILISTS)      | Temporal  | $O(n + m)$     | [16]      |
| Time-Respecting Static Graph (TRS) | Static    | $O((n + m)^2)$ | [17]      |
| Directed Line Graph (DLG)          | Static    | $O(n^2)$       | [20]      |
| Aggregated Graph (AGGR)            | Static    | $O(n^2)$       | [20]      |

### TABLE II
OVERVIEW OF THE IMPLEMENTED ALGORITHMS.

| Type                  | Algorithm             | Data Structure | Reference |
|-----------------------|-----------------------|----------------|-----------|
| Distance              |                       |                |           |
|                       | Earliest Arrival/     | STREAM         | [19]      |
|                       | Latest Departure      | ILISTS         | [21], [22]|
|                       | Min. Transition Sum/  | STREAM         | [19]      |
|                       | Min. Hops             | ILISTS         | New       |
|                       | Fastest               | STREAM         | [19]      |
|                       | (In/Out-)Degree       | ILISTS         | New       |
|                       | Temporal Closeness    | STREAM/ILISTS/TRS | [24]     |
|                       | Top-k Closeness       | ILISTS         | [16], [23]|
|                       | (Min. duration)       |ilists          |           |
|                       | Temporal Edge Betweenness | DLG     | New       |
|                       | Temporal Katz         | STREAM         | [25]      |
|                       | Temporal PageRank     | STREAM         | [26]      |
|                       | Temporal Walk Centrality | STREAM     | [27]      |
| Global/Local Properties | Burstiness (Edges)    | STREAM/ILISTS  | [28]      |
|                       | Burstiness (Nodes)    | STREAM/ILISTS  | [28]      |
|                       | Temporal Diameter     | STREAM/ILISTS/TRS | [20]   |
|                       | Temporal Efficiency   | STREAM/ILISTS/TRS | [29]    |
|                       | Topological Overlap   | STREAM/ILISTS  | [30], [31]|

1) **Temporal Edge Streams (STREAM):** The temporal graph is given as a sequence of its $m$ edges, chronologically ordered by the availability time of the edges in increasing order, with ties being broken arbitrarily [19]. This representation is often natural when events represented by the edges are sequentially recorded over time. Algorithms for temporal edge streams usually pass over the edges in forward or backward sequential order and are often very efficient. However, the STREAM data structure can be disadvantageous for local computations, e.g., if we are interested in the immediate neighborhood of a single vertex, as we cannot directly access these neighbors.

2) **Edge Incidence Lists (ILISTS):** Here, the temporal graph consists of a list of temporal vertices, and each vertex has a list of temporal edges to its neighbors [16]. The advantage of this representation is the local access to neighbors of a node.

3) **Static Expansions:** Various static representations of temporal graphs offer different trade-offs between the size of the resulting static graph and the loss of temporal information. We implemented the following versions:

   a) **Time-Respecting Static Graph (TRS):** This representation was introduced in [15] and is an improved version of a data structure from [19]. Here, the temporal graph is transformed into a static, i.e., non-temporal, graph. The time-respecting static graph representation $S(G) = (V_s, E_s)$ of a temporal graph $G = (V, E)$ is defined as follows. First, let $V_s(u) = \{(u, t) \mid (u, v, t, \lambda) \in E\}$, and $t_m(w) = \max\{t + \lambda \mid (v, w, t, \lambda)\}$ if $w$ has at least one incoming temporal edge. We define $V'(u) = V_s(u) \cup \{(u, t_m(u))\}$ (or $V'(u) = V_s(u)$ if $u$ does not have an incoming edge) and $V_s = \cup_{v \in V} V'(u)$. For each temporal edge $(u, v, t, \lambda) \in E$, we introduce a with $\lambda$ weighted static edge $((u, t), (v, t'))$ where $t'$ is the smallest arrival time at $v$ larger or equal to $t + \lambda$. Furthermore, for each $u \in V$, the vertices in $V'(u)$ are connected with zero weighted edges in ascending order. Figure 3 shows the TRS of the temporal graph shown in Figure 2.

   b) **Directed Line Graph (DLG):** The directed line graph expansion has been previously used for survivability and reliability analysis [32], [33]. In [7], [17], the authors used the DLG for lifting static graph kernels to the temporal domain. In a recent work [27], the authors use the DLG for algebraic weighted walk counting. Given a temporal graph $G = (V, E)$, the directed line graph $DL(G) = (V', E')$ is the directed graph, where every temporal edge $(u, v, t, \lambda) \in E$ is represented by a vertex $v_{u,t}^\lambda$, and there is an edge from $v_{u,t}^\lambda$ to $v_{v,t}^\lambda$ if $u = v$ and $t + \lambda \leq s$.

   c) **Aggregated Static Graph (AGGR):** Given a temporal graph $G$, removing all time stamps and traversal times, and merging resulting multi-edges, we obtain the aggregated, or underlying static, graph $A(G) = (V, E_s)$ with $E_s = \{(u, v) \mid (u, v, t, \lambda) \in E\}$. The edges can be weighted depending on the number of temporal edges, e.g., using the contact frequency, i.e., $\phi((u, v)) = |\{(u, v, t, \lambda) \in E\}|$. The aggregated graph can be much smaller than the temporal graph as its number of edges is in $O(n^2)$. However, it does not preserve the temporal information of the network.

### IV. IMPLEMENTED ALGORITHMS

Table II gives an overview of the implemented algorithms and the underlying data structures. All implemented algorithms can be restricted to only consider a given time interval $I$ without increasing the running times. In the following, we discuss the implemented algorithms.

#### A. Temporal Paths, Reachability, and Distances

Finding temporal paths, deciding reachability, and determining temporal distances are essential tasks in various applications and scenarios, e.g., in the computation of temporal centrality measures [16], [34], solving time-dependent transportation problems [35–37], or in the simulation and analysis of epidemics [38], [39].

**Definition 1.** A temporal walk in a temporal graph $G$ is an alternating sequence $(v_1, e_1, \ldots, v_k, e_{k+1})$ of vertices and temporal edges connecting consecutive vertices where for $1 \leq i < k$, $e_i = (v_{i-1}, v_i, t_i, \lambda_i) \in E$, and $e_{k+1} = (v_k, v_{k+1}, t_{k+1}, \lambda_{k+1}) \in E$ the time $t_i + \lambda_i \leq t_{i+1}$ holds. A temporal path $P$ is a temporal walk in which each vertex is visited at most once.
For notational convenience, we omit vertices. The length of a temporal walk \( \omega \) is the number of edges it contains, and we denote it with \( |\omega| \). Let \( \omega = (e_1, \ldots, e_l) \) be a temporal walk in a temporal graph \( G \). The starting time of \( \omega \) is \( s(\omega) = t_1 \), the arrival time is \( a(\omega) = t_l + \lambda_\text{f} \), and the duration is \( d(\omega) = a(\omega) - s(\omega) \). Finally, we define \( l(\omega) = \sum_{i=1}^{l} \lambda_i \).

For example, in Figure 2, there are three paths between vertices \( a \) and \( d \). The first one consists of only the edge \( P_1 = ((a, d, 1, 5)) \) and with \( d(P_1) = 5 \). The second is \( P_2 = ((a, b, 2, 1), (b, d, 7, 2)) \) with \( d(P_2) = 7 \). And, path three \( P_3 = ((a, b, 5, 2), (b, d, 7, 2)) \) with \( d(P_3) = 4 \).

There are several optimality criteria for temporal paths used in the literature, and we distinguish the following.

**Definition 2.** Let \( G \) be a temporal graph and \( P \) be the set of all temporal paths in \( G \). A \((s, z)\)-path\(^2 \) \( P \in P \) is

- an earliest arrival path if there is no other \((s, z)\)-path \( P' \in P \) with \( a(P') < a(P) \),
- a latest departure path if there is no other \((s, z)\)-path \( P' \in P \) with \( s(P') > s(P) \),
- a minimum duration, or fastest, path if there is no other \((s, z)\)-path \( P' \in P \) with \( d(P') < d(P) \),
- a shortest path if there is no other \((s, z)\)-path \( P' \in P \) with \( l(P') < l(P) \), and
- a minimum hops path if there is no other \((s, z)\)-path \( P' \in P \) with \( |P'| < |P| \).

For the example in Figure 2, \( P_3 \) is the only fastest \((a, d)\)-path. Notice that the subpath \( P_3'' = ((a, b, 5, 2)) \) is not a fastest \((a, b)\)-path. The only fastest \((a, b)\)-path consists of edge \((a, b, 2, 1)\) and has a duration of one.

We provide algorithms for determining the temporal distances of Definition 2 for the different data structures, see Table II. The reason for providing the algorithms for different data structures is that depending on the topology of the graph, different algorithms can be more efficient than others, see, e.g., [15], [16], [19]. We additionally introduce a new shortest paths algorithm for the ILISTS data structure and a new earliest arrival algorithm for the TRS data structure\(^3 \). For each temporal distance, we provide algorithms to obtain an optimal temporal path and the temporal diameter, which is defined as the maximum optimal temporal distance between any two (reachable) vertices in the network [40].

**B. Centrality Measures**

The centrality of a node (edge) in a network quantifies its structural importance. Various functions can be used to measure node (edge) centrality by assigning values corresponding to some measurement of importance to each node (edge), where the informative value must be assessed based on a research question. For introductions of centrality approaches, see, e.g., [41]–[44]. TGLIB provides the following centrality measures designed explicitly for temporal graphs.

1) **Temporal Closeness:** Due to the differences in reachability and optimality in temporal graphs, several versions of temporal closeness have been suggested, see, e.g., [16], [45], [46]. Using the optimal distance computations for earliest arrival paths, fastest paths, etc., we provide different versions of harmonic temporal closeness defined as

\[
C(u) = \sum_{v \neq u \in V} \frac{1}{d'(u, v)},
\]

where \( d' \) is a temporal distance function and \( 1/\infty = 0 \) for non-reachable vertices. Furthermore, we provide the top-\( k \) approach introduced in [16] for finding the \( k \) highest closeness centrality values and the corresponding vertices. The authors of [16] only introduced their top-\( k \) algorithm for harmonic closeness wrt. to the minimum duration distance. We provide an additional (new) implementation of the top-\( k \) algorithm for the minimum transition times distance.

2) **Temporal Edge Betweenness:** Similar to the static edge betweenness [47], the temporal edge betweenness is an edge centrality measure and quantifies the importance of the temporal edges in terms of the shortest temporal paths crossing the temporal edge. It can be computed by counting the shortest paths in the directed line graph representation due to the one-to-one mapping of walks in \( G \) and \( D.L(G) \) [17].

3) **Temporal Katz Centrality:** The Katz centrality introduced in [48] measures vertex importance in terms of the number of random walks starting (or arriving) at a vertex, down-weighted by their length. The authors of [25], [49] adapt the walk-based Katz centrality to temporal graphs.

4) **Temporal PageRank Centrality:** Rozenshtein and Gionis [26] incorporate the temporal character in the definition of the static PageRank originally introduced by [50]. They obtain a temporal PageRank by replacing walks with temporal walks.

5) **Temporal Walk Centrality:** Temporal Walk Centrality is a recently proposed centrality measure that aims to rank the vertices according to their ability to obtain and distribute information [27].

**C. Further Local and Global Properties**

Moreover, we implemented the following local and global temporal graph properties.

1) **Burstiness:** Burstiness measures how much a sequence of contacts \( \tau \) (of a single node or between a pair of nodes) deviates from the memoryless random Poisson process [20]. It is defined as

\[
B(\tau) = \frac{\sigma_\tau - m_\tau}{\sigma_\tau + m_\tau} \in [-1, 1],
\]

where \( \sigma_\tau \) and \( m_\tau \) denote the standard deviation and mean of the inter-contact times \( \tau \), respectively [28]. A value close to one indicates a very bursty sequence, and a value close to minus one a more periodic sequence.

2) **Temporal Clustering Coefficient:** The temporal clustering coefficient [29] is defined as

\[
C_G(u) = \frac{\sum_{e \in C(u)} \pi_e(u)}{|T_G| \left( \binom{|N(u)|}{2} \right)},
\]

2 We use \( z \) instead of \( t \) as the target vertex because we use \( t \) to denote a time stamp.  
3 A formal description of the algorithms with correctness and complexity proofs will be in an extended version of this paper at https://arxiv.org/abs/2209.12587.
where $\pi_t(u) = \{(v, w, t, \lambda) \in \mathcal{E} \mid v, w \in N(u)\}$.

3) Temporal Efficiency: The temporal efficiency is a global statistic based on the temporal closeness values of the nodes [29]. It is defined as

$$T_{\text{eff}}(\mathcal{G}) = \frac{1}{n(n-1)} \sum_{u \in V(\mathcal{G})} \sum_{v \in V(\mathcal{G})} \frac{1}{d'(u, v)},$$

with $d'(u, v)$ being a temporal distance and $1/\infty = 0$ in case of non-reachable vertices.

4) Topological Overlap: The topological overlap of a node is defined as

$$T_{\text{lo}}(u) = \frac{1}{T(\mathcal{G})} \sum_{t=1}^{T(\mathcal{G})} \frac{\sum_{v \in N(u)} \phi_{tu}^t \phi_{uv}^{t+1}}{\sqrt{\sum_{v \in N(u)} \phi_{uv}^t \sum_{w \in N(u)} \phi_{wu}^{t+1}}},$$

where $\phi_{uv}^t = 1$ iff there exists a temporal edges between $u$ and $v$ at time $t$ and zero otherwise [30], [31]. In case that the denominator equals one, we define $T_{\text{lo}}(u) = 1$.

The topological overlap is in the range between zero and one. A value close to one means many edges change between consecutive time steps, and a value close to one means there are often only a few changes.

V. COMPARISON TO RELATED SOFTWARE

There are several popular graph libraries designed for conventional static graphs, e.g., Networkit [51], OGDF [52], LEMON [53], or Boost graph [54]. However, they are not designed to handle the peculiarities of temporal graphs, e.g., they do not support algorithms that respect the temporal restrictions in temporal walks and paths. The SNAP library provides various algorithms for temporal graphs, like counting specific temporal motifs [55]. We are not aware of a dedicated C++ library for temporal graphs. For Python, the Teneto library is a dedicated temporal graph library supporting various analytical methods [14]. However, the library focuses on analyzing small networks obtained from fMRI brain scans and does not support transition times on the edges. Moreover, it does not perform well on mid-size to large temporal graphs as its mainly based on Python code and matrix-based computations. Finally, Teneto is published under the, compared to the MIT license, more restrictive GNU GPL3 license.

VI. EXAMPLE USE-CASE

We load two real-world data sets and compare different variants of temporal closeness. The first one is the AskUbuntu, a network consisting of interactions on the stack exchange website Ask Ubuntu [55]. The second data set is the Enron email network between employees of a company [56]. To obtain the basic statistics, we load the temporal graph and call the get_statistics function (see Listing 1). Note that Teneto is unable to load these data sets due to its temporal graph representation as a sequence of adjacency matrices and the resulting out-of-memory error (6.13 EiB for Enron). Next, we compute the closeness centrality with respect to minimum duration distance and earliest arrival time. We compare the obtained rankings using Kendall’s $\tau$ rank correlation using Scipy’s. We obtain correlations of 0.79 for AskUbuntu and 0.94 for Enron, showing strong correlations in both graphs.

```
1 import pytgalib as tgl # tgalib
2 import scipy.stats as ss # for correlation
3 tgs = tgl.load_ordered_edge_list("datasetname")
4 stats = tgl.get_statistics(tgs)
5 print(stats)
6 closeness_fastest = tgl.temporal_closeness(tgs, Distance_Type.Fastest)
7 closeness_distance = tgl.temporal_closeness(tgs, Distance_Type.Earliest_Arrival)
8 tau, p_value = ss.kendalltau(closeness_fastest, closeness_distance)
```

Listing 1. Example Python use-case code

VII. OPEN-SOURCE DEVELOPMENT AND LICENSE

TGLIB free software licensed under the permissive MIT License. It is available at https://github.com/gpublic/tglib. We aim to encourage a diverse community, including network researchers, data mining practitioners, and algorithm engineers, to use TGLIB and contribute to the open-source development.

VIII. CONCLUSION AND FUTURE WORK

We introduced the open-source toolkit TGLIB, a C++ library for efficient temporal graph analysis featuring an easy-to-use and accessible Python front-end. So far, we have implemented a wide range of algorithms for distance, centrality, and analytical computations based on various efficient temporal graph data structures. TGLIB offers researchers, practitioners, and students convenient access to temporal graph algorithms. Furthermore, it offers a unified and accessible approach for reproducibility and comparability. We are actively working to integrate further and future methods and algorithms into our library. Using the permissive MIT license, we hope that TGLIB will be used and extended by the temporal graph community.

ACKNOWLEDGMENT

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy – EXC-2047/1 – 390685813. This research has been funded by the Federal Ministry of Education and Research of Germany and the state of North-Rhine Westphalia as part of the Lamarr-Institute for Machine Learning and Artificial Intelligence, LAMARR22B.

REFERENCES

[1] J. Candia, M. C. González, P. Wang, T. Schoenharl, G. Madey, and A.-L. Barabási, “Uncovering individual and collective human dynamics from mobile phone records,” Journal of physics A: mathematical and theoretical, vol. 41, no. 22, p. 224015, 2008.

[2] J.-P. Eckmann, E. Moses, and D. Sergi, “Entropy of dialogues creates coherent structures in e-mail traffic,” Proceedings of the National Academy of Sciences, vol. 101, no. 40, pp. 14333–14337, 2004.

[3] P. Holme, C. R. Edling, and F. Liljeros, “Structure and time evolution of an internet dating community,” Social Networks, vol. 26, no. 2, pp. 155–174, 2004.

[4] C. Avin, M. Koucký, and Z. Lotker, “How to explore a fast-changing world (cover time of a simple random walk on evolving graphs),” in International Colloquium on Automata, Languages, and Programming. Springer, 2008, pp. 121–132.

4 https://scipy.org/
