Provenance in Temporal Interaction Networks

Chrysanthi Kosyfaki and Nikos Mamoulis
Department of Computer Science and Engineering, University of Ioannina, Greece
{xkosifaki,nikos}@cs.uoi.gr

***Abstract*—In temporal interaction networks (TINs), vertices correspond to entities, which exchange data quantities (e.g., money, bytes, messages) over time. We study the problem of continuously tracking the origins of quantities at network vertices, as interactions take place over time. We target applications, such as financial exchange networks, where the selected transferred units at each interaction are not specified. We investigate several quantity selection policies that apply to different application scenarios. For each policy, we propose space- and time-efficient meta-data propagation mechanisms for continuously tracking provenance at vertices. For the hard case of proportional selection policy, we reduce the cost of tracking in practice, by either (i) limiting provenance tracking to a subset of vertices or groups of vertices, or (ii) tracking provenance only for quantities that were generated in the near past or limiting the provenance data in each vertex by a budget constraint. Our experimental evaluation on real datasets demonstrate the efficiency and scalability of our techniques compared to baseline approaches that extend flow computation algorithms to track provenance.**

I. Introduction

Many real world applications can be represented as *temporal interaction networks* (TINs) [24], where vertices correspond to entities that *exchange data* over time. Examples of such graphs are financial exchange networks and transportation networks. The TIN can be viewed as a *stream of interactions*, where each interaction r is characterized by a *source* vertex $r.s$, a *destination* vertex $r.d$, a timestamp $r.t$ and the transferred quantity $r.q$ (e.g., money, passengers, etc.). For example, the sequence of bitcoin transactions form a TIN, where the vertices are user addresses, exchanging bitcoins over time.

We study a provenance problem in TINs; the objective is to know the origins of quantities at the vertices of the network over time. As data are transferred from one vertex to another, the origins of quantities at a vertex v are not necessarily direct neighbors of v, hence the problem is not easy to solve. In general, provenance in graphs is a well-studied problem with numerous applications (auditing, debugging, reproducibility, explainability, etc.) [23].

We adopt a data propagation model used in previous work on flow computation in temporal networks [2], [31]. We assume that each vertex v has a *buffer* B_v (e.g., bitcoin wallet), wherein it keeps all incoming quantities to v. Naturally, the buffer B_v changes over time. Specifically, each interaction r from a vertex $r.v$ to a vertex $r.u$ *transfers* $r.q$ units from $B_{r.v}$ to $B_{r.u}$ at time $r.t$. If $B_{r.v}$ has less than q units by time $r.t$, then the difference is *generated* at $r.v$ before being transferred to $r.u$. In a financial exchange network, quantity generation means that new assets are brought from external sources (e.g., a user mines bitcoins). In a road network, new quantities are cars entering the network from a given location. If $B_{r.v}$ has more than $r.q$ units, $r.q$ units should be *selected* from $B_{r.v}$ to be transferred to $B_{r.u}$. We observe that in most applications, the recorded interactions do not give us any information about the selection of transferred quantity units. In addition, the data units which move in the network may not be easy to "tag", therefore tracing back their lineage could be hard.

Example 1: Consider a financial TIN (e.g., Bitcoin transaction network) where vertices correspond to accounts and interactions are money transfers. Accounts do not have explicit information about the provenance of their balances. At the same time, a transaction r which transfers less than the sender's balance $B_{r.v}$ does not specify which part of the balance is transferred as the monetary units are not tagged/marked. Therefore, to trace the origin of account balances over time, we have to rely on assumptions about the selections in transfers.

Example 2: In a transportation network, people or vehicles (quantities) move between districts (vertices). Transportation authorities monitor the volumes of objects that move between districts (interactions), however, they do not track data about individuals due to privacy constraints.[1] To trace the origin of moving objects currently in a region, one has to assume a selection policy for object transfer.

Our contribution We target applications, such as the above, where the recorded interactions do not capture how the transfered units are selected. Given this, we define and study alternative *selection policies* for quantity transfer for different application scenarios. A policy prioritizes quantities based on the time they were first generated at their origins, or on the order they were added to the buffer, or may select quantities proportionally based on their origins. We propose algorithms that *proactively* create and propagate *lightweight provenance information* in the TIN for the generated quantities, as they are transferred through the network (i.e., our techniques fall into the class of meta-data propagation approaches [23]). Our approach updates provenance information after each interaction; hence, we know the origins of quantities at vertices *at any time*, as interactions are processed in a streaming fashion. Our algorithms are space- and time-efficient, since we do not "tag" each individual quantity unit that moves in the network, but we *group* together units of common origin.

Note that previous works on flow computation in temporal networks [2], [31], [45] aim at just computing the flow from a source to a destination and do not compute provenance information. For flow computation, the selection of transferred

[1] https://www.its.dot.gov/factsheets/privacy.htm

units does not make any difference (it is only the quantity what matters, not where it came from). To our knowledge, there is no previous work that studies data provenance in TINs, especially in situations where transferred data cannot be tagged, or when tagging is too expensive. Our approach is orthogonal to and can be applied in combination with any flow computation method [2], [31] to track flow provenance.

Applications Provenance tracking in TINs finds several applications. In a financial network, by continuously monitoring the provenance of account balances, we can identify accounts that receive amounts indirectly via intermediaries, which may indicate illegal activities (e.g. "smurfing"). Such analyses are routinely done by financial intelligent units (e.g. fiubelize.org, www.jfiu.gov.hk). We present a detailed use case in Section VII. In a transportation network, studying the provenance of problems (e.g., traffic) can help solving them. As a concrete TIN provenance application example, consider one of the TINs used in our experiments, published by www1.nyc.gov, which captures the transfers of passengers by taxi between NYC districts on 2019.01.01. Note that there are no identifiers of passengers in the data due to privacy constraints; we assume that passengers move between districts in a FIFO manner. Figure 1 shows the number of passengers that are accumulated in East Village (vertex #79) from other districts (directly or indirectly) over time (i.e., after each interaction). The provenance distribution of East Village visitors over time (shown as pie charts) can be used for social/demographics analysis or (location-aware) marketing. For example, if a shop owner knows that many passengers originating from upscale districts accumulate into the shop's district, the owner can design promotion campaigns for wealthy customers.

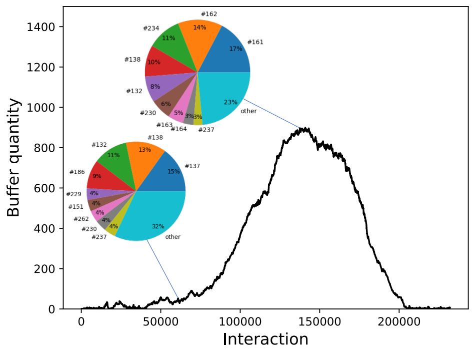


Fig. 1. Buffered (accumulated) passengers at vertex #79 (East Village) on 2019.01.01, after each interaction in our Taxis Network

Outline Section II reviews related work on temporal networks and data provenance. In Section III, we formally define the provenance problem in TINs that we study in this paper, after presenting in detail a data propagation model from previous work [31], which we extend to capture provenance. Section IV presents the different information propagation policies and the corresponding provenance tracking algorithms that we propose. We analyze the space and time complexity of provenance tracking under each policy and find that the *proportional propagation policy* is infeasible for large graphs because it has a $O(|V|^2)$ space complexity and bears a $O(|V|)$ cost per interaction, where $|V|$ is the number of vertices.

In view of this, in Section V, we propose scaleable techniques for provenance tracking under the proportional propagation policy. Our *selective provenance tracking* approach maintains provenance data only from a designated subset of k vertices, which are of interest to the analyst, reducing the space complexity to $O(k \cdot |V|)$ and the time complexity to $O(k)$ per interaction. The *grouped provenance tracking* approach tracks provenance from k groups of vertices instead of individual vertices (e.g., categories of financial entities or accounts). This approach also has $O(k \cdot |V|)$ space complexity and $O(k)$ cost per interaction. We also propose two techniques that limit the scope of provenance tracking from all (individual) vertices. The first approach limits provenance tracking up to a certain time in the past from the current interaction (i.e., a time-window approach). The second approach allocates a provenance budget to each vertex. Both techniques save resources, while providing some guarantees with respect to either time or importance of the tracked provenance information.

In Section VI, we extend our propagation algorithms for provenance annotations to capture not only the origins of the generated data, but the routes (i.e., the paths) that they travelled along in the graph until they reached their destinations.

Section VII experimentally evaluates the runtime and memory requirements of our methods on real TINs with different characteristics. The results show the scalability and limitations of the different selection policies and the corresponding propagation algorithms for provenance data. We compare our approach against a baseline method that divides the quantities into units and tracks their origins; while this method solves our problem, it is extremely slow to be practical. We also compared our method against a simple approach that does not track provenance but infers it from global statistics, which, however, is very inaccurate on large graphs. Finally, we include a use case of provenance tracking in financial networks, which demonstrates the utility of provenance in TINs. Finally, Section VIII concludes the paper with a discussion about future work.

II. Related Work

There has been a lot of research in data provenance over the years [3], [11], [12], [33], [36], [38], [40], [42]. However, we are the first to study the problem of tracking the origins of quantities that flow in temporal interaction networks. In this section, we summarize representative works in temporal networks and provenance tracking.

A. Temporal Networks

Temporal networks model network dynamics over time, such as time-dependent edge capacities [2], [45] and evolving structure [34], [35]. Kosyfaki et al. [30] studied the problem of finding recurrent flow path patterns in a TIN, during time windows of specific length. Motif discovery in TINs was first studied in [37]. Pattern detection in temporal networks with an application in social network analysis was also studied in [4],

[48]. A related problem is how to measure the total quantity that flows between two specific vertices in a temporal network [2] or in a TIN [31], based on the data propagation model that we use in this paper. Zhou et al. [47] study the problem of dynamically synthesizing realistic TINs by learning from log data. Provenance in TINs can be combined with structural and flow analysis, in order to explain the existence of patterns. A network protocol that mitigates synchronization problems when updating Bitcoin's blockchain is proposed in [13]; this problem is not relevant to data propagation in TINs.

B. Theory and Applications of Provenance

Buneman et al. [6] were the first who defined and studied the problem of provenance in database systems. An annotation mechanism for where-provenance was proposed in [7] and implemented in DBNotes [5]. As query operators (select, project, join, etc.) are executed, annotations are *propagated* to eventually reach the query output tuples. Geerts et al. [18] proposed another annotation-based model for the manipulation and querying of both data and provenance, which allows annotations on *sets of values* and for effectively querying how they are associated. There are important differences between our work and provenance approaches for database systems. First, the TIN graphs that we examine are very large (as opposed to small query graphs) and we track provenance for any vertex in them (i.e., we do not distinguish between input and output vertices). Second, the data transfer model between vertices in TINs is very different compared to data transfer in query graphs. Third, interactions can happen in any order in our TINs, as opposed to query graphs, where edges have a specific order (query graphs are typically DAGs).

Data Provenance has also been studied in social networks [3]. An important application is to detect where from a rumor has started before spreading through the internet. Gundecha et al. [21] represent social networks as directed graphs and try to recover paths to find out how information spread through the network by isolating important nodes, based on their centrality. Taxidou et al. [46] studied provenance within an information diffusion model, based on the W3C Provenance Data Model (https://www.w3.org/TR/prov-dm/). These approaches are not applicable to TINs, because, in social networks, information *is copied and diffused*, whereas in TINs data are *buffered* and *moved* (i.e., not copied) from one vertex to another.

Savage et al. [44] propose a stochastic packet marking mechanism that can be used for probabilistic tracing of packet-flooding attacks in the Internet. We target a more generic provenance problem in TINs, where we consider information propagation based on several alternative policies. Moreover, we aim at exact provenance tracking wherever possible. Finally, Dey et al. [16] propose a provenance mechanism for regular path queries (RPQs) in graphs which witnesses the fact that pairs of nodes v,u which are connected by a path that qualifies a given regular expression. Provenance methods for PRQs do not apply on TINs, but on static, edge-labeled graphs and they have a different goal and approach (enriching RPQ computation by additional DBMS operations that generate the desired witnesses).

C. Provenance Systems

Over the years, a number of systems for provenance tracking have been developed, mainly to serve the need of efficiently storing and managing the annotation data. Chapman et al. [9] propose a factorization technique, which identifies and unifies common query evaluation subtrees for reducing the provenance storage requirements. Heinis and Alonso [22] represent workflow provenance mechanisms as DAGs and compress DAGs with common nodes, in order to save space.

Several systems [1], [20] have been developed to support the answering of *data provenance questions*, where the objective is to find how a data element has appeared in the query result. Karvounarakis et al [27] developed ProQL, a query language which can be used to detect errors and side effects during the updates of a database. ProQL takes advantage of the graph representation and path expressions to simplify operations involving traversal and projection on the provenance graph. A provenance query language and algorithms for datalog programs, supporting tracking for selected graph subsets were proposed in [14]. Deutch et al. [15] reduce the granularity of provenance tracking in order to make it feasible on large graphs. Titian [25] adds provenance support to Spark, aiming at identifying errors during query evaluation.

Glavic et al. [19] present a system for provenance tracking in data stream management systems (DSMS). They propose an *operator instrumentation* model, which annotates data tuples that are generated or propagated by the streaming operators with their provenance. They also propose an alternative approach (called *replay lazy*), which uses the original operators and, whenever provenance information is needed, the approach replays query processing on the relevant inputs through a instrumented copy of the network (hence, data processing and provenance computation are decoupled). We also propose space-economic models for tracking provenance. However, our input graphs (TINs) are larger and different than DSMS graphs and our propagation models consider the transfer of quantities between vertices as a result of a stream of interactions.

Provenance has also been studied in blockchain systems especially after the huge success of Bitcoin. In [41], a secure and efficient system called LineageChain is implemented on top of Hyperledger (https://www.hyperledger.org), for capturing provenance during contract execution and safely storing it in a Merkle tree. The objective of this work is to explicitly track the historical states of the blockchain, by recording read-write dependencies between successive states. Unlike our work, this approach does not explicitly track the origins of the account balances, but its main goal is to be able to efficiently reconstruct the balances of accounts at any time in the past.

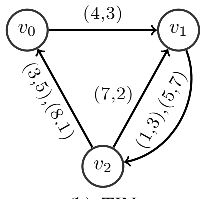
III. Definitions

In this section, we formally define temporal interaction networks (TINs) and data propagation in TINs. Then, we define the provenance problem that we study in this paper. Table I summarizes the notation used frequently in the paper.

Definition 1 (Temporal Interaction Network): A temporal interaction network (TIN) is a directed graph $G(V,E,R)$. Each edge (v,u) in E carries the interactions from vertex v to vertex u. R denotes the set of interactions on all edges of E, ordered by time. Each interaction $r \in R$ is characterized by a quadruple $\langle r.s, r.d, r.t, r.q\rangle$, where $r.s \in V$ ($r.d \in V$) is the source (destination) vertex of the interaction, $r.t \in \mathbb{R}^+$ is the time when the interaction took place and $r.q \in \mathbb{R}^+$ is the transferred quantity from vertex $r.s$ to $r.d$, due to r.

$r.s$	$r.d$	$r.t$	$r.q$
v_1	v_2	1	3
v_2	v_0	3	5
v_0	v_1	4	3
v_1	v_2	5	7
v_2	v_1	7	2
v_2	v_0	8	1

(a) Interactions



(b) TIN

Fig. 2. A set of interactions and the corresponding TIN

Figure 2 shows the set R of interactions in a TIN and the corresponding graph. For example, sequence $\{(1,3),(5,7)\}$ on edge (v_1,v_2) means that v_1 transferred to v_2 a quantity of 3 units at time 1 and then 7 units at time 5. The corresponding interactions in R are $\langle v_1, v_2, 1, 3\rangle$ and $\langle v_1, v_2, 5, 7\rangle$.

TABLE I
TABLE OF NOTATIONS

Notation	Description
$G(V,E,R)$	TIN (vertices, edges, interactions)
$r.s$ ($r.d$)	source (destination) vertex of interaction $r \in R$
$r.t$	time when interaction $r \in R$ took place
$r.q$	transferred quantity during interaction $r \in R$
B_v, $\|B_v\|$	buffer of vertex v, total quantity in B_v
$O(t,B_v)$	origin (provenance) data for the quantity at B_v by time t
$(\tau.o, \tau.q)$	quantity $\tau.q$ originating from $\tau.o$ in $O(t,B_v)$
$\mathbf{p}_v$	provenance vector of a vertex $v \in V$

We consider all interactions R in the TIN *in order of time*, i.e. as a stream, and assume that each vertex $v \in V$ has a *buffer* B_v, which stores the total quantity that has flown into v but has not been transferred yet to other vertices via outgoing interactions from v. $|B_v|$ denotes the total quantity in B_v.

As an effect of an interaction $\langle r.s, r.d, r.t, r.q\rangle$, vertex $r.s$ transfers a quantity of $r.q$ to vertex $r.d$. Quantity $r.q$ (or part of it) could be data that have been accumulated at vertex $r.s$ by time $r.t$, or $r.q$ could (partially) be generated at $r.s$ [31]. More specifically, we distinguish between two cases:

- $|B_{r.s}| \leq r.q$. In this case, *all* units from $B_{r.s}$ are transferred to $B_{r.d}$ due to the interaction. In addition, $r.q - |B_{r.s}|$ units are *generated* by the source vertex $r.s$ and transferred to $B_{r.d}$. Hence, $|B_{r.s}|$ becomes 0 and $|B_{r.d}|$ is increased by $r.q$.
- $|B_{r.s}| > r.q$. In this case, $r.q$ units are *selected* from $B_{r.s}$ to be transferred to $B_{r.d}$. Hence, $|B_{r.s}|$ is decreased by $r.q$ and $|B_{r.d}|$ is increased by $r.q$.

Algorithm 1 Propagation algorithm in a TIN

```
Require: TIN $G(V,E,R)$
1: for each $v \in V$ do
2:     $|B_v| = 0$                              ▷ Initialize buffers
3: end for
4: for each interaction $r \in R$ in order of time do
5:     $q = \min\{r.q, B_{r.s}\}$               ▷ relayed quantity from $B_{r.s}$
6:     $|B_{r.s}| = |B_{r.s}| - q$              ▷ decrease by $q$
7:     $|B_{r.d}| = |B_{r.d}| + r.q$            ▷ increase by $r.q$; $r.q-q$ is newborn
8: end for
```

Algorithm 1 is a pseudocode of the data propagation procedure. Interactions in R are processed in order of time, i.e., as a stream. For each interaction $r \in R$, we first determine the *relayed* quantity q from the buffer of the source vertex $r.s$ (Line 5). This quantity cannot exceed the currently buffered quantity $|B_{r.s}|$ at $r.s$. Line 6 decreases $B_{r.s}$, accordingly. The target node's buffer $B_{r.d}$ is increased by $r.q$ (Line 7). If $r.q > q$, a new quantity $r.q - q$ is *born* by the source vertex $r.s$ to be transferred to $B_{r.d}$ as part of $r.q$. Table II shows the changes in the buffers of the three vertices in the example TIN (Figure 2), during the application of Algorithm 1. Values in parentheses are newborn quantities at $r.s$, transferred to $r.d$.

TABLE II
CHANGES AT BUFFERS AT EACH INTERACTION

$r.s$	$r.d$	$r.t$	$r.q$	$\|B_{v_0}\|$	$\|B_{v_1}\|$	$\|B_{v_2}\|$
v_1	v_2	1	3	0	0	3 (3)
v_2	v_0	3	5	5 (2)	0	0
v_0	v_1	4	3	2	3	0
v_1	v_2	5	7	2	0	7 (4)
v_2	v_1	7	2	2	2	5
v_2	v_0	8	1	3	2	4

Definition 2 formally defines the provenance problem that we study in this paper.

Definition 2 (Provenance Problem): Given a TIN $G(V,E,R)$, at any time moment t and at any vertex $v \in V$ determine the origin(s) $O(t,B_v)$ of the total quantity accumulated at buffer B_v by time t. $O(t,B_v)$ is a set of $(\tau.o, \tau.q)$ tuples τ, such that each quantity $\tau.q$ was generated by vertex $\tau.o$ and $\sum_{\tau \in O(t,B_v)} \tau.q = |B_v|$.

At any time t, the objective is to know the *origin* vertices which have generated the quantities that have been accumulated at buffer B_v, for any vertex v.

IV. SELECTION POLICIES AND PROVENANCE

For each interaction $r \in R$, the selection policy in the case where $|B_{r.s}| > r.q$ affects the provenance of the quantities that are accumulated at vertices, because the content of $B_{r.s}$ may originate from different vertices. In this section, we present possible selection policies that apply on different application scenarios and, for each policy, we present annotation mechanisms that can be used to trace the provenance of the quantities accumulated at the vertices of the TIN.

A. Selection based on generation time

The first class of selection policies is based on the time when the candidate quantities to be transferred are generated.

We will first discuss the *least recently born* (LRB) selection policy. To implement this approach, any generated quantity should be marked with the vertex v that generates it and the timestamp t when it is generated. Hence, during the course of the algorithm, each buffer B_v is modeled and managed as a set of (o, t, q) triples, where o is the origin of quantity q and t is the birthtime of q. At interaction r, if $|B_{r.s}| > r.q$, the triples in $B_{r.s}$ with the smallest birthtimes whose quantities sum up to $r.q$ are selected and transferred to $B_{r.d}$. To facilitate selection, a min-heap organizes the triples in each buffer B_v.

Algorithm 2 describes provenance tracking under LRB. For the current interaction $r \in R$, we keep in variable $resq$ the *residue quantity*, which has yet to be transferred from $r.s$ to $r.d$. Initially, $resq = r.q$. While $q > 0$ and $B_{r.s}$ is not empty, we find the least recently born triple τ in $B_{r.s}$. If $\tau.q > q$, only *part* of the quantity in the triple is transferred to $B_{r.d}$, hence, we *split* τ, by keeping it in $B_{r.s}$ and reducing $\tau.q$ by q and initializing a new triple τ' with the same origin and birthtime as τ and quantity q. The new triple is added to $B_{r.d}$. If $\tau.q \leq q$, we transfer the entire triple τ from $B_{r.a}$ to $B_{r.d}$. If $B_{r.s}$ becomes empty and $resq > 0$, then this means that it was $|B_{r.s}| < r.q$ in the beginning, so we *generate* a newborn triple τ' with the residue quantity $resq$, having as origin vertex $r.s$ and marked to be generated at time $r.t$.

Algorithm 2 Least-recently born selection policy

```
Require: TIN $G(V, E, R)$
 1: for each $v \in V$ do
 2:     $B_v = \emptyset$; $|B_v| = 0$                          ▷ Initialize buffers
 3: end for
 4: for each interaction $r \in R$ in order of time do
 5:     $resq = r.q$                          ▷ residue quantity to be transferred
 6:     while $resq > 0$ and $|B_{r.s}| > 0$ do
 7:         $\tau$ = least recent triple in $B_{r.s}$        ▷ top element in heap
 8:         if $\tau.q > resq$ then                                    ▷ split $\tau$
 9:             $\tau'.o = \tau.o$; $\tau'.t = \tau.t$; $\tau'.q = resq$;       ▷ new triple
10:             add $\tau'$ to $B_{r.d}$;
11:             $\tau.q = \tau.q - r.q$;                              ▷ update $\tau$
12:             $resq = 0$;                               ▷ transfers completed
13:         else
14:             remove $\tau$ from $B_{r.s}$ and add it to $B_{r.d}$;
15:             $resq = resq - \tau.q$                ▷ update residue quantity
16:         end if
17:     end while
18:     if $resq > 0$ then                      ▷ newborn quantity and triple
19:         $\tau'.o = r.s$; $\tau'.t = r.t$; $\tau'.q = resq$;
20:         add $\tau'$ to $B_{r.d}$;
21:     end if
22: end for
```

Table III shows the changes in the buffers of the vertices after each interaction in our running example. The quantities in the buffers are shown as (o, t, q) triples.

By running Algorithm 2, we can have at any time t the set of vertices that contribute to a vertex v by time t and the corresponding quantities (i.e., the solution to Problem 2). In other words, the heap contents for each vertex v at time t corresponds to $O(t, B_v)$. Finally, to implement the *most recently born* (MRB) selection policy, we should change Line 7 of Algorithm 2 to "τ = most recent triple in $B_{r.s}$" and organize each buffer as a max-heap (instead of a min-heap).

TABLE III
CHANGES AT BUFFERS (OLDEST-FIRST POLICY)

$r.s$	$r.d$	$r.t$	$r.q$	B_{v_0}	B_{v_1}	B_{v_2}
v_1	v_2	1	3	∅	∅	{(1,1,3)}
v_2	v_0	3	5	{(1,1,3),(2,3,2)}	∅	∅
v_0	v_1	4	3	{(2,3,2)}	{(1,1,3)}	∅
v_1	v_2	5	7	{(2,3,2)}	∅	{(1,1,3),(1,5,4)}
v_2	v_1	7	2	{(2,3,2)}	{(1,1,2)}	{(1,1,1),(1,5,4)}
v_2	v_0	8	1	{(1,1,1),(2,3,2)}	{(1,1,2)}	{(1,5,4)}

Application The LRB policy is applicable when the generated quantities lose their value over time (or even expire), which means that the vertices prefer to keep the most recently generated data. The MRB policy is relevant to applications, where quantities have *antiquity value*, i.e., they become more valuable as time passes by. For example, in a loans-exchange network, the generation time of a loan affects its value as it determines the owed interest; hence, it is reasonable to prioritize loan transfers based on generation time.

Complexity Analysis In the worst case, each interaction r increases the total number of triples by one (i.e., by splitting the last transferred triple or by generating a new triple), hence, the space complexity is $O(|R|)$, where $|R|$ is the number of processed interactions so far. In terms of time, each interaction accesses in the worst case the entire set of triples at vertex $r.s$. This set is $O(|R|)$ in the worst case, but we expect it to be $O(|R|/|V|)$; for each triple in the set, we update two priority queues in the worst case (i.e, by triple transfers) at an expected cost of $O(\log |R|/|V|)$. Hence, the overall expected cost (assuming an even distribution of triples) is $O(|R| \cdot |R|/|V| \cdot \log |R|/|V|) = O(|R|^2/|V| \log |R|/|V|)$.

B. Selection based on order of receipt

Another policy would be to select the transferred quantities in order of their receipt. Specifically, the content of each buffer B_v is modeled and managed as a set of (o, q) pairs, where o is the vertex which generated q. These pairs are organized based on the order by which they have been added to B_v. If, for the current transaction r, $|B_{r.s}| > r.q$, the last (or the first) quantities in $B_{r.s}$ which sum up to $r.q$ are selected and added to $B_{r.d}$. To implement this policy, each buffer is implemented as a FIFO (or LIFO) queue, hence, it is not necessary to keep track of birthtimes. The algorithm is identical to Algorithm 2, except that Line 7 becomes "least recently added triple in $B_{r.s}$" in the FIFO policy and "most recently added triple in $B_{r.s}$" in the LIFO policy. Table IV shows the changes in the buffers after each interaction when the LIFO policy is applied.

Application The FIFO policy is used in applications where the buffers are naturally implemented as FIFO queues (pipelines, traffic networks). The LIFO policy applies when the accumulated quantities are organized in a stack (e.g., cash registers, wallets) before being transferred.

Complexity Analysis The space complexity is $O(|R|)$, same as that of generation time selection policies (Sec. IV-A), because the only change is that we replace the heap by a

TABLE IV
CHANGES AT BUFFERS (LIFO POLICY)

$r.s$	$r.d$	$r.t$	$r.q$	B_{v_0}	B_{v_1}	B_{v_2}
v_1	v_2	1	3	$\emptyset$	$\emptyset$	{(1,3)}
v_2	v_0	3	5	{(1,3),(2,2)}	$\emptyset$	$\emptyset$
v_0	v_1	4	3	{(1,2)}	{(1,1),(2,2)}	$\emptyset$
v_1	v_2	5	7	{(1,2)}	$\emptyset$	{(1,1),(2,2),(1,4)}
v_2	v_1	7	2	{(1,2)}	{(1,2)}	{(1,1),(2,2),(1,2)}
v_2	v_0	8	1	{(1,2),(1,1)}	{(1,2)}	{(1,1),(2,2),(1,1)}

FIFO queue (or a stack). This replacement changes the access and update costs from $O(\log |R|/|V|)$ to $O(1)$. Hence, the expected time cost is reduced from $O(|R|^2/|V| \log |R|/|V|)$ to $O(|R|^2/|V|)$.

C. Proportional selection

When $|B_{r.s}| > r.q$, the *proportional* selection policy, chooses the relayed quantity from $r.s$ to $r.d$ *proportionally* based on the current contribution of each vertex to $B_{r.s}$. Formally, for each vertex $v \in V$, we define a $|V|$-length vector $\mathbf{p}_v$, which captures the provenance of the quantity currently in its buffer B_v. The i-th value of $\mathbf{p}_v$ is the quantity fragment in B_v which originates from the i-th vertex of the TIN G.

Algorithm 3 shows how the provenance vectors are updated after each interaction r. We distinguish between two cases. The first one is when $r.q \geq |B_{r.s}|$, i.e., the quantity $r.q$ to be transferred by the current interaction is greater than or equal to the buffered quantity $|B_{r.s}|$ at the source buffer. In this case, the entire buffered quantity in $B_{r.s}$ is relayed to $B_{r.d}$. Hence, vector $\mathbf{p}_{r.s}$ is added to $\mathbf{p}_{r.d}$ (symbol $\oplus$ denotes vector-wise addition). If $r.q$ is strictly greater than $|B_{r.s}|$, a newborn quantity $r.q - |B_{r.s}|$ at $r.s$ is added to $B_{r.d}$, hence, we should add the corresponding provenance information to the $r.s$-th element of $\mathbf{p}_{r.d}$ (Line 6). This is denoted by the addition of vector $\mathbf{e}_{r.s,(r.q-B_{r.s})}$, where $\mathbf{e}_{v,x}$ denotes a vector with all 0's except having value x at position v. The second case is when $r.q < |B_{r.s}|$. In this case, the quantity $r.q$ which is transferred from $r.s$ to $r.d$ is chosen proportionally. Specifically, if vertex $r.s$ has in its buffer $B_{r.s}$ a quantity q which was born by the i-th vertex, then a quantity $q \cdot \frac{r.q}{|B_{r.s}|}$ should be transferred from the i-th position of $\mathbf{p}_{r.s}$ to the i-th position of $\mathbf{p}_{r.d}$. This translates into the vector-wise operations at Lines 9 and 10 of Algorithm 3. Table V shows the changes in the buffer vectors after each interaction when proportional selection is applied.

TABLE V
CHANGES AT BUFFERS (PROPORTIONAL SELECTION)

$r.s$	$r.d$	$r.t$	$r.q$	$\mathbf{p}_{v_0}$	$\mathbf{p}_{v_1}$	$\mathbf{p}_{v_2}$
v_1	v_2	1	3	$[0,0,0]$	$[0,0,0]$	$[0,3,0]$
v_2	v_0	3	5	$[0,3,2]$	$[0,0,0]$	$[0,0,0]$
v_0	v_1	4	3	$[0,1.2,0.8]$	$[0,1.8,1.2]$	$[0,0,0]$
v_1	v_2	5	7	$[0,1.2,0.8]$	$[0,0,0]$	$[0,5.8,1.2]$
v_2	v_1	7	2	$[0,1.2,0.8]$	$[0,1.66,0.34]$	$[0,4.14,0.86]$
v_2	v_0	8	1	$[0,2.03,0.97]$	$[0,1.66,0.34]$	$[0,3.31,0.69]$

Application Proportional selection is suitable for applications where the quantities are naturally mixed in the buffers. This includes cases when the buffered data are liquids or indistinguishable financial units in accounts (i.e., balances in bank accounts, capital stocks in digital portfolios). In such cases, it is reasonable to consider that the origins of the buffered quantities contribute proportionally to a transfer.

Algorithm 3 Proportional selection model

```
Require: TIN $G(V, E, R)$
 1: for each $v \in V$ do
 2:     $|B_v| = 0$; $\mathbf{p}_v = \mathbf{0}$;          ▷ Initialize buffers and vectors
 3: end for
 4: for each interaction $r \in R$ in order of time do
 5:     if $r.q \geq |B_{r.s}|$ then
 6:         $\mathbf{p}_{r.d} = \mathbf{p}_{r.d} \oplus \mathbf{p}_{r.s} \oplus \mathbf{e}_{r.s,(r.q-B_{r.s})}$; $\mathbf{p}_{r.s} = \mathbf{0}$;
 7:         $|B_{r.d}| = |B_{r.d}| + r.q$; $|B_{r.s}| = 0$;
 8:     else                                        ▷ $r.q < |B_{r.s}|$
 9:         $\mathbf{p}_{r.d} = \mathbf{p}_{r.d} \oplus (r.q/|B_{r.s}|)\mathbf{p}_{r.s}$; $B_{r.d} = B_{r.d} + r.q$;
10:         $\mathbf{p}_{r.s} = \mathbf{p}_{r.s} \ominus (r.q/|B_{r.s}|)\mathbf{p}_{r.s}$; $B_{r.s} = B_{r.s} - r.q$;
11:     end if
12: end for
```

Complexity Analysis The space requirements of this policy are $O(|V|^2)$, since we need a $|V|$-length vector for each vertex. In the next section, we investigate approaches that reduce the space requirements and make proportional provenance tracking feasible for large graphs with millions of vertices. The time complexity is also high, because we need one or two vector-wise operations per interaction, which accumulates to a $O(|R| \cdot |V|)$ cost. In our implementation, we exploit SIMD instructions [39] to reduce the cost of vector-wise operations.

Sparse vector representations In sparse graphs, each vertex v receives quantities originating from a small subset of vertices. To save space, instead of storing each space-demanding vector $\mathbf{p}_v$ explicitly, we can represent it by an ordered list of (u, q) pairs, for each vertex u contributing a quantity $q > 0$ in the buffer B_v. For example, after the temporally first interaction in our running example, instead of storing $\mathbf{p}_{v_2}$ as $[0, 3, 0]$, we store it as $[(v_1, 3)]$, implying that v_2 received its 3 units from v_1. The vector update operations of Algorithm 3 can be replaced by merging the ordered lists of the corresponding sparse vector representations. This way, the space requirements are reduced from $O(|V|^2|)$ to $O(|V| \cdot \ell)$, where ℓ is the average length of the list representations of the vectors. The time complexity is reduced to $O(|R| \cdot \ell)$, accordingly.

V. SCALABLE PROPORTIONAL PROVENANCE

Proportional provenance tracking (Section IV-C) has high space and time complexity compared to the models based on generation time (Section IV-A) or receipt order (Section IV-B). We investigate techniques constitute proportional provenance feasible on very large graphs and long streams of interactions.

A. Selective provenance tracking

In many applications, we may not have to track provenance from all vertices in the graph, but from a selected subset of V of size k. For example, in a financial network, we could limit our focus to a specific set of entities, suspected to be involved in illegal activities. To apply this, for each vertex $v \in V$, we maintain a vector $\mathbf{p}_v$ of size $k + 1$, where the first k positions correspond to the vertices of interest and the last position represents the rest of the vertices. Algorithm 3 can now directly be applied, after the following change: if any of the source vertex $r.s$ or the destination $r.d$ is not in the set of the k vertices of interest, we update the $(k+1)$-th position. The space requirements of *selective proportional provenance* are $O(k \cdot |V|)$ and its time complexity is $O(k \cdot |R|)$.

B. Grouped provenance tracking

Provenance data from all individual vertices of a big graph could be too large and hard to interpret. Sometimes, it is more practical to divide the vertices into groups and track provenance from each group. To implement this, we can replace the long $\mathbf{p}_v$ vectors by shorter vectors of length k, where k is the number of groups. This means that, for each vertex v we maintain in $\mathbf{p}_v$ the total quantity in buffer B_v which originates from each group. The grouping of vertices can be done in different ways depending on the application. For example, the values of one or more attributes that characterize the vertices (e.g., gender, country) can be used for grouping. In addition, network clustering algorithms (e.g., METIS [28]) or geographical clustering can be used to define the groups. Algorithm 3 can easily be adapted to operate on groups. The vertices involved at each interaction (i.e., $r.s$ and $r.d$) are mapped to group-ids and the corresponding positions are updated in the vector-wise operations. As in selective provenance tracking (Section V-A), the space and time complexity is $O(k \cdot |V|)$ and $O(k \cdot |R|)$, respectively.

C. Limiting the scope of provenance

If selective and grouped provenance is not an option, tracking proportional provenance in large graphs with millions of vertices could be infeasible. We investigate two techniques that limit the scope of provenance by either avoiding the tracking of quantities generated far in the past or setting a budget for provenance at each vertex. Our techniques are especially suitable when the interactions R are processed as a stream in limited memory; preciseness is traded for speed and feasibility.

1) Windowing approach: Our first approach takes as input a parameter W, representing a *window*, which determines *how far in the past* we are interested in tracking provenance. Specifically, for each vertex v we can guarantee finding the provenance of quantities that reach v, which where born up to W interactions before. To achieve this, for each v, we initialize *two* sparse (i.e., list) provenance vector representations $\mathbf{p}_v^{odd}$ and $\mathbf{p}_v^{even}$. At each interaction, both lists are updated. However, whenever we reach an interaction r whose order is a multiple of W, we *reset* either $\mathbf{p}_v^{odd}$ or $\mathbf{p}_v^{even}$ as follows. If the order of r in the sequence R of interactions is an odd multiple of W, for each vertex $v \in V$, we reset its provenance list $\mathbf{p}_v^{odd}$ by setting $\mathbf{p}_v^{odd} = [(\alpha, |B_v|)]$, where α is an *artificial vertex*, representing the entire set V of vertices. This means that we assume that the entire quantity in B_v has unknown provenance. If the order of r is an even multiple of W, for all vertices v, we reset $\mathbf{p}_v^{even}$ by setting $\mathbf{p}_v^{even} = [(\alpha, |B_v|)]$. After any interaction r, we can track provenance for any vertex v using whichever of $\mathbf{p}_v^{even}$ or $\mathbf{p}_v^{odd}$ was least recently reset. This guarantees that we can track the provenance of quantities born up to (at least) W interactions before. The space requirements (i.e., the total space required to store the provenance lists) are now controlled due to the provenance list resets.

Figure 3 illustrates how, for each vertex v, $\mathbf{p}_v^{odd}$ and $\mathbf{p}_v^{even}$ are updated and used. Assuming that $W = 100$, until the 100-th interaction, $\mathbf{p}_v^{odd}$ and $\mathbf{p}_v^{even}$ are identical and either of them can be used. Since $\mathbf{p}_v^{odd}$ is reset at the 100-th interaction, between the 100-th and the 200-th interaction $\mathbf{p}_v^{even}$ is used to track the provenance of quantities which were generated since the first interaction. Similarly, between the 200-th and the 300-th interaction $\mathbf{p}_v^{odd}$ is used to track provenance up to the 100-th interaction.

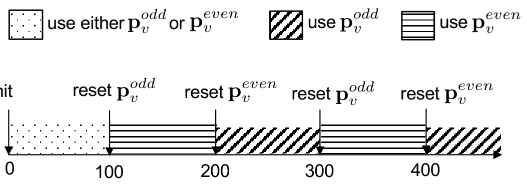

use either $\mathbf{p}_v^{odd}$ or $\mathbf{p}_v^{even}$
use $\mathbf{p}_v^{odd}$
use $\mathbf{p}_v^{even}$
init
reset $\mathbf{p}_v^{odd}$
reset $\mathbf{p}_v^{even}$
reset $\mathbf{p}_v^{odd}$
reset $\mathbf{p}_v^{even}$
0
100
200
300
400

Fig. 3. Windowing approach in provenance tracking

2) Budget-based provenance: Another way to control the memory requirements is to allocate a maximum capacity C (budget) to each vertex v for its provenance list $\mathbf{p}_v$. Whenever we have to add new entries to $\mathbf{p}_v$, if the addition requires more space than C, we select a certain fraction f of entries to keep in $\mathbf{p}_v$. We remove the remaining entries and assume that the total quantity Q which originates from them was born at an artificial vertex α, modeling all vertices (i.e., unknown source). Hence, if $\mathbf{p}_v$ includes an (α, q) entry, the entry is updated to $(\alpha, q+Q)$; if not, a new entry (α, Q) is added to $\mathbf{p}_v$.

With this approach, the space requirements of proportional provenance tracking become $O(|V| \cdot C)$. The larger the value of C the more accurate provenance tracking becomes. Parameter f should be chosen such that the memory allocated at each vertex is not underutilized and, at the same time, shrinking does not happen very often. We suggest a value between 0.6 and 0.8. Finally, the selection of entries to keep when the budget C is reached in $\mathbf{p}_v$ can be done using different criteria. For example, we can keep the entries with the largest quantities, or set a priority/importance order to vertices.

As an example, let $\mathbf{p}_v = \{(v,1),(u,3),(w,2),(z,1)\}$ and $C = 5$. Let $\{(x,2),(w,1),(y,4)\}$ be the new entries that have to be added/merged into $\mathbf{p}_v$. After the change, $\mathbf{p}_v$ should become $\mathbf{p}_v = \{(v,1),(u,3),(w,3),(x,2),(y,4),(z,1)\}$, i.e., the capacity constraint $C = 5$ is violated. If $f = 0.6$, we should keep $0.6 \cdot C = 3$ entries; let us assume that we keep the ones with the largest quantities, i.e., $\{(u,3),(w,3),(y,4)\}$. The remaining three entries are replaced in $\mathbf{p}_v$ by an entry $(\alpha, 4)$, since the sum of their quantities is 4. Hence, after the update, $\mathbf{p}_v$ becomes $\{(u,3),(w,3),(y,4),(\alpha,4)\}$. Note that selecting the entries with the largest quantities may cause a bias in favor of origins that generate quantities early over origins whose generation is spread more evenly in the timeline.

VI. Tracking the paths

So far, we have studied the problem of identifying the origins of the quantities accumulated at the vertices. An additional question is which path did each of the quantities, accumulated at a vertex v, follow from its origin to v. This information can provide more detailed *explanation* for the reasons behind data transfers [10]. To solve this problem, for each quantity element in the buffer B_v of every node v, we maintain a *transfer* path, which captures the route that the element has followed so far from its origin to v. When a new quantity element is generated (i.e., Line 20 of Algorithm 2), its path is initialized to include just the origin vertex $r.s$. Every time a quantity element is transferred from one vertex to another as a result of an interaction r' (i.e., Line 14 of Algorithm 2), its path is extended to include the transmitter vertex $r'.s$. This way, for each element, we keep track of not just its origin but also the path which the quantity has followed.

Note that path tracking in the case of proportional selection is not meaningful, because, if $r.q < |B_{r.s}|$, all quantities in $B_{r.s}$ are split to a fraction that remains at $B_{r.s}$ and a fraction that moves to $B_{r.s}$, wherein they are *combined* with the corresponding quantities from the same origins. This means that quantities in a buffer from the same origin (but potentially from multiple different paths) are mixed and indistinguishable.

Complexity Analysis Path tracking does not change the time complexity, as the number of path changes is $O(|R|)$ and each path initialization or extension costs $O(1)$. On the other hand, the space complexity increases by a factor of $O(|R|/|V|)$, i.e., the expected number of quantity element transfers (executions of Line 14 of Algorithm 2). Hence, the space complexity increases to $O(|R|^2/|V|)$.

VII. Experimental Evaluation

We experimentally evaluated the performance and scalability of our proposed provenance tracking techniques, using four real TINs, described in Section VII-A. We compare the different selection policies for information propagation in terms of throughput and memory requirements in Section VII-C. Section VII-B compares Algorithm 2 to two baseline approaches that extend previous work to track provenance. In Section VII-D, we evaluate the performance of selective and grouped provenance tracking. Section VII-E tests the approaches for limiting the scope of provenance tracking. Section VII-F evaluates the memory and computational overhead of tracking the paths of quantities accumulated at each vertex. Finally, Section VII-G presents a use case that demonstrates the practicality of provenance in TINs. All methods were implemented in C and compiled using gcc with -O3 flag. The experiments were run on a machine with a 3.6GHz Intel i9-10850k processor and 32GB RAM. The source code is publicly available at https://github.com/KosyfakiChrysanthi/ICDE2022-code

A. Description of datasets

Table VI summarizes the statistics for each of the datasets that we use in the experiments. Below, we provide a detailed description for each of them.

Bitcoin Network: This dataset, provided by the authors of [29], includes all transactions in the bitcoin network up to 2013.12.28; we considered these transactions as interactions. For each interaction, the quantity is the corresponding amount of BTCs exchanged between bitcoin addresses (vertices). We did not consider transactions with insignificant quantities (i.e., less than 0.0001 BTC). Data provenance in this network unveils the funding sources of addresses over time.

CTU Network: A Botnet traffic network was extracted and created by CTU University [17]. We designed a TIN from these data, where the vertices are the IP addresses and the interactions are byte transfers between them over time. Provenance in such a network helps identifying the origins of received data [44].

Prosper Loans: We downloaded this dataset from http://konect.cc and created the corresponding interaction network. The vertices of the network correspond to users and the interactions represent loans between them, where the quantities are the loan amounts. Tracking the provenance of amounts that reach certain nodes helps us to identify the direct or indirect relationships between lenders and borrowers.

Taxis Network: We considered NYC yellow taxi trips[2] on January 1st 2019 as interactions in a TIN, where vertices are taxi zones (pick-up and drop-off districts), the drop-off time represents the time of interactions and the number of passengers are the corresponding quantities. Provenance tracking helps investigating the origins of arriving passengers at different zones over time.

TABLE VI
Characteristics of Datasets

Dataset	#nodes	#interactions	average $r.q$
Bitcoin	12M	45.5M	34.4₿
CTU	608K	2.8M	19.2KB
Prosper Loans	100K	3.08M	$76
Taxis	255	231K	1.53

B. Comparison to baselines

In the first experiment, we compare our approach to extensions of previous work [2], [31] that could be used to track provenance in TINs. Recall that previous work computes the total flow between two TIN vertices, but does not perform provenance tracking. An intuitive extension of a flow computation algorithm would be to consider each generated quantity as a set of atomic units (e.g., cents, messages, passengers) and give each unit a tag. This allows us to track the movement of each unit throughout the network, regardless the selection policy used.[3] For example, in the TIN of Fig. 2, the smallest quantity unit is 1, hence, we tag each unit by the vertex that generates it, in order to track its route in the network. We call this approach BaselineA.

Another baseline approach is not to do any tracking, but to probabilistically estimate the provenance of each buffered

[2]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[3]Since there no information in the datasets about the selected units in data transfers, we have to assume a selection policy.

quantity, based on the generated quantity by each vertex. Specifically, we estimate the quantity at each vertex to originate from all other vertices proportionally to the generated quantities by them. For example, in the TIN of Fig. 2, after the last interaction, the buffer of B_{v_2} has 4 units. Given that v_0, v_1, v_2 generated in total 0, 7, and 2 units respectively, we estimate $\frac{7}{9} \cdot 4$ units in B_{v_2} to originate from v_1 and $\frac{2}{9} \cdot 4$ units to originate from v_2. We call this approach BaselineB.

Table VII compares our approach (Alg. 2) to the two baselines. For each tested method, we show the total runtime cost for processing all interactions and the average cosine similarity of the provenance vectors of all vertices to the corresponding true provenance vectors. BaselineA and our approach always compute the correct result, however, BaselineA is up to three orders of magnitude slower compared to our approach. This is expected, because our algorithm treats all units that have the same provenance as a *group*, saving a huge number of computations. BaselineB simply extends Algorithm 1 to keep track of the total generated quantity at each vertex, so it is the fastest method; however, it fails to estimate provenance accurately because it does not conduct any tracking. In large graphs (Bitcoin, CTU), the estimated vectors are very dissimilar compared to the correct ones.

TABLE VII
COMPARISON TO BASELINE ALGORITHMS

Dataset	Performance	BaselineA	BaselineB	Alg. 2
Bitcoin	Time (sec)	1284.94	0.22	3.10
	Similarity	1	0.01	1
CTU	Time (sec)	44.24	0.012	0.08
	Similarity	1	0.16	1
Prosper	Time (sec)	0.25	0.001	0.055
	Similarity	1	0.09	1
Taxis	Time (sec)	0.003	0.0006	0.002
	Similarity	1	0.64	1

C. Provenance tracking performance

In our first set of experiments, we measure the average throughput and the memory requirements of provenance tracking based on the different selection policies for information propagation, presented in Section IV. For each method, we processed the entire sequence of interactions and updated the provenance data at each interaction, according to the algorithms described in Section IV. Tables VIII and IX show the throughput and the peak memory use by the different selection policies. As a point of reference, we also included the basic propagation algorithm that does not track provenance (Algorithm 1), denoted by NoProv.

From the two tables, we observe that the methods based on generation time (Section IV-A) and receipt order (Section IV-B) are scaleable, since they have a very high throughput even at very large graphs with millions of interactions (i.e., Bitcoin network). Naturally, they are one to two orders of magnitude slower than NoProv, as NoProv has $O(1)$ cost per interaction. Their space overhead compared to NoProv is not high for big and sparse graphs, like Bitcoin and CTU. Methods for provenance tracking based on receipt order (LIFO and FIFO) are faster and more space-efficient compared to methods based on generation time (LRB and MRB), because they use more update-efficient data structures (queues) compared to the heaps used by LRB and MRB; also, they do not need to keep track of the birthtimes of generated quantities.

The proportional selection policy (Section IV-C) performs well only when the number of vertices in the graph is small (i.e., Taxis network). This is expected because of its $O(|V|^2)$ space requirements. Specifically, the proportional policy using dense vector representations can be used only for the Taxis network, with very good performance; in all other cases we run out of memory. Even when the sparse vector representations are used, the required memory exceeds the capacity of our machine in the Bitcoin and CTU networks after about 500K interactions. This approach can be used on the Prosper Loans network, however, it requires a lot of space (2.4GB) and it is significantly slower than the policies of Sections IV-A and IV-B, because it needs to manage and maintain long lists. This necessitates the use of the scope limiting techniques described in Section V-C, as tracking provenance from all vertices in the entire history of interactions becomes infeasible.

Fig. 4 plots the throughput of all methods over time, when applied on the three largest datasets. The plots also include the scaleable methods for proportional selection of Section V. Observe that for the methods based on generation time (LRB/MRB) and receipt order (FIFO/LIFO) the throughput is stable, with a small drop as the number of processed interactions increases, which is due to the increase of the buffered elements. As the buffers are managed by efficient data structures (heap/queue) the cost of processing interactions increases insignificantly over time. As discussed above, the proportional policy could only be applied on Prosper Loans, where we see a sharp drop in the throughput. This is attributed to the population of sparse provenance lists which become very expensive to store and process over time.

D. Selective and grouped provenance

In the next set of experiments, we evaluate the performance of proportional provenance only for a subset of vertices or for groups of vertices as described in Sections V-A and V-B. We conduct the experiments on the three largest networks (in terms of number of vertices), i.e., Bitcoin, CTU, and Prosper Loans. Recall that on these networks tracking proportional provenance from all vertices is infeasible or very expensive. Let k denote the number of selected vertices (for selective provenance) or the number of groups (for grouped provenance). We measure the runtime cost and memory requirements for different values of k. In the case of selective provenance, we select the top-k contributing vertices as the set of vertices for which we will measure provenance. That is, we first run NoProv (Algorithm 1) and measure the total quantity generated by each vertex and then choose the ones that generate the largest quantity.[4]

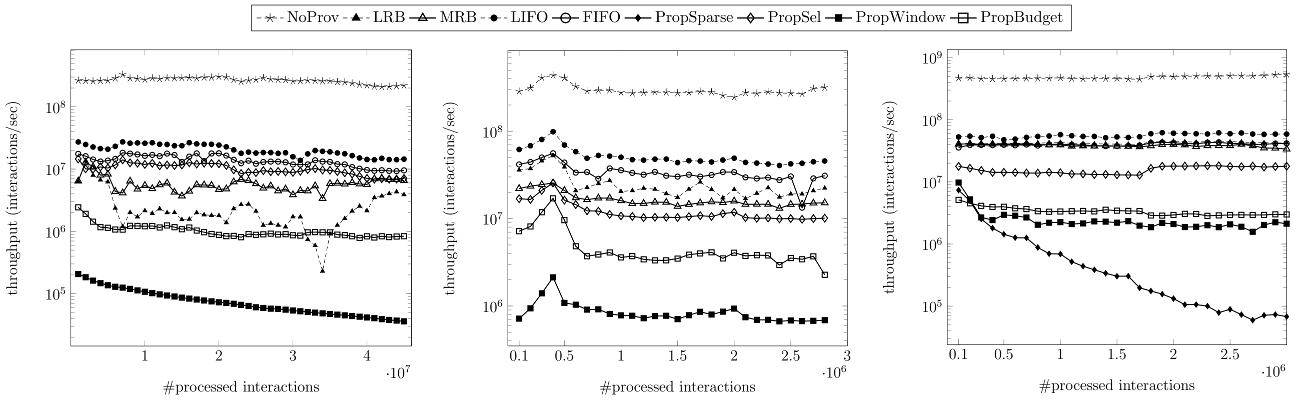
[4]The k vertices could be selected by any other method without affecting the performance of the algorithm.

TABLE VIII
THROUGHPUT (INTERACTIONS/SEC) FOR EACH SELECTION POLICY

Dataset	No Provenance	Least Recently Born	Most Recently Born	LIFO	FIFO	Proportional (dense)	Proportional (sparse)
Bitcoin	239M	1.4M	4.9M	14.6M	11.6M	–	–
CTU	280M	17.5M	14.7M	35M	25.4M	–	–
Prosper Loans	513M	34M	37M	56M	38.5M	–	196K
Taxis	462M	16.5M	15.4M	115M	57.7M	7.2M	4.6M

TABLE IX
PEAK MEMORY USED BY EACH SELECTION POLICY

Dataset	No Provenance	Least Recently Born	Most Recently Born	LIFO	FIFO	Proportional (dense)	Proportional (sparse)
Bitcoin	96MB	891MB	892MB	536MB	535MB	–	–
CTU	4.85MB	56.4MB	56.4MB	33.8MB	33.8MB	–	–
Prosper Loans	800KB	61.4MB	61.4MB	36.8MB	36.8MB	–	2.4GB
Taxis	2KB	0.93MB	1.02MB	0.59MB	0.6MB	0.52MB	0.44MB



(a) Bitcoin Network (b) CTU Network (c) Prosper Loans Network

Fig. 4. Throughput over time

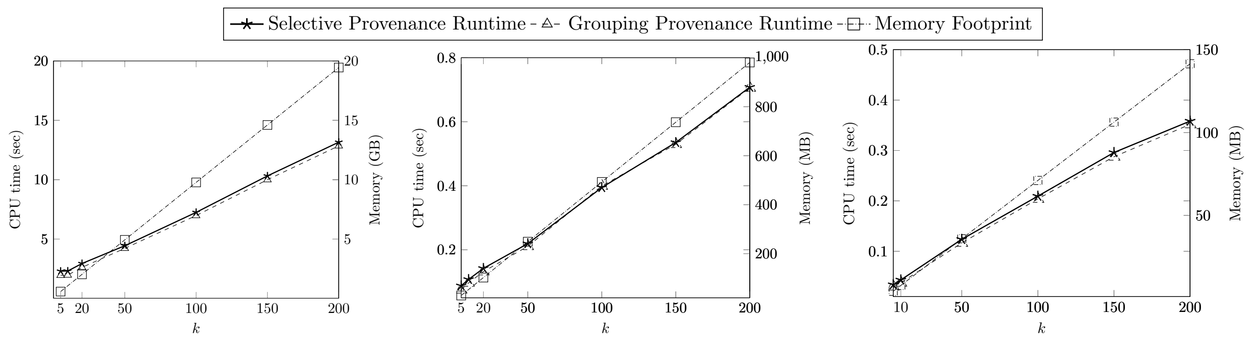
In case of grouped provenance, we randomly allocate vertices to groups in a round-robin fashion; since the runtime performance and memory requirements are not affected by the group sizes or the way the vertices are allocated to groups, this allocation does not affect the experimental results.

Figure 5 shows the runtime performance (in sec.) and memory requirements (in MB) for the different values of k on the different datasets. As expected the runtime and the memory requirements are roughly proportional to k. For small values of k (less than 20) the runtime is roughly constant with respect to k (see Figure 5(a)). This is because of the effect of SIMD instructions, which make vector operations (lines 9 and 10 of Algorithm 3) unaffected by the vector size. SIMD data parallelism is already in full action for values of k greater than 20, so we observe linear scalability from thereon. Fig. 4 shows the throughput of selective provenance (PropSel) over time, on the three TINs (for $k = 100$). Note that the throughput does not drop over time. This is expected, because the cost per interaction is linear to k and not sensitive to the number of interactions processed so far. Grouped provenance (not shown in Fig. 4) also has stable throughput for the same reasons.

E. Limiting the scope of provenance tracking

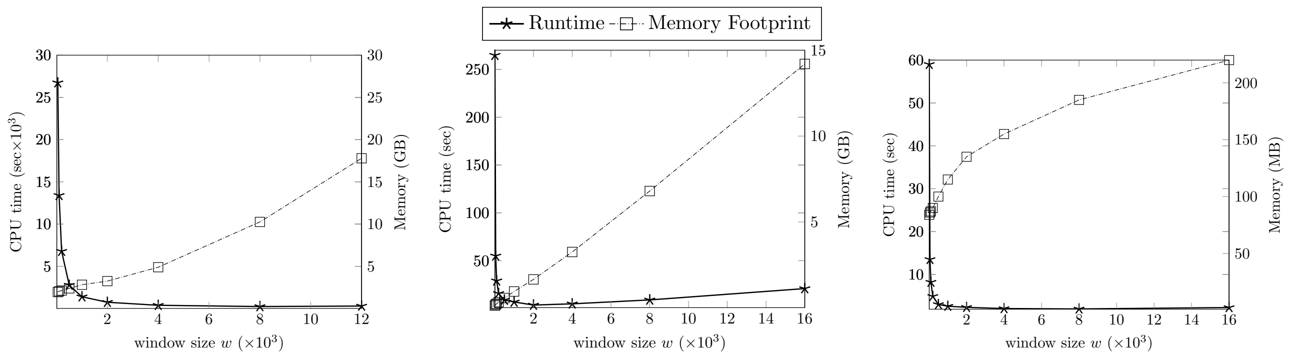
As shown in Section VII-C, proportional provenance tracking throughout the entire history of interactions is infeasible, due to its high memory requirements. In addition, keeping and updating sparse representations of provenance vectors becomes expensive over time as the lists grow larger because of the higher cost of merging operations (see Figure 4(c)).

We now evaluate the solutions proposed in Section V-C for limiting the scope of provenance tracking in order to make the maintenance of proportional provenance vectors feasible for large graphs, and real-time provenance tracking possible when the interactions R are processed as an endless stream. Once again, we experimented with the three largest networks and applied the two approaches proposed in Section V-C on them. Figure 6 shows the runtime cost and the memory requirements of the windowing approach for different values of the window parameter W. By increasing the size of the window, the runtime performance is improved as the buffers have to be reset less frequently. On the other hand, increasing the window size increases the memory requirements and increases the cost of list management. For Bitcoin and Prosper Loans, larger window sizes are affordable, as the memory requirements do not increase a lot. For CTU, the memory requirements almost double when W doubles, and the cost increases for windows larger than 2000. Fig. 4 shows the throughput of the windowing approach over time on the three datasets (for $W = 2000$). Note that there is a drop at the throughput over time and this can be attributed to the fact that more and more



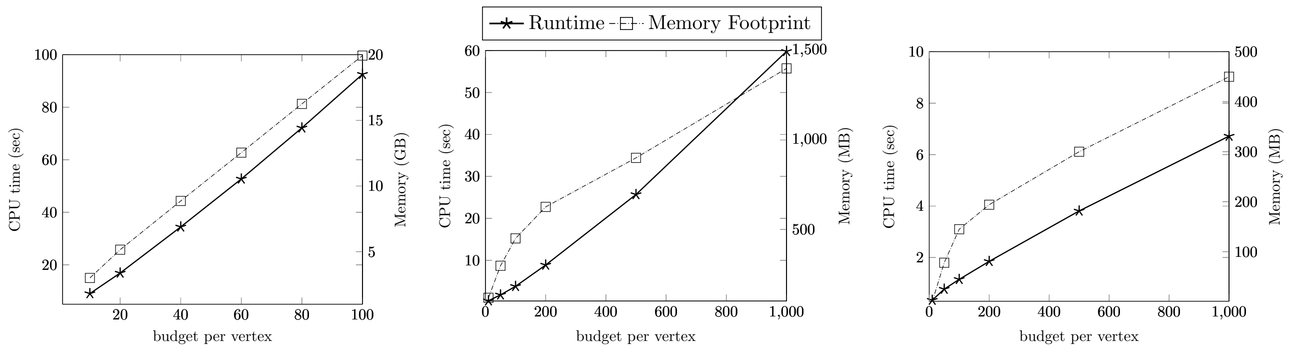
(a) Bitcoin Network (b) CTU Network (c) Prosper Loans Network

Fig. 5. Selective and grouped proportional provenance



(a) Bitcoin Network (b) CTU Network (c) Prosper Loans Network

Fig. 6. Windowing approach



(a) Bitcoin Network (b) CTU Network (c) Prosper Loans Network

Fig. 7. Budget-based provenance

vertices obtain an α element over time, which adds some overhead to list merging. Still, the throughput remains at high levels for a streaming application.

Figure 7 shows the runtime cost and the memory requirements of the budget-based approach for different values of the maximum budget C given as capacity for provenance entries to each vertex. As the figure shows, by increasing the budget C per vertex, the runtime cost to maintain provenance increases, as the provenance information at buffers becomes larger and merging lists becomes more expensive. The increase in the runtime cost is not very high though, because many lists remain relatively short and the number of list shrinks are less frequent. At the same time, the space requirements grow linearly with C, which means that very large values of C are not affordable for large graphs like Bitcoin. Fig. 4 includes the throughput of budget-based proportional provenance over time on the three datasets ($C = 100$ for Bitcoin and $C = 1000$ for CTU and Prosper Loans). As for the windowing approach, there is a small drop in the throughput over time, which is due to the fact that more and more vertices have non-empty buffers over time.

In order to assess the value of budget-based proportional provenance, in Table X, we measured for each of the three large datasets and for different values[5] of C, (i) the number

[5]We could not use values of C larger than 100 on Bitcoin due to memory constraints.

of times each non-empty buffer has been shrunk and (ii) the percentage of vertices (with non-empty buffer) whose buffer was shrunk at least once. Especially for the larger networks with high memory requirements (Bitcoin and CTU), we observe that the number of shrinks and the percentage of vertices where they take place converge to low values and, after some point, increasing C does not offer much benefit. Overall, the budget-based approach is attractive since each buffer is shrunk only a few times on average, meaning that the provenance information loss is limited even in large graphs. For example, at the Bitcoin network, for a value of $C = 50$, each buffer is shrunk 1.5 times on average after 45M interactions, meaning that each buffer tracks provenance information that traces back to millions of transactions before.

TABLE X
SHRINKING STATISTICS IN BUDGET-BASED PROVENANCE

C	Bitcoin Network		CTU Network		Prosper Loans Network	
	avg. shrinks	% vertices	avg. shrinks	% vertices	avg. shrinks	% vertices
10	1.94	18.38	7.27	31.07	20.67	94.7
50	1.51	14.79	5.1	28.68	4.77	79.29
100	1.43	14.21	4.77	27.94	2.97	69.09
200	–	–	4.53	26.6	2.1	59.16
500	–	–	4.34	25.24	1.5	47.64
1000	–	–	4.3	25.02	1.23	41.39

F. Path tracking

In the next experiment, we evaluate the overhead of tracking the paths compared to just tracking the origins of the quantities (see Section VI). We implemented path tracking as part of the LIFO selection policy for provenance (Section IV-B) and used it to track the paths for all (origin, quantity) pairs accumulated at vertices after processing all interactions in all datasets. Table XI shows the runtime performance, the memory requirements, and the average path length for each quantity element. The memory requirements are split into the memory required to store the provenance entries in the lists (as in LIFO) and the memory required to store the paths. Observe that for most datasets the memory overhead for keeping the paths is not extremely high. This overhead is determined by the average path length (last column of the table), which is relatively low in all datasets. The runtime is only up to a few times higher compared to tracking just the origins and not the paths, meaning that path tracking is feasible even for very long sequences of interactions on large graphs, like Bitcoin.

TABLE XI
TRACKING PROVENANCE PATHS IN LIFO

Dataset	time (sec.)	MB for entries	MB for paths	total MB	avg. path length
Bitcoin	13.35	534.62	847.50	1382.13	4.75
CTU	0.36	33.87	7.16	41.03	0.63
Prosper	0.4	36.85	0.74	37.59	0.06
Taxis	0.008	0.58	1.09	1.68	5.55

G. Use case

Figure 8 demonstrates a real-life application example of provenance tracing in TINs. The plot shows the total accumulated quantities at the vertices of Bitcoin after each interaction (first 100K interactions, proportional selection policy). Consider a data analyst who wants to be alerted whenever a vertex v accumulates a significant amount of money, which does not originate from v's direct neighbors (i.e., v's neighbors just relay amounts to v). Hence, after each interaction, we issue an alert when the receiving vertex does not have any quantity that originates from its neighbors and the total quantity in its buffer exceeds 10K BTC. The colored dots in the figure show these alerts (89 in total). Red dots are alerts where the number of non-neighbors that cause the alert is less than five (the rest of them are blue). We observe that in most cases the amount was received from numerous vertices (an indication of possible "smurfing"[6]). This alerting mechanism is very efficient and easy to implement, as we only have to maintain at each vertex v the total quantity that originates from vertices that transfer quantities to v (i.e., direct neighbors of v).

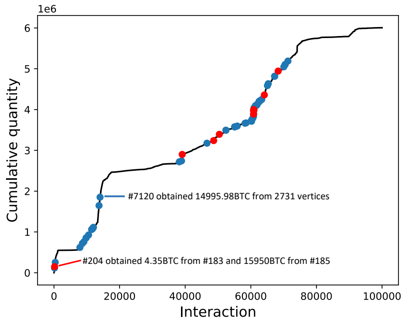


Fig. 8. Provenance alerts in Bitcoin

VIII. CONCLUSIONS

We introduced and studied provenance tracking in temporal interaction networks (TINs). We investigated different selection policies for data propagation in TINs, suitable for applications where transferred quantity units are not tagged in the network. For each policy, we proposed propagation mechanisms for provenance meta-data and analyze their space and time complexities. For the hardest policy (proportional selection), we propose to track provenance from a limited set of vertices or from groups thereof. We also propose to limit provenance tracking up to a sliding window of past interactions or to set a space budget at each vertex for provenance tracking. We evaluated our methods using four real datasets and demonstrated their scalability.

In the future, we plan to investigate *lazy* approaches [19], why-not provenance [8], [32] in TINs, and the automatic grouping of quantities to groups to improve the performance of existing work on "tagged" quantities. We will also consider spatio-temporal information in TINs (e.g., districts in passenger flow networks) to track coarse-grained provenance for spatial regions and/or identify hot paths and other interesting insights [26], [43].

ACKNOWLEDGEMENT

This research was co-financed by the European Regional Development Fund of EU and Greek national funds, under call Research-Create-Innovate (project code: T2EDK-02848).

[6]https://www.investopedia.com/terms/s/smurf.asp

References

[1] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 1151–1154, 2006.

[2] E. C. Akrida, J. Czyzowicz, L. Gasieniec, L. Kuszner, and P. G. Spirakis. Temporal flows in temporal networks. In *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, pages 43–54, 2017.

[3] G. Barbier, Z. Feng, P. Gundecha, and H. Liu. Provenance data in social media. *Synthesis Lectures on Data Mining and Knowledge Discovery*, pages 1–84, 2013.

[4] C. Belth, X. Zheng, and D. Koutra. Mining persistent activity in continually evolving networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 934–944, 2020.

[5] D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004*, pages 900–911, 2004.

[6] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001, 8th International Conference*, pages 316–330, 2001.

[7] P. Buneman, S. Khanna, and W. C. Tan. On propagation of deletions and annotations through views. In *PODS*, pages 150–158, 2002.

[8] A. Chapman and H. V. Jagadish. Why not? In U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 523–534. ACM, 2009.

[9] A. Chapman, H. V. Jagadish, and P. Ramanan. Efficient provenance storage. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 993–1006, 2008.

[10] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Found. Trends Databases*, pages 379–474, 2009.

[11] Z. Chothia, J. Liagouris, F. McSherry, and T. Roscoe. Explaining outputs in modern data analytics. *Proc. VLDB Endow.*, pages 1137–1148, 2016.

[12] R. de Paula, M. Holanda, L. S. A. Gomes, S. Lifschitz, and M. E. M. T. Walter. Provenance in bioinformatics workflows. *BMC Bioinform.*, page S6, 2013.

[13] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *13th IEEE International Conference on Peer-to-Peer Computing*, pages 1–10, 2013.

[14] D. Deutch, A. Gilad, and Y. Moskovitch. Selective provenance for datalog programs using top-k queries. *Proc. VLDB Endow.*, 8(12):1394–1405, 2015.

[15] D. Deutch, Y. Moskovitch, and N. Rinetzky. Hypothetical reasoning via provenance abstraction. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference, Amsterdam, The Netherlands*, pages 537–554. ACM, 2019.

[16] S. C. Dey, V. Cuevas-Vicenttín, S. Köhler, E. Gribkoff, M. Wang, and B. Ludäscher. On implementing provenance-aware regular path queries with relational query engines. In *Joint 2013 EDBT/ICDT Conferences, EDBT/ICDT '13, Genoa, Italy, March 22, 2013, Workshop Proceedings*, pages 214–223, 2013.

[17] S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Comput. Secur.*, pages 100–123, 2014.

[18] F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: annotating and querying databases through colors and blocks. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE, 3-8 April 2006, Atlanta, GA, USA*, page 82, 2006.

[19] B. Glavic, K. S. Esmaili, P. M. Fischer, and N. Tatbul. Ariadne: managing fine-grained provenance on data streams. In *DEBS*, pages 39–50. ACM, 2013.

[20] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Provenance in ORCHESTRA. *IEEE Data Eng. Bull.*, pages 9–16, 2010.

[21] P. Gundecha, Z. Feng, and H. Liu. Seeking provenance of information using social media. In *22nd ACM International Conference on Information and Knowledge Management, CIKM*, pages 1691–1696, 2013.

[22] T. Heinis and G. Alonso. Efficient lineage tracking for scientific workflows. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1007–1018, 2008.

[23] M. Herschel, R. Diestelkämper, and H. Ben Lahmar. A survey on provenance: What for? what form? what from? *VLDB J.*, pages 881–906, 2017.

[24] P. Holme and J. Saramäki. Temporal networks. *Physics reports*, pages 97–125, 2012.

[25] M. Interlandi, K. Shah, S. D. Tetali, M. A. Gulzar, S. Yoo, M. Kim, T. D. Millstein, and T. Condie. Titian: Data provenance support in spark. *Proc. VLDB Endow.*, pages 216–227, 2015.

[26] V. Kaffes, G. Giannopoulos, N. Tsakonas, and S. Skiadopoulos. Determining the provenance of land parcel polygons via machine learning. In *SSDBM 2020: 32nd International Conference on Scientific and Statistical Database Management, Vienna, Austria, July 7-9, 2020*, pages 21:1–21:4. ACM, 2020.

[27] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD, Indianapolis, Indiana, USA, June 6-10*, pages 951–962, 2010.

[28] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[29] D. Kondor, M. Pósfai, I. Csabai, and G. Vattay. Do the rich get richer? an empirical analysis of the bitcoin transaction network. *CoRR*, abs/1308.3892, 2013.

[30] C. Kosyfaki, N. Mamoulis, E. Pitoura, and P. Tsaparas. Flow motifs in interaction networks. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT, Lisbon, Portugal, March 26-29*, pages 241–252, 2019.

[31] C. Kosyfaki, N. Mamoulis, E. Pitoura, and P. Tsaparas. Flow computation in temporal interaction networks. In *37th IEEE International Conference on Data Engineering, ICDE, Chania, Greece, April 19-22*, pages 660–671, 2021.

[32] S. Lee, B. Ludäscher, and B. Glavic. PUG: a framework and practical implementation for why and why-not provenance. *VLDB J.*, 28(1):47–71, 2019.

[33] S. Lee, B. Ludäscher, and B. Glavic. Approximate summaries for why and why-not provenance. *Proc. VLDB Endow.*, pages 912–924, 2020.

[34] A. Li, S. P. Cornelius, Y.-Y. Liu, L. Wang, and A.-L. Barabási. The fundamental advantages of temporal networks. *Science*, 358(6366):1042–1046, 2017.

[35] N. Masuda and P. Holme. Predicting and controlling infectious disease epidemics using temporal networks. *F1000Prime Reports*, 5(6), 2013.

[36] M. H. Namaki, A. Floratou, F. Psallidas, S. Krishnan, A. Agrawal, Y. Wu, Y. Zhu, and M. Weimer. Vamsa: Automated provenance tracking in data science scripts. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27*, pages 1542–1551, 2020.

[37] A. Paranjape, A. R. Benson, and J. Leskovec. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM, Cambridge, United Kingdom, February 6-10*, pages 601–610. ACM, 2017.

[38] N. N. Parulian, T. M. McPhillips, and B. Ludäscher. A model and system for querying provenance from data cleaning workflows. In *Provenance and Annotation of Data and Processes - 8th and 9th International Provenance and Annotation Workshop, IPAW 2020 + IPAW 2021, Virtual Event, July 19-22, Proceedings*, pages 183–197, 2021.

[39] O. Polychroniou, A. Raghavan, and K. A. Ross. Rethinking SIMD vectorization for in-memory databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1493–1508, 2015.

[40] F. Psallidas and E. Wu. Demonstration of smoke: A deep breath of data-intensive lineage applications. In *Proceedings of the International Conference on Management of Data, SIGMOD Conference, Houston, TX, USA, June 10-15*, pages 1781–1784, 2018.

[41] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang. Fine-grained, secure and efficient data provenance on blockchain systems. *Proceedings of the VLDB Endowment*, pages 975–988, 2019.

[42] L. Rupprecht, J. C. Davis, C. Arnold, Y. Gur, and D. Bhagwat. Improving reproducibility of data science pipelines through transparent provenance capture. *Proc. VLDB Endow.*, pages 3354–3368, 2020.

[43] D. Sacharidis, K. Patroumpas, M. Terrovitis, V. Kantere, M. Potamias, K. Mouratidis, and T. K. Sellis. On-line discovery of hot motion paths. In *EDBT 2008, 11th International Conference on Extending Database Technology, Nantes, France, March 25-29, 2008, Proceedings*, volume 261 of *ACM International Conference Proceeding Series*, pages 392–403. ACM, 2008.

[44] S. Savage, D. Wetherall, A. R. Karlin, and T. E. Anderson. Practical network support for IP traceback. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 28 - September 1, Stockholm, Sweden*, pages 295–306, 2000.

[45] M. Skutella. An introduction to network flows over time. In *Bonn Workshop of Combinatorial Optimization*, 2008.

[46] I. Taxidou, T. D. Nies, R. Verborgh, P. M. Fischer, E. Mannens, and R. V. de Walle. Modeling information diffusion in social media as provenance with W3C PROV. In *Proceedings of the 24th International Conference on World Wide Web Companion, WWW*, pages 819–824, 2015.

[47] D. Zhou, L. Zheng, J. Han, and J. He. A data-driven graph generative model for temporal interaction networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 401–411, 2020.

[48] A. Züfle, M. Renz, T. Emrich, and M. Franzke. Pattern search in temporal social networks. In *Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*, pages 289–300, 2018.