HAL open science

Computing Temporal Twins in Time Logarithmic in History Length

Binh-Minh Bui-Xuan, Hugo Hourcade, Cédric Miachon

To cite this version:

Binh-Minh Bui-Xuan, Hugo Hourcade, Cédric Miachon. Computing Temporal Twins in Time Logarithmic in History Length. Complex Networks 2020, Dec 2020, Madrid, Spain. hal-02997890

HAL Id: hal-02997890

https://hal.science/hal-02997890v1

Submitted on 10 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Computing Temporal Twins in Time Logarithmic in History Length *

Binh-Minh Bui-Xuan[1], Hugo Hourcade[1,2], and Cédric Miachon[2]

[1] LIP6 (CNRS – Sorbonne Université), [buixuan,hugo.hourcade]@lip6.fr
[2] Courtanet, [hugo.hourcade,cedric.miachon]@lesfurets.com

Abstract. A temporal graph $\mathcal{G}$ is a sequence of static graphs indexed by a set of integers T representing time instants. For Δ an integer, a pair of Δ-twins is a pair of vertices $u \neq v$ which, starting at some time instant, have exactly the same neighbourhood outside $\{u, v\}$ for Δ consecutive instants. We address the enumeration problem of all pairs of Δ-twins in $\mathcal{G}$, such that the overall runtime depends the least on the history length, namely $\max\{t : G_t \in \mathcal{G} \text{ not empty }\} - \min\{t : G_t \in \mathcal{G} \text{ not empty }\}$. We give logarithmic solutions, using red-black tree data structure. Numerical analysis of our implementation on graphs collected from real world data scales up to 10^8 history length[3].

Keywords: graph theory, historical data, modular decomposition

1 Introduction

Graph data from historical databases are well captured in the formalism of a link stream [13], a time varying [3], temporal [7] or evolving graph [2]. These notions occur in as various use cases as transportation timetables [5,8,11], navigation programs [4,14], email exchanges [12], proximity interactions [16], and many other types of dataset [17]. Therein, duplicated data rather correspond to twin vertices, in the sense of [15]: a pair of true twins are vertices $u \neq v$ sharing the same neighbourhood; else, they are still false twins if their neighbourhoods outside $\{u, v\}$ are the same. In a static graph, removing redundant twin vertices helps in reducing both space and time complexity of graph problems via modular decomposition, see *e.g.* [6,9,15] for a broad survey.

The notion of twin vertices in a temporal graph is ambivalent on the time dimension: shall twins be eternally or just temporary so, then for how long? We formally define two versions of such a notion in the subsequent section. Informally, in the EternalTwins version, we address the problem of enumerating all pairs of vertices which are twins ubiquitously in a historical dataset of graphs; we refer to Δ-Twins, with Δ an integer, as the problem of enumerating all pairs of vertices which are twins in at least Δ consecutive time instants, at some moment in the dataset.

* Supported by Courtanet – Sorbonne Université convention C19.0665 and ANRT grant 2019.0485.

[3] Source code at https://github.com/DaemonFire/deltatwinsMEI.

Data duplicates are well modeled by EternalTwins. Two vertices in this case would have exactly the same behaviour at any time in the link stream. Δ-Twins on the other hand could represent a temporary likeliness between two agents who are not duplicates per se. Indeed, outside the corresponding Δ time window, they are prone to have different behaviours. Δ-Twins could therefore help in detecting behavioural patterns on a specific time period which would characterize populations of a same group.

Previous works. In a static graph, standard approaches for finding twin vertices include modular decomposition [15], partition refinement [10], and the following bucket sort for finding true twins. The idea is to stream the neighbourhood `N(v)` of every vertex `v` into the corresponding entry `T[N(v)]` of an array `T` in linear time in the vertex number n using the following function:

$$\texttt{v} \mapsto \texttt{T[N(v)]==null? T[N(v)]=v: print(v,T[N(v)])},$$

where `N(v)` is encoded as an n-bit array of 1 and 0 representing the line of the adjacency matrix corresponding to vertex `v`. This does not enumerate (`print`) all possible pairs of true twins as (a, b) and (a, c) can thus be found but then not (b, c). Besides, buckets come with a costly space complexity as the machine representation of `N(v)` can be a very big integer, forcing table `T` to have a lot of entries. However, this latter inconvenience can be circumvented by replacing `T[N(v)]` by `T[hash(N(v))]`, accepting some very low probability of false positives. Consider now a temporal graph as a sequence $\mathcal{G} = (G_t)_{t\in T}$ of static graphs. Likewise, the true twins case of EternalTwins can be solved by replacing `T[hash(N(v))]` with $\texttt{T[hash(N}_1\texttt{(v)}\times\texttt{N}_2\texttt{(v)}\times\ldots\texttt{)]}$, where $\texttt{N}_\texttt{t}\texttt{(v)}$ is the neighbourhood of vertex `v` in graph G_t. The main crux here is that the pair of vertices must be twins at any time instant. Hence, we "only" need to check the property everywhere, *e.g.* in the Cartesian product of all neighbourhoods. If we consider that applying the `hash` function to an array already present in RAM memory (reading $\texttt{N}_\texttt{t}\texttt{(v)}$'s from adjacency matrices) is in constant time, then the previous bucket streaming is very fast. The problem is much less simple with Δ-Twins, where we do not know when the Δ time window starts. Note that problems with a sliding Δ time window have recently been widely studied, *e.g.* with vertex cover [1] and cliques [18], and the extensive references therein. However, up to our knowledge, twin vertices have not been considered before.

Basically, for the specific case of true twins of EternalTwins, the above described bucket sort could be considered a solution linear in $O(n + \tau)$, where n is the number of vertices and τ is the history length $\tau = \max\{t : G_t \in \mathcal{G} \text{ not empty}\} - \min\{t : G_t \in \mathcal{G} \text{ not empty}\}$. More generally, EternalTwins can be solved by calling modular decomposition algorithms [9,15] on every graph G_t for $t \in T$, for a global time complexity in $O((n + m) \times \tau)$, where $m = m_1 + m_2 + \ldots$ is the total number of all recorded edges in $(G_t)_{t\in T}$. As for Δ-Twins, partition refinement techniques such as in [10] can be deployed consecutively Δ times at every instant graph G_t for a global polynomial time complexity with a factor in $\tau \times \Delta$. In settings where the history length τ is important, a new algorithm with time logarithmic in τ is desirable. Note that we can have $\tau >> m$ if there are many instants t where G_t is an empty graph. If we only consider

T as the set of time instants with at least one recorded edge, then computing Δ-twins which are defined on Δ consecutive instants, would require additional arithmetic operations to take into account those deleted instants.

Our paper addresses the following question: *Would there be instantaneous response to* Δ-TWINS *on historical graph data collected from human activities? Can it be numerically confirmed by implementation of those algorithms?* In particular, the foci in this paper are: long history (big τ), few vertices (small n) and good number of recorded edges (medium m). We revisit red-black tree data structure and devise a computation with runtime logarithmic in the history length of the input, and confront its implementation to one generated dataset and three datasets collected from real world data.

Theoretical contribution. We revisit matrix-based implementation of partition refinement and use red-black tree data structure in order to devise two variants of an algorithm for Δ-TWINS. The two variants differ in the use of a large matrix in memory, the matrix-less version being the key to avoid *out of RAM* problems. Furthermore, the use of red-black trees allows our algorithm to compute even in the case where input graphs G_t for $t \in T$ are given unordered, mixing parts of one graph to another. This feature is fault tolerant for batched data which come asynchronously. All in all, the computation time is $O(m \times n \log \tau + N)$ with a $O(n^2 \times \tau)$ size adjacency matrix in memory and $O(m^2 \times n \log \tau + N)$ without it, where N represents the size of the output.

Numerical experiments. We confront our implementations to generated data in order to confirm that the implementation is sound and its runtime is logarithmic in history length τ. We then confront it to real world datasets, with two collections from previous experiments [12,16] and a new one called `LesFurets`. The runtime of our algorithm averages at 12 seconds for all but one dataset, where it averages at 70 seconds.

The formal framework of Δ-TWINS is defined in next Section 2. In Section 3, we use red-black tree data structure in order to achieve logarithmic runtime in the history length of the input data. All numerical analysis of our implementation are presented in Section 4, before we close the paper with concluding remarks.

2 Twin vertices in a historical recording of graphs

Graphs in this paper are simple, undirected, and unweighted. A *temporal graph* is a sequence of graphs indexed by integers representing time instants. For practical use, it can also be formalized as a *link stream* $L = (T, V, E)$ such that $T \subseteq \mathbb{N}$ is an interval, V is a finite set, and $E \subseteq T \times \binom{V}{2}$. The elements of V are called *vertices* and the elements of E are called *(recorded) edges*. For $t \in T$, the subgraph G_t of L induced by t is a graph over the same vertex set V, with edge set $E_t = \{\{u, v\} : (t, \{u, v\}) \in E\}$. In this paper, we indifferently refer to temporal graphs as link streams. The *adjacency matrix sequence* $(M_t)_{t \in T}$ is the sequence of adjacency matrices of graphs $(G_t)_{t \in T}$.

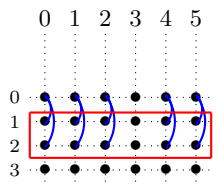

Fig. 1. In this link stream, vertices 1 and 2 (represented by rows with the corresponding identifiers) have exactly the same links to other vertices at every instants from 0 to $5 = \tau - 1$. They form a pair of eternal-twins.

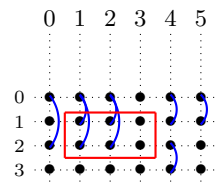

Fig. 2. In this link stream, vertices 1 and 2 (represented by rows with the corresponding identifiers) have exactly the same links to other vertices for instants 1, 2 and 3. They form a pair of Δ-twins for any $\Delta \leq 3$.

Temporal twin vertices have two variants. A pair of *eternal twins* $\{u, v\} \in \binom{V}{2}$ is a pair of vertices for which the neighbourhoods $N_t(u)$ and $N_t(v)$ are strictly equal in $V \setminus \{u, v\}$ for every instant $t \in T$. For an integer Δ, a pair of *Δ-twins* $\{u, v\} \in \binom{V}{2}$ is a pair of vertices for which the neighbourhoods $N_t(u)$ and $N_t(v)$ are strictly equal in $V \setminus \{u, v\}$ for Δ consecutive instants $t_0 \leq t < t_0 + \Delta$ with $[\![t_0, t_0 + \Delta[\![\subseteq T$. Our paper addresses the following problems, which both have polynomial time solutions.

EternalTwins
Input : A link stream L.
Output : A list of all pairs of eternal twins in L.

Δ-Twins
Input : A link stream L and an integer Δ.
Output : A list of all pairs of Δ-twins in L.

The matrix based technique for *partition refinement* is defined as follows. Assume initially that all pair of vertices are eternal twins: $\texttt{Tw}(u, v) = \texttt{true}$ for all $u \neq v$. For every pair of vertices $u \neq v$, time instant $t \in T$, and vertex $w \in V \setminus \{u, v\}$, if $M_t(w, u) \neq M_t(w, v)$, then $\texttt{Tw}(u, v) = \texttt{false}$. Such a vertex w is called a *splitter of* $\{u, v\}$. At the end of the process, output every entry $u \neq v$ of table $\texttt{Tw}$ where $\texttt{Tw}(u, v) = \texttt{true}$. This results in a naive $O(n^3 \times \tau)$ solution for EternalTwins. A similar process repeated Δ times using Δ different tables $\texttt{Tw}$ allows to solve Δ-Twins in time $O(n^3 \times \tau \times \Delta)$.

3 Temporal twins in time logarithmic in history length

In many cases, an input link stream $L = (T, V, E)$ is not given by its adjacency matrix sequence $(M_t)_{t \in T}$, but rather by a list of its recorded edges E. Then, computing EternalTwins in time independent from history length τ can be done by using the triangular structure of splitters, as in Algorithm 1.

The overall complexity is $O(m \times n + n^2)$ if $(M_t)_{t \in T}$ is also given as input, for a constant time $(t, \{u, w\}) \notin E$ testing. This is the matrix version or Matrix Edge

```
Data: Linkstream L : (T, V, E)
Result: List of all eternal-twins of L
for vertex u do
    for vertex v ≠ u do
        Initialize Tw(u, v) = true;
    end
end
for recorded edge (t, {u, v}) ∈ E do
    for vertex w ∈ V \ {u, v} do
        if (t, {u, w}) ∉ E then
            Tw(v, w) = false;                    // u is a splitter of {v, w}
        end
    end
end
return every entry u ≠ v of table Tw where Tw(u, v) = true.
```

Algorithm 1: Edge Iteration algorithm for eternal-twins listing

Iteration algorithm (MEI). When $(M_t)_{t \in T}$ is part of the input, $O(n^2 \times \tau)$ space complexity at runtime is required, which usually causes *out of RAM* problems for big τ. It is in $O(m^2 \times n + n^2)$ otherwise, since scanning $(t, \{u, w\}) \notin E$ could take $O(m)$ especially if E is given unordered by $t \in T$. This is the matrix-less version or Matrix-less Edge Iteration algorithm (MLEI). We note that in practice the latter $O(m)$ factor is small, especially when E is chronologically ordered, by dichotomy search. We have proved the following property.

Property 1. EternalTwins *can be solved in time independent from history length.*

In order to address Δ-Twins, we will use a tree-based data structure inspired from red-black trees. Each node represents a time range $P \subseteq T$ and contains a time range $D \subseteq P$ of consecutive instants having been removed from T. The 2 sons of this node represents time ranges $Q \subseteq P$ and $R \subseteq P$ with Q the time range preceding D and R the time range following D. Removing an instant t from T will result in trying to remove it from the node at the root of this tree which represents T. If this node contains no time range of time deleted, the range of time deleted becomes $[t, t]$ and we create both sons of this node, which represent $\{a \in T, a < t\}$ and $\{a \in T, a > t\}$. If the node contains a time range of deleted instants $D = \{a \in T, t_0 \leq a \leq t_1\}$, if $t = t_0 - 1$ or $t = t_1 + 1$ we add t to D. If not, we try to remove t in the adequate son of this node.

After an update of a node, we check if time ranges erased contained in nodes are colliding. If $D(left_son) = [t_0, t_1]$ and $D(father) = [t_1 + 1, t_2]$, we fuse the son in the father, $D(father) \leftarrow [t_0, t_2]$ and $left_son(father) \leftarrow left_son(left_son(father))$ This data structure allows us to store a discontinued time range. We can then extract from it all time spans of at least Δ consecutive instants by exploring the tree and looking up time ranges represented by leaves of this tree.

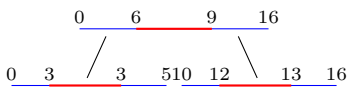

Fig. 3. Representation of $[0, 16]$ where $3, 6, 7, 8, 9, 12, 14$ are removed. The two leaves respectively represent time ranges preceding and following $[6, 9]$. All remaining intervals can be enumerated from the intervals in blue contained in the leaves.

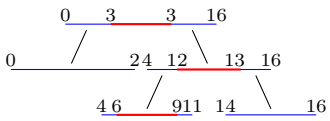

Fig. 4. This tree represents the same time partitioning as the one in Fig. 3 but is one level deeper and therefore requires more operations and a greater computation time to delete a new instant.

Complexity of the deletion of an instant and computation of the time ranges of Δ consecutive instants in such a structure is at worst case in $O(d)$ with d the depth of the tree, this depth being lesser or equal to the number of non-colliding time range deleted. Some optimizations can be accomplished using this data structure. For instance, using it to solve Δ-twins listing problem, when an instant t is removed, if either $t_f - t < \Delta$ with t_f the last instant in the node in which t is being removed or $t - t_i < \Delta$ with t_i the first instant in the node, we can consider that the interval $[t, t_f]$ (respectively $[t_i, t]$) is removed, thus diminishing computation time by avoiding useless computations.

This type of trees can be balanced in order to minimize computation time of deletion. But this balancing comes at a price. It requires computation of the depth of each sub-tree and to recursively balance each node. If we proceed to this balancing operation at each deletion of an instant, we only need to balance the node on which we deleted the instant and all the parent nodes, root included. We therefore proceed to $O(d)$ operations, where d is the depth of the tree. The balancing operation in itself is in constant time, providing the depth of a sub-tree is saved in its root node and updated accordingly.

Balancing operation of a node: if $depth(left_son) - depth(right_son) > 2$, then we can rotate left:

$t_f(father) \leftarrow t_i(right_son) - 1;$
$t_i(right_son) \leftarrow t_i(father);$
$(right_son(father)) \leftarrow left_son(right_son);$
$left_son(right_son) \leftarrow father;$
$father \leftarrow right_son;$

A rotation of a node loses no information, which means that sub-trees rooted on each of the sons of one node will have a depth at most one level distant. We can therefore ensure a depth in $log(p)$ where p is the number of non-colliding time ranges deleted from the tree. That means that the depth of those trees would be inferior at all time to $log(\tau)$ where τ is the history length of the input link stream. Therefore, deletion and balancing operations' complexity would be at worst case in $O(log(\tau))$.

Δ-twins listing algorithm based on edges iteration (MEI and MLEI). We now adapt the algorithm solving the eternal-twins problem to our Δ-twins listing problem, as in Algorithm 2.

```
Data: A linkstream L : (T, V, E) and an integer δ
Result: A list of all Δ-twins of L
We initialize for each pair of vertices Tw(u, v) = Tree(T);
Initialize a list R of all entries in Tw;
for recorded edge (t, {u, v}) of E do
    for vertex w do
        if (t, {u, w}) ∉ E then
            Remove instant t in Tw(v, w);
            if a removal exhausts the time instants in Tw(v, w) then
                remove entry (v, w) from R;
            end
        end
    end
end
for (u, v) left in R do
    scan all time ranges of at least Δ consecutive instants in Tw(u, v) and add
      all ranges to output
end
return output
```

Algorithm 2: Edge Iteration Algorithm for Δ-twins listing

The overall complexity of this Matrix Edge Iteration algorithm (MEI) is $O(m \times n \log \tau + N)$ with n the number of vertices, m the number of recorded edges, τ the history length and N the number of pairs of Δ-twins if $(M_t)_{t \in T}$ was given. The space complexity is $O(n^2 \times \tau)$, due to the use of $(M_t)_{t \in T}$. If $(M_t)_{t \in T}$ is not given, scanning E to ascertain that $(t, \{u, w\}) \notin E$ adds to the complexity. If E is chronologically ordered, the scanning can be fast, by dichotomy search. However, for a worst case complexity this step requires $O(m)$ time. Complexity for this version of the algorithm is therefore $O(m^2 \times n \log \tau + N)$. This is the Matrix-less Edge Iteration algorithm (MLEI). The overall space complexity of algorithm MLEI is $O(n^2 \log \tau)$, due to the use of `Tw`. This space complexity remains reasonable as our focus is for graphs with small n. Due to space restriction, we omit the proof of the following theorem.

Theorem 1. *On input a link stream with n vertices, m recorded edges, τ history length, and N pairs of Δ-twins, Δ-*TWINS *can be solved in time $O(m \times n \log \tau + N)$ with $O(n^2 \times \tau)$ space complexity, or in time $O(m^2 \times n \log \tau + N)$ with $O(n^2 \log \tau)$ space complexity.*

4 Numerical Analysis

All the algorithms listed in the previous sections have been implemented in Java[4] and run on a standard laptop clocking at 2.7 Ghz. Since the use of ETERNALTWINS is practically somewhat limited, plus the fact the algorithms is independent from history length, we only present numerical results for Δ-TWINS.

[4] Source code at https://github.com/DaemonFire/deltatwinsMEI.

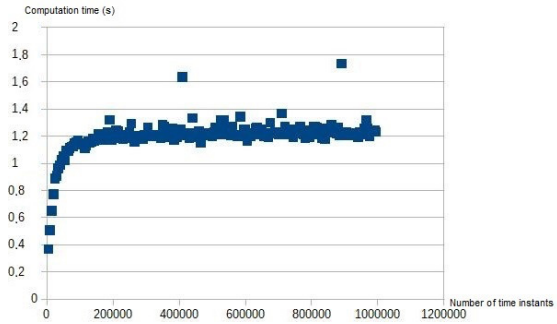

Fig. 5. Computation time of MLEI algorithms solving the Δ-TWINS listing problem in function of history length on the `Timeprogression` datasets. We do not have consistent results for MEI due to *out of RAM*.

Basically, for the correctness of the various implementations, our methodology is unit-testing. The control groups are obtained from running an implementation of the naive computations in $O(n^3 \times \tau \times \Delta)$ described in Section 2. Due to the high time complexity of the naive computations, we do unit-testing uniquely for instances where the naive computations do not exceed 45 minutes. This covers $\approx 33\%$ instances of all our experiments in both below Section 4.1 and Section 4.2. Results are positive on all these samples. In what follow we will totally skip the discussion about correctness, and only focus on computation time. We first stress-test the implementation on big values of history length with a generated dataset, in Section 4.1. Then, we confront our implementation to three different datasets collected from real world data, in Section 4.2. Our overall experiments run for more than 3000 (three thousand) hours CPU time.

4.1 Logarithmic dependency in history length on runtime

Theoretically, our algorithms compute twin vertices in time logarithmic in the history length of the input temporal graph. We would like to confirm this on runtime of their implementation.

Hypothesis 1. The runtime computation is logarithmic in the history length of the input temporal graph.

Dataset and experiment result. Our methodology is to generate an artificial dataset called `Timeprogression` in order to monitor history length's influence on computation time while maintaining constant numbers of vertices and edges. Number of vertices was set to 50 and number of edges to 10^5, ascertaining that both dimensions are small enough so that history length's influence on algorithm's computation time would not be prone to be negligible. There are 199 instances, with history length varying from 5000 to 10^6 time instants. This dataset is not ordered by time instants. Results are presented in Figure 5.

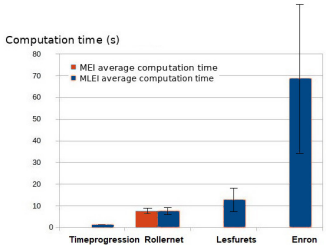

Fig. 6. Overview of computation time of all experiments.

Discussion: confirmation of Hypothesis 1. Progression of computing time is logarithmic, with few jumps (2 cases) probably due to some noisy use of the PC during computation.

4.2 Runtime on real world datasets

We confront our implementations on real world datasets, and would like to experiment both hypothesis below.

Hypothesis 2. Δ-twins can be enumerated in reasonable time.

Hypothesis 3. Algorithm MLEI is able to compute link streams that cause exhaustion of memory for the MEI version.

Datasets and experiment result. Our methodology is to confront the implementations on three different datasets collected from real world data. We focus on Δ-Twins with $\Delta = 102$. In the sequel, we describe our sampling method over the three datasets. Then, a global view of all computation time is captured in Figure 6. We develop with detailed views on each of the three datasets, in Figures 7, 8, and 9, respectively. We leave all discussions for the corresponding paragraph at the end of this section.

`Rollernet` dataset [16] has been collected from rollerbladers touring Paris. Links will be recorded at instant t whenever two rollerbladers are close enough during a given period. This is a dense linkstream. This dataset is more likely to present a relatively greater number of Δ-twins than the two other datasets. We run our experiments on the following seven batches of extracts, each batches containing 100 extracts. Two first batches contain link streams induced by $n_1 = 40$, resp. $n_2 = 50$, vertices of the raw Rollernet dataset. Three batches contain link streams induced by $m_1 = 10^5$, $m_2 = 2 \cdot 10^5$, and $m_3 = 3 \cdot 10^5$, recorded edges of the raw dataset. Two last batches contain link streams induced by $\tau_1 = 5000$, resp. $\tau_2 = 8000$, successive time instants of the raw dataset.

`Enron` dataset [12] is parsed from the log of e-mail exchanges between employees of a same company over a period of 3 years. Δ-twins would emerge from this link stream as people from the same service are sent the same e-mails for a certain period of time. This link stream is very sparse. We do the following

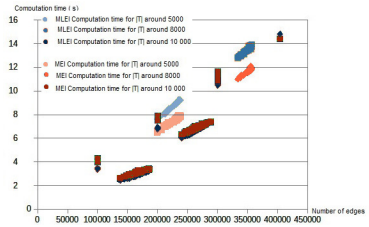

Fig. 7. Computation time of MLEI and MEI algorithms solving the Δ-twins listing problem in function of the number of edges in the link stream on the Rollernet datasets.

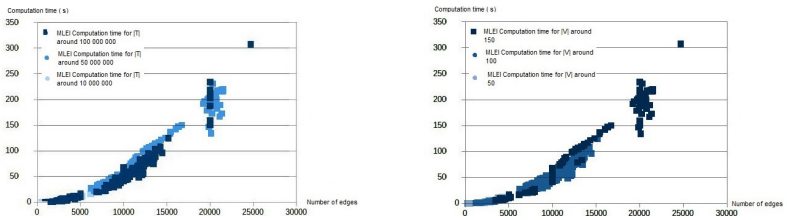

Fig. 8. Computation time of MLEI algorithms solving the Δ-twins listing problem in function of the number of edges in the link stream on the Enron datasets.

seven batches of 100 extracts each, similarly as for `Rollernet`, with $n_1 = 50$, $n_2 = 100$, $m_1 = 5000$, $m_2 = 10000$, $m_3 = 20000$, $\tau_1 = 10^7$, and $\tau_2 = 5 \cdot 10^7$.

`LesFurets` dataset is parsed from the log of user behaviour on the lesfurets's funnel, some vertices representing the various users and the others representing events on the funnel. This link stream is therefore a bipartite link stream. This dataset is not ordered by time instants. The latter feature also provides a way to test the robustness of our approach, in the sense of fault tolerance. We do the following seven batches of 100 extracts each, similarly as the other datasets, with $n_1 = 300$, $n_2 = 600$, $m_1 = 3000$, $m_2 = 6000$, $m_3 = 8000$, $\tau_1 = 10000$, and $\tau_2 = 13000$.

Discussion: slight confirmation of Hypothesis 2; confirmation of Hypothesis 3. Our experiments on the Rollernet datasets are where naive algorithm computes in reasonable computation time, allowing us to ascertain that MEI and MLEI algorithms compute Δ-twins correctly, *cf.* Figure 7. It is also the only dataset we treated on which MEI doesn't encounter *out of RAM* issues, where we observe that MEI tends to be a bit more quicker to compute than MLEI. According to the heat-map, $|T|$ seems to have a minor impact on computation time as many datasets with large $|T|$ compute faster than datasets with small $|T|$. The overall tendency towards greater computation time being due to the increase of the number of vertices and of edges more than history length.

As soon as we reach Enron and LesFurets datasets, $|V|$ gets too big and naive algorithm computation time grows unreasonably. $|T|$ also grows to a large number and MEI, as expected, starts to face memory issues. Hypothesis 2 seems

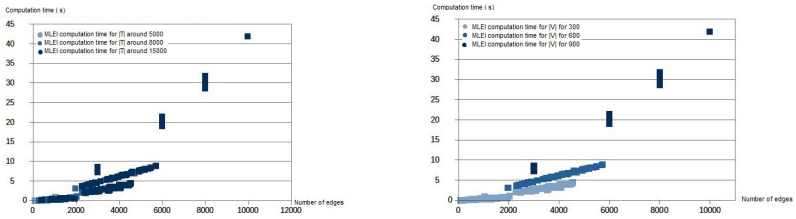

Fig. 9. Computation time of MLEI algorithms solving the Δ-twins listing problem in function of the number of edges in the link stream on the Lesfurets datasets.

to be strained on Enron dataset. But we can still use those datasets to experiment on MLEI algorithm. For Enron, the left hand side of Figure 8 allows us to confirm our theoretical complexity regarding $|E|$ as our experimental curve of computation time in function of number of edges seems to describe a second degree polynomial function. The heat-map once again allows us to picture that history lenght's influence on complexity seems not to be so clear, indicating that number of edges has a greater impact on computation time than history length of the link stream. On the other hand, the right hand side of Figure 8, where the heat-map correspond to number of vertices shows us what can be expected of a heat-map about an important complexity factor as darker points correspond to greater computation time than lighter ones. We proceed similarly for the results on LesFurets datasets, *cf.* Figure 9. They confirm the same tendencies as with Enron.

We conclude from our experiments that Hypothesis 2 is strained on Enron dataset, whereas Hypothesis 3 is confirmed. All in all, twin vertices can be computed within some minutes, even on Enron dataset.

5 Conclusion and perspectives

We introduced two variants of twin vertices in a historical collection of graphs. The corresponding algorithmic problems of enumerating all such twin vertices are polynomial. We address the problem of solving them in time depending the least in the history length. Revisiting partition refinement techniques along with red-black tree data structures, we devise a logarithmic solution. Our solution is subject to two sub-variants: with or without the use of adjacency matrices in runtime memory. Confronting to datasets collected from real world data, our solutions scales up to 10^8 history length. An interesting development of this work could be the replacement of all matrix data by hash-maps or sorted arrays. Then, extensive numerical analysis should be made in order to compare these three approaches (matrix, hash-maps and sorted arrays).

Acknowledgements: We are grateful to Emmanuel Chailloux for helpful discussion and pointers. We are grateful to the anonymous reviewers for their helpful comments which greatly improved the paper.

References

1. Akrida, E., Mertzios, G., Spirakis, P., Zamaraev, V.: Temporal vertex cover with a sliding time window. Journal of Computer and System Sciences 107, 108–123 (2020)
2. Bui-Xuan, B.M., Ferreira, A., Jarry, A.: Computing shortest, fastest, and foremost journeys in dynamic networks. International Journal of Foundations of Computer Science 14(2), 267–285 (2003)
3. Casteigts, A., Flocchini, P., Godard, E., Santoro, N., Yamashita, M.: Expressivity of time-varying graphs. In: 19th International Symposium on Fundamentals of Computation Theory. pp. 95–106 (2013)
4. Dean, B.: Continuous-Time Dynamic Shortest Path Algorithms. Ph.D. thesis, Massachusetts Institute of Technology (1999)
5. Dibbelt, J., Pajor, T., Strasser, B., Wagner, D.: Connection scan algorithm. ACM Journal of Experimental Algorithmics 23 (2018)
6. Ehrenfeucht, A., Harju, T., Rozenberg, G.: The Theory of 2-Structures: A Framework for Decomposition and Transformation of Graphs. World Scientific (1999)
7. Erlebach, T., Hoffmann, M., Kammer, F.: On Temporal Graph Exploration. In: 42nd International Colloquium on Automata, Languages, and Programming. LNCS, vol. 9134, pp. 444–455 (2015)
8. Foschini, L., Hershberger, J., Suri, S.: On the complexity of time-dependent shortest paths. Algorithmica 68(4), 1075–1097 (2014)
9. Habib, M., Paul, C.: A survey of the algorithmic aspects of modular decomposition. Computer Science Review 4(1), 41–59 (2010)
10. Habib, M., Paul, C., Viennot, L.: Partition refinement techniques: an interesting algorithmic tool kit. International Journal of Foundations of Computer Science 10(2), 147–170 (1999)
11. Kempe, D., Kleinberg, J., Kumar, A.: Connectivity and inference problems for temporal networks. Journal of Computer and System Sciences 64(4), 820–842 (2002)
12. Klimt, B., Yang, Y.: Introducing the Enron Corpus. In: CEAS (2004)
13. Latapy, M., Viard, T., Magnien, C.: Stream graphs and link streams for the modeling of interactions over time. Social Network Analysis and Mining 8(61) (2018)
14. Ros, F., Ruiz, P., Stojmenovic, I.: Acknowledgment-based broadcast protocol for reliable and efficient data dissemination in vehicular ad-hoc networks. IEEE Transactions on Mobile Computing 11(1), 33–46 (2012)
15. Spinrad, J.: Efficient Graph Representations. Field Institute Monographs, vol. 19. American Mathematical Society (2003)
16. Tournoux, P.U., Leguay, J., Benbadis, F., Conan, V., De Amorim, M.D., Whitbeck, J.: The Accordion Phenomenon: Analysis, Characterization, and Impact on DTN routing. In: 28th IEEE Conference on Computer Communications (2009)
17. Tsalouchidou, I., Baeza-Yates, R., Bonchi, F., Liao, K., Sellis, T.: Temporal betweenness centrality in dynamic graphs. International Journal of Data Science and Analytics pp. 1–16 (2019)
18. Viard, T., Magnien, C., Latapy, M.: Enumerating maximal cliques in link streams with durations. Information Processing Letters 133, 44–48 (2018)