On Computing the Diameter of (Weighted) Link Streams

Marco Calamai
University of Florence, Italy

Pierluigi Crescenzi
Gran Sasso Science Institute, L'Aquila, Italy

Andrea Marino
University of Florence, Italy

Abstract

A weighted link stream is a pair $(V, \mathbb{E})$ comprising V, the set of nodes, and $\mathbb{E}$, the list of temporal edges (u, v, t, λ), where u, v are two nodes in V, t is the starting time of the temporal edge, and λ is its travel time. By making use of this model, different notions of diameter can be defined, which refer to the following distances: earliest arrival time, latest departure time, fastest time, and shortest time. After proving that any of these diameters cannot be computed in time sub-quadratic with respect to the number of temporal edges, we propose different algorithms (inspired by the approach used for computing the diameter of graphs) which allow us to compute, in practice very efficiently, the diameter of quite large real-world weighted link stream for several definitions of the diameter. Indeed, all the proposed algorithms require very often a very low number of single source (or target) best path computations. We verify the effectiveness of our approach by means of an extensive set of experiments on real-world link streams. We also experimentally prove that the temporal version of the well-known 2-sweep technique, for computing a lower bound on the diameter of a graph, is quite effective in the case of weighted link stream, by returning very often tight bounds.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Temporal graph, shortest path, diameter

Digital Object Identifier 10.4230/LIPIcs.SEA.2021.11

Supplementary Material *Software (Source Code)*: `https://github.com/marcocalamai/Link-stream-diameter` archived at `swh:1:dir:b33c6ae74c0739836c1874ca3fc7ac30f3893be9`

Funding Partially supported by MIUR under PRIN Project n. 20174LF3T8 AHeAD (Efficient Algorithms for HArnessing Networked Data), and by the University of Florence under Project GRANTED (GRaph Algorithms for Networked TEmporal Data).

Acknowledgements We thank Filippo Brunelli and Laurent Viennot for several discussions concerning the complexity of computing single source (target) best paths in temporal graphs. We also thank Roberto Grossi for kindly providing us the computing platform.

1 Introduction

Link stream, distance, eccentricity, and diameter. A time-dependent network is a graph $G = (V, E)$ in which each edge $e \in E$ has associated an arrival function specifying, for each starting time, the corresponding arrival time (see [4] for a classification of different types of time-dependent networks). A weighted *link stream* [14] (also called temporal graph in [22, 23] and point-availability time-dependent network in [4]) is a time-dependent network in which the domain of the arrival functions is a finite set T of time instants (in this paper, we will assume that T is a set of integer numbers). A weighted link stream is commonly represented as a list $\mathbb{E}$ of *temporal edges* (u, v, t, λ), where u and v are two nodes in V, $t \in T$ is the *starting* time of the edge, and λ denotes the *travel* time of the edge: according to this

representation, the arrival time of the edge (u, v) at time t is equal to $t + \lambda$. In this paper, we refer to this representation by assuming that the list is sorted either in non-decreasing or in non-increasing order with respect to the edge starting times: in the former case we will denote the list as $\overrightarrow{\mathbb{E}}$, while in the latter case we will denote the list as $\overleftarrow{\mathbb{E}}$. We will also assume that the travel time of any temporal edge is an integer *positive* value. Moreover, we will assume that if the link stream is *undirected*, then a temporal edge $(u, v, t, \lambda) \in \mathbb{E}$ implicitly implies that the temporal edge (v, u, t, λ) is also in the link stream. Finally, we will say that the link stream is *unweighted* if the travel time of all temporal edges is equal to 1.

When dealing with weighted link streams, the definition of path asks to satisfy, besides the typical constraints of a path in a graph, some natural time constraints. In particular, a *path* from a node u to a node v is a sequence of temporal edges

$$(u = w_1, w_2, t_1, \lambda_1), (w_2, w_3, t_2, \lambda_2), \ldots, (w_{k-1}, w_k, t_{k-1}, \lambda_{k-1}), (w_k, w_{k+1} = v, t_k, \lambda_k)$$

such that, for each i with $1 < i \leq k$, $t_i \geq t_{i-1} + \lambda_{i-1}$. The *departure time* of the path is t_1, its *arrival time* is $t_k + \lambda_k$, its *duration* is $t_k + \lambda_k - t_1$, and its *travel time* is $\sum_{i=1}^{k} \lambda_i$. In the following, for any time interval $[t_\alpha, t_\omega]$, we will say that the path is $[t_\alpha, t_\omega]$-compatible, if its departure time is no earlier than t_α and its arrival time is no later than t_ω.

By making use of the above four path cost functions, we can then define the following four corresponding distances between two nodes u and v, in a specific time interval $[t_\alpha, t_\omega]$ (for all distances, we assume that their value is ∞, if there is no $[t_\alpha, t_\omega]$-compatible path) [24, 22, 23].

Earliest arrival time $d_{\text{EAT}}^{[t_\alpha, t_\omega]}(u, v)$ is the minimum arrival time of any $[t_\alpha, t_\omega]$-compatible path from u to v minus t_α.

Latest departure time $d_{\text{LDT}}^{[t_\alpha, t_\omega]}(u, v)$ is t_ω minus the maximum departure time of any $[t_\alpha, t_\omega]$-compatible path from u to v.

Fastest time $d_{\text{FT}}^{[t_\alpha, t_\omega]}(u, v)$ is the minimum duration time of any $[t_\alpha, t_\omega]$-compatible path from u to v.

Shortest time $d_{\text{ST}}^{[t_\alpha, t_\omega]}(u, v)$ is the minimum travel time of any $[t_\alpha, t_\omega]$-compatible path from u to v.

In the following, for the sake of simplicity, we will almost always avoid to specify the time interval in the superscript, and assume that it is the interval in which t_α is the minimum departure time of all temporal edges, and t_ω is the maximum arrival time.

Once a notion of distance is adopted, the corresponding notions of eccentricity and of diameter can be introduced, analogously to the case of standard graphs. That is, for any $\text{D} \in \{\text{EAT}, \text{LDT}, \text{FT}, \text{ST}\}$, the (forward) *eccentricity* $\text{ECCF}_\text{D}(u)$ of a node u is its maximum finite distance to any other node, and the *diameter* $\varnothing_\text{D}$ of a weighted link stream is the maximum (defined) eccentricity of all its nodes.[1] *The goal of this paper is to analyse the problem of computing the diameter of a weighted link stream, in the case of the four previously defined distances.* To this aim, let us first note that several algorithms have been proposed in the literature in order to compute the distance of a source node s to all other nodes, for each of the distance definitions considered in this paper. For any $\text{D} \in \{\text{EAT}, \text{LDT}, \text{FT}, \text{ST}\}$, let $\texttt{ssbp}_\text{D}$ be an algorithm that, given in input $\overrightarrow{\mathbb{E}}$ and a source node u, returns an array containing, for each node v, the value $d_\text{D}(u, v)$, and let S-TIME_D (respectively, S-SPACE_D) be the worst-case time (respectively, space) complexity of this algorithm, as a function of the number n of nodes and of the number m of temporal edges in the weighted link stream. In this paper, we

[1] In this paper, we use the symbol $\varnothing$ since it is obtained by the command `diameter` in LaTeX, and since graphically it reminds the notion of a diameter.

Table 1 The time and space complexity of the single source best path and the single target best path algorithms with input $\overrightarrow{\mathbb{E}}$ and $\overleftarrow{\mathbb{E}}$, respectively (without considering the time and space necessary for sorting the link stream).

D	S-TIME$_{\text{D}}$	S-SPACE$_{\text{D}}$	Ref.	T-TIME$_{\text{D}}$	T-SPACE$_{\text{D}}$	Ref.
EAT	$O(m)$	$O(n)$	[22, 23]	$O(m)$	$O(m)$	[9, 10]
LDT	$O(m)$	$O(m)$	[9, 10]	$O(m)$	$O(n)$	[22, 23]
FT	$O(m)$	$O(m)$	[9, 22, 23]	$O(m)$	$O(m)$	[9, 22, 23] & Lemma 1
ST	$O(m \log m)$	$O(m)$	[22, 23]	$O(m \log m)$	$O(m)$	[22, 23] & Lemma 1

mostly refer to the algorithms proposed in [9, 10, 22, 23]: as far as we know, they have the best time and space complexity (see the second and third column of Table 1). Nevertheless, our results can be easily adapted to other algorithms proposed in the literature, that usually have the same time and space complexity. In the following, we will assume that both $\overrightarrow{\mathbb{E}}$ and $\overleftarrow{\mathbb{E}}$ are available: otherwise, all time complexities of Table 1 become $O(m \log m)$, and all space complexities become $O(m)$.

Computing the diameter. For any $\text{D} \in \{\text{EAT}, \text{LDT}, \text{FT}, \text{ST}\}$, in order to compute the corresponding diameter of a weighted link stream, we can execute the algorithm $\texttt{ssbp}_{\text{D}}$, for each source node u (we will refer to such "text-book" approach as the algorithm TB_{D}). However, the time complexity of this approach is $O(n \cdot \text{S-TIME}_{\text{D}}(n, m))$: by looking at Table 1, we have that this time complexity is not affordable whenever we have thousands or millions of nodes, and millions or billions of temporal edges in the link stream. Unfortunately, *our first result consists in showing that it is very unlikely that there exists an algorithm computing any of the four diameters in time sub-quadratic in the number of temporal edges*. In other words, it is reasonable to conjecture that the known algorithms for computing any of the four diameters are, indeed, optimal.

In order to deal with this complexity conjecture, in this paper we propose to follow the approach that has been adopted while computing the diameter in real-world large graphs [7, 18, 8, 6, 19, 3] (and other distance-based measures like hyperbolicity [1]). This approach consists in *sorting the nodes* of the graph in a "clever" way, in computing, for each node in the given order, the eccentricity of the node, and in *updating a lower bound* on the value of the diameter and *an upper bound* on the value of the eccentricities of the remaining nodes, until the upper bound becomes less than or equal to the lower bound. The first main difficulty of this approach is, hence, to determine which order should be used in order to stop the process as soon as possible. For example, in the case of unweighted undirected connected graphs, the iFUB algorithm [6] first performs a breadth-first search starting from a random node x, and subsequently visits the nodes in the breadth-first search tree in a bottom-up fashion (this approach can be generalised to strongly connected directed graphs by executing both a forward and a backward breadth-first search starting from x, and by then combining paths entering x with paths exiting x [8]). The second main difficulty of this approach is finding a lower and an upper bound, which are easy to compute and to update, and that also allows the algorithm to stop as soon as possible. In the case of the iFUB algorithm, for instance, the lower bound is simply the maximum eccentricity computed so far, while the upper bound is a simple value connected to the level of the breadth-first search tree at which the algorithm is arrived.

In this paper, we show that this approach can also be applied to the computation of the diameter of a weighted link stream, for any of the four previously defined distances. To this aim, as in the case of directed graphs [8], we will make use of a "backward" variation

of the single source best path algorithms. For any $\mathrm{D} \in \{\text{EAT}, \text{LDT}, \text{FT}, \text{ST}\}$, let $\texttt{stbp}_{\mathrm{D}}$ be an algorithm that, given in input $\overleftarrow{\mathbb{E}}$ and a target node v, returns an array containing, for each node u, the value $d_{\mathrm{D}}(u, v)$, and let $\text{T-TIME}_{\mathrm{D}}$ (respectively, $\text{T-SPACE}_{\mathrm{D}}$) be the worst-case time (respectively, space) complexity of this algorithm, as a function of the number n of nodes and of the number m of temporal edges in the weighted link stream. Once again, in this paper we mostly refer to the algorithms proposed in [9, 10, 22, 23]: as far as we know, they have the best time and space complexity (see the fifth and sixth column of Table 1). In the following, we will denote with $\text{ECCB}_{\mathrm{D}}(u)$ the *backward eccentricity* of a node u, that is its maximum finite distance from any other node: note that the diameter $\varnothing_{\mathrm{D}}$ of a weighted link stream can also be defined as the maximum (defined) backward eccentricity of all its nodes.

1.1 Our results

Computing $\varnothing_{\mathsf{eat}}$ and $\varnothing_{\mathsf{ldt}}$. In the case of the earliest arrival time (respectively, latest departure time) distance, we propose to sort the nodes according to the maximum arrival (respectively, minimum departure) time of the edges entering (respectively, exiting) a node, in a non-increasing (respectively, non-decreasing) order. This choice is mostly inspired by the fact that, in the case of collaboration or citation link streams (such as the IMDB [11] and the DBLP-Citation [16, 20] networks), the earliest arrival time (respectively, latest departure time) diameter of the link stream coincides with the collaboration or citation of one of the last (respectively, first) nodes which entered the time-dependent network. Surprisingly, we will show that this intuition leads us, in practice, to a very efficient algorithm for computing the earliest arrival time (respectively, latest departure time) diameter of weighted link streams of different types (such as, public transport networks). For each analysed node, the algorithm executes a single target (respectively, source) best path computation, and update, in an appropriate way, both the lower and the upper bound: in particular, the lower bound is always the maximum distance seen so far, while the upper bound becomes the maximum arrival (respectively, minimum departure) time of the edges entering (respectively, exiting) the next node. As a result, our algorithm is able to compute the EAT and the LDT diameter of large public transport networks using much lower visits with respect to the text book algorithm. This performance improvement is particularly significant for the three biggest public transport networks in our dataset, as the number of performed visits becomes smaller than 0.5% of the number of nodes.

Computing $\varnothing_{\mathsf{ft}}$ and $\varnothing_{\mathsf{st}}$. In the case of the fastest and the shortest time distance, the situation is more complicated. Indeed, aiming at adopting the approach used for weighted directed graphs by the iFUB-like algorithm, we need to guarantee that the paths leading to a node x, whose forward and backward distances to and from all other nodes have been computed, are temporally compatible with the paths exiting from x and, that they can, hence, be "temporally" combined. Unfortunately, this is not always the case. This situation is similar to the one that arises when the iFUB approach is applied to weakly connected graphs: indeed, in this case either the diameter of the largest strongly connected component only is computed, or the component graph is computed in order to choose a pivot node for each strongly connected component, to bound the eccentricities of pivot nodes, and to propagate these bounds within each strongly connected component. The definition of (strongly) connected component in the case of link streams is somehow more involved (see [14]) and, as far as we know, no efficient algorithm capable of computing the analogue of the component graph has been proposed so far. For this reason, in this paper we have chosen to adopt the first solution, that is, restricting ourselves to the computation of what

we have called the *pivot-diameter*. In our case, a pivot is a node at a given time instant, that is, a pair (u, t), where $u \in V$ and $t \in T$. Given a set P of pivots, the pivot-diameter of a weighted link stream with respect to P is the diameter restricted to the set $R(P)$ of pair of nodes in V such that, for any pair $(u, v) \in R(P)$, there is a path from u to v which passes through one the pivots in P (our notion of pivot-diameter is connected to the notion of "pivotability" introduced in [5]). Note that, in some real-world weighted link streams, it should be possible to find a small pivot set such that almost all pairs of nodes are included in $R(P)$. For example, in the case of public transport networks, such a pivot set could be formed by the central station taken at different times of the day. In any case, once the set of pivots are given, we propose an algorithm that is able to compute the pivot-diameter of large link streams, improving very often the text-book algorithm of at least one order of magnitude. The worst case running time can be quadratic, but this seems to be unavoidable, as witnessed by the conditional lower bound we prove for this problem. It is also worth noting that, in the case of public transport networks, the pivot-diameter is, in many cases, very close to the diameter of the link stream, when the pivots are the top out-degree nodes, taken at few time instants.

Computing lower bounds on the diameter. Thanks to the backward and the forward visits discussed in Section 1.2 and whose complexities are summarized in Table 1, we are able to extend a well-known method for computing lower bounds for the diameter of static graphs to the case of temporal graphs. This method is called double-sweep and can be applied both to directed and undirected graphs: it selects a random node r (sometimes chosen as a high degree node [8]), it performs a forward visit from r obtaining the node a_1 which is the farthest from r, and set $lb_1 = \max_{b \in V} d(b, a_1)$. Then it performs a backward visit from r obtaining the node a_2 which is the farthest from r, and set $lb_2 = \max_{b \in V} d(a_2, b)$. Finally, it returns the maximum between lb_1 and lb_2. This method naturally extends to the case of weighted link streams, for all the distances EAT, LDT, FT, ST, by using for each distance D the corresponding forward and backward best path search, i.e. using $\texttt{ssbp}_{\text{D}}$ and $\texttt{stbp}_{\text{D}}$. We analyse the performance of this revised temporal double sweep in our experiments for all these distances, and we show that the computed lower bound is very often tight, when performing $O(\log_2 n)$ double sweeps. In the case of ST, the lower bounds are often tight when dealing with public transport networks, while they seem to be rarely tight in the case of social networks. In any case, also in this latter case, double sweep significantly outperforms random sampling approaches.

1.2 Two useful link stream transformations

In [9], the authors introduce a simple transformation from a weighted linked stream $(V, \mathbb{E})$ to an unweighted link stream $(V \cup I, \mathbb{F})$, where I is a set of at most $|\mathbb{E}|$ "intermediate" nodes. Intuitively, this transformation changes the travel time of a temporal edge into a waiting time in the intermediate nodes. More precisely, for each temporal edge $e = (u, v, t, 1) \in \mathbb{E}$, e is also included in $\mathbb{F}$. For each temporal edge $e = (u, v, t, \lambda) \in \mathbb{E}$ with $\lambda > 1$, a new node i_e is inserted in I, and the two temporal edges $(u, i_e, t, 1)$ and $(i_e, v, t + \lambda - 1, 1)$ are included in $\mathbb{F}$. It is easy to verify that, for any two nodes $u, v \in V$, $d_{\text{D}}(u, v)$ in $(V, \mathbb{E})$ is equal to $d_{\text{D}}(u, v)$ in $(V \cup I, \mathbb{F})$, where $\text{D} \in \{\text{EAT}, \text{LDT}, \text{FT}\}$. This implies that the diameter of $(V, \mathbb{E})$ is equal to the maximum (defined) eccentricity of the nodes in V computed in $(V \cup I, \mathbb{F})$.

The following lemma (whose proof is given in Appendix), instead, introduces another transformation which will allow us to easily design a backward version of a best path search and to relate the EAT distance and the LDT distance.

▶ **Lemma 1.** *Given a weighted link stream $(V, \mathbb{E})$, let $(V, \mathbb{F})$ be the weighted link stream obtained by substituting each temporal edge $e = (u, v, t, \lambda) \in \mathbb{E}$ with the temporal edge $\rho(e) = (v, u, -t - \lambda, \lambda)$. Then, for any two nodes $u, v \in V$, $d_{\text{D}}^{[t_\alpha, t_\omega]}(u, v)$ in $(V, \mathbb{E})$ is equal to $d_{\text{D}}^{[-t_\omega, -t_\alpha]}(v, u)$ in $(V, \mathbb{F})$, where $\text{D} \in \{\text{FT}, \text{ST}\}$. Moreover, $d_{\text{EAT}}^{[t_\alpha, t_\omega]}(u, v)$ in $(V, \mathbb{E})$ is equal to $d_{\text{LDT}}^{[-t_\omega, -t_\alpha]}(v, u)$ in $(V, \mathbb{F})$, and $d_{\text{LDT}}^{[t_\alpha, t_\omega]}(u, v)$ in $(V, \mathbb{E})$ is equal to $d_{\text{EAT}}^{[-t_\omega, -t_\alpha]}(v, u)$ in $(V, \mathbb{F})$.*

In the rest of the paper, we are going to use the above two transformations for the following three purposes. First, we can ignore the LDT distance, since computing $\varnothing_{\text{LDT}}$ is equivalent to computing $\varnothing_{\text{EAT}}$. Second, we can obtain an algorithm $\texttt{stbp}_{\text{FT}}$ by applying the transformation ρ of the lemma, by then applying the transformation in [9] to reduce to unitary weights, and by finally applying the algorithm in [22]. Third, we can obtain an algorithm $\texttt{stbp}_{\text{ST}}$ by applying the transformation ρ of the lemma and by then using the algorithm $\texttt{ssbp}_{\text{ST}}$ described in [22].

2 Negative results

The *Strong Exponential Time Hypothesis* (in short, *SETH*) states that there is no algorithm for solving the k-Sat problem in time $O((2-\epsilon)^n)$, where $\epsilon > 0$ does not depend on k [12]. This hypothesis has been repeatedly used in the last few years in order to prove the hardness of polynomial-time solvable problem (see, for example, [21], which is one of the first papers along this line of research, where the authors address the hardness of many problems, like computing all pairs shortest paths and finding triangles in a graph). We will here use it in order to prove that the diameter of an unweighted undirected link stream cannot be computed in time sub-quadratic with respect to the number of temporal edges.

To this aim, we will refer to the *k-Big Two Disjoint Set* (in short, *k-BTDS*) problem, which is defined as follows. Given a set X and a collection $\mathcal{C}$ of subsets of X such that $|X| \leq \log^k(|\mathcal{C}|)$, the solution is 1 if there are two disjoint sets $c, c' \in \mathcal{C}$, 0 otherwise. Clearly, this problem can be solved in quadratic time. It is also known that, for any k, the k-BTDS problem is not solvable in time $\tilde{O}(|\mathcal{C}|^{2-\epsilon})$, unless the SETH is false [2] (where the $\tilde{O}$ notation ignores poly-logarithmic factors).

▶ **Theorem 2.** *For any $\text{D} \in \{\text{EAT}, \text{FT}, \text{ST}\}$, computing the diameter $\varnothing_{\text{D}}$ of a linked stream $(V, \mathbb{E})$ cannot be done in time $\tilde{O}(|\mathbb{E}|^{2-\epsilon})$ for any $\epsilon > 0$, unless the SETH is false, even if the link stream is unweighted and undirected.*

Proof. We show that the k-BTDS problem is reducible (in quasi-linear time) to the link stream diameter computation problem, even in the case in which the diameter is equal either to 2 or to 3. Given an input $(X = \{x_1, \ldots, x_{|X|}\}, \mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\})$ of k-BTDS with $|X| \leq \log^k(|\mathcal{C}|)$, we construct an unweighted undirected link stream $(X \cup \mathcal{C}, \mathbb{E})$, where the set $\mathbb{E}$ of temporal edges is defined as follows (see also Figure 3 in Appendix).

- For each $x_i, x_j \in X$, $\mathbb{E}$ contains the two temporal edges $(x_i, x_j, 1, 1)$ and $(x_i, x_j, 2, 1)$.
- For each c_j and for each $x_i \in c_j$, $\mathbb{E}$ contains the three temporal edges $(x_i, c_j, 1, 1)$, $(x_i, c_j, 2, 1)$, and $(x_i, c_j, 3, 1)$.

For any $\text{D} \in \{\text{EAT}, \text{FT}, \text{ST}\}$, let us now compute the eccentricities of all nodes, by distinguishing between nodes in X and nodes in $\mathcal{C}$.

Nodes in X For each $x_i, x_j \in X$, $d_{\text{D}}(x_i, x_j) = 1$ (since we can use the temporal edge with starting time equal to 1). For each $x_i \in X$ and $c_j \in \mathcal{C}$, $d_{\text{D}}(x_i, c_j) = 1$ if $x_i \in c_j$ (since we can use the temporal edge with starting time equal to 1). Otherwise, $d_{\text{D}}(x_i, c_j) = 2$ (since we can first use the temporal edge from x_i to x_k with starting time equal to 1, for some $x_k \in c_j$, and then use the temporal edge from x_k to c_j with starting time equal to 2). Hence, for each $x_i \in X$, we have that $\text{ECCF}_{\text{D}}(x_i) = 2$.

Nodes in $\mathcal{C}$ For each $c_i \in \mathcal{C}$ and $x_j \in X$, we have already shown that $d_{\text{D}}(c_i, x_j) \leq 2$. For each other $c_j \in \mathcal{C}$, $d_{\text{D}}(c_i, c_j) = 2$ if $c_i \cap c_j \neq \emptyset$ (since we can first use the temporal edge from c_i to x_k with starting time equal to 1, for some $x_k \in c_i \cap c_j$, and then use the temporal edge from x_k to c_j with starting time equal to 2). Otherwise (that is, if $c_i \cap c_j = \emptyset$), we have that $d_{\text{D}}(c_i, c_j) \leq 3$ (since we can first use the temporal edge from c_i to x_k with starting time equal to 1, for some $x_k \in c_i$, then use the temporal edge from x_k to x_l with starting time equal to 2, for some $x_l \in c_j$, and finally use the temporal edge from x_l to c_j with starting time equal to 3). Note that, in this case, $d_{\text{D}}(c_i, c_j) = 3$ since there is no way of arriving in c_j starting from c_i before time 3, since no neighbor of c_i is also a neighbor of c_j, and, hence, we are forced to pass through two nodes in X. Hence, for each $c_i \in \mathcal{C}$, $\text{ECCF}_{\text{D}}(c_i) = 3$ if there exists $c_j \in \mathcal{C}$ such that $c_i \cap c_j = \emptyset$, otherwise $\text{ECCF}_{\text{D}}(c_i) = 2$.

We can the conclude that the diameter $\varnothing_{\text{D}}$ of the link stream is either 2 or 3: it is 3 if and only if there exist two $c_i, c_j \in \mathcal{C}$ which are disjoint.

Since $|X| \leq \log^k(|\mathcal{C}|)$, the reduction can be executed in $\tilde{O}(|\mathcal{C}|)$ time, and $|\mathbb{E}| = \tilde{O}(|\mathcal{C}|)$. Hence, if we can compute the diameter of the link stream in $\tilde{O}(|\mathbb{E}|^{2-\epsilon})$ for some $\epsilon > 0$, then we could solve the k-BTDS in $\tilde{O}(|\mathcal{C}|^{2-\epsilon})$ for some $\epsilon > 0$. From the result of [2], it follows that the SETH would be falsified, and the theorem is proved. ◀

Note that, from the above theorem and from Lemma 1, it follows that the same result holds for the LDT distance. Moreover, the proof of the above theorem gives strong evidence that a sub-quadratic $(3/2 - \epsilon)$-approximation algorithm for the diameter may be very hard to find, even for undirected unweighted link streams.

3 Computing the EAT diameter

In this section we focus on the EAT distance, and we propose a quite simple algorithm for computing the diameter of a weighted link stream. As we already said in the introduction, the algorithm follows the approach used in the case of graphs, which consists in sorting the nodes of the link stream, in computing, for each node in the given order, the eccentricity of the node, and in updating a lower bound on the value of the diameter and an upper bound on the backward eccentricities of the remaining nodes, until the upper bound becomes less than or equal to the lower bound. In the case of the EAT distance, nodes are sorted with respect to their last "entering" time, i.e. $\delta(v) = \max_{t,\lambda:\exists(u,v,t,\lambda)\in\mathbb{E}}\{t + \lambda\} - t_\alpha$, the lower bound is the maximum backward eccentricity computed so far, and the upper bound is the difference between the entering time of the next node that has been examined and t_α. This strategy is formalised in Algorithm 1.

▶ **Lemma 3.** *Let $v_1, \ldots, v_n$ be the ordering of the nodes in V with respect to $\delta(\cdot)$. For each $i \geq 1$, $\max_{j=1}^{i} ECCB_{EAT}(v_j) \leq \varnothing_{EAT}$ and $\max_{j=i}^{n} ECCB_{EAT}(v_j) \leq \delta(v_i)$.*

Proof. The first inequality is obvious, since $\varnothing_{\text{EAT}} = \max_{i=1}^{n} \text{ECCB}_{\text{EAT}}(v_i)$. The second inequality follows from the fact that, for each $v_j \in V$, we have $\text{ECCB}_{\text{EAT}}(v_j) \leq \delta(v_j)$, and that, because of the ordering, for each $j \geq i$, we have $\delta(v_j) \leq \delta(v_i)$, which implies $\max_{j=i}^{n} \text{ECCB}_{\text{EAT}}(v_j) \leq \max_{j=i}^{n} \delta(v_j) \leq \delta(v_i)$. ◀

▶ **Theorem 4.** *Algorithm 1 correctly computes the EAT diameter.*

Proof. Because of the previous lemma, we have that, at the end of each iteration of the **while** loop, the value of lb is a lower bound on $\varnothing_{\text{EAT}}$, while the value of ub is an upper bound on the backward eccentricity of all the remaining nodes to be visited. Since $\varnothing_{\text{EAT}} =$

$\max_{i=1}^{n} \text{ECCB}_{\text{EAT}}(v_i)$, sooner or later the value of lb has to become equal to $\varnothing_{\text{EAT}}$, and the value of ub has to become less than or equal to $\varnothing_{\text{EAT}}$. When this happens, the loop stops, and the algorithm correctly returns the value of lb. ◀

Algorithm 1 EAT diameter.

Input : Weighted link stream $(V, \overrightarrow{\mathbb{E}})$ with n nodes
Output : Diameter $\varnothing_{\text{EAT}}$

foreach $v \in V$ **do**
  $\delta(v) \leftarrow \max_{t,\lambda:\exists(u,v,t,\lambda)\in\mathbb{E}}\{t+\lambda\} - t_\alpha$
sort nodes $v_1, \ldots, v_n$ in non-increasing ordering with respect to $\delta(\cdot)$
$lb \leftarrow 0$
$i \leftarrow 1$
$ub \leftarrow \delta(v_i)$
while $ub > lb$ **do**
  $lb \leftarrow \max\{lb, \text{ECCB}_{\text{EAT}}(v_i)\}$
  $i \leftarrow i+1$ **if** $i \leq n$ **then** $ub \leftarrow \delta(v_i)$
return lb

Algorithm 2 Pivot-diameter.

Input : Weighted link stream $(V, \mathbb{E})$, set $P \subseteq V \times T$, distance $\text{D} \in \{\text{EAT}, \text{FT}, \text{ST}\}$
Output : Pivot-diameter $\varnothing_{\text{D}}^{P}$

$lb \leftarrow 0,\ ub \leftarrow \infty,\ \hat{A}_P \leftarrow \emptyset,\ \hat{B}_P \leftarrow \emptyset$
while $ub > lb$ **do**
  $\text{M} \leftarrow \arg\max_{p\in P} \text{UB}_{\text{D}}(p)$
  $ub \leftarrow \text{UB}_{\text{D}}(\text{M})$
  if $ub = -\infty$ **then return** lb
  $(v, w) \leftarrow \arg\max_{z\in A_P\setminus\hat{A}_P, y\in B_P\setminus\hat{B}_P} \text{UB}_{\text{D}}(\text{M}, z, y)$
  $lb \leftarrow \max\{lb, \text{GETLOWERBOUND}(v, w, ub)\}$
return lb

Function GETLOWERBOUND(v, w, ub)
  $y \leftarrow \text{ECCB}_{\text{D}}^{P}(w)$, Add w to $\hat{B}_P$
  if $y \geq ub \vee \text{D} = \text{EAT}$ **then return** y
  $z \leftarrow \text{ECCF}_{\text{D}}^{P}(v)$, Add v to $\hat{A}_P$
  return $\max\{z, y\}$

Note that, by applying Lemma 1, Algorithm 1 can also be used to compute the LDT diameter. It should be clear, that, in the worst case, Algorithm 1 has to execute a backward best path search starting from each node of the link stream. That is, the worst-case time complexity of the algorithm is $O(n \cdot \text{T-TIME}_{\text{D}}(n, m))$. However, we will experimentally show that, in the case of real-world link streams, the number of searches that have to be performed is much lower than the number of nodes, thus making the algorithm extremely efficient.

4 Pivot-diameter

Let $(V, \mathbb{E})$ be a weighted link stream, and let T be the set of the starting times of all temporal edges in $\mathbb{E}$. Given a subset P of $V \times T$, let $R(P) \subseteq V \times V$ be the set of pairs defined as follows: $R(P) = \{(u, v) \mid \exists (x, t) \in P : d_{\text{EAT}}^{[t_\alpha, t]}(u, x) < \infty \wedge d_{\text{EAT}}^{[t, t_\omega]}(x, v) < \infty\}$. In other words, P is a set of pivots, and $R(P)$ contains only pairs of nodes (u, v) such that u can reach v passing through the node x at time t, for some pivot $(x, t) \in P$. By restricting our attention to the set $R(P)$, we are sure that the pairs we are considering are connected by at least one path (in particular, a path passing through a pivot). However, while analyzing the diameter restricted to these pairs (u, v), we will consider *all* the possible paths and not only the ones passing through the pivots. Formally, for any node u, let $A_P(u) = \{v : (v, u) \in R(P)\}$ and $B_P(u) = \{v : (u, v) \in R(P)\}$. Moreover, let $A_P = \{u : B_P(u) \neq \emptyset\}$ and $B_P = \{u : A_P(u) \neq \emptyset\}$. For any $\text{D} \in \{\text{EAT}, \text{LDT}, \text{FT}, \text{ST}\}$,

- the *forward pivot-eccentricity* of a node $u \in A_P$ is $\text{ECCF}_{\text{D}}^{P}(u) = \max_{v\in B_P(u)} d_{\text{D}}(u, v)$, and
- the *backward pivot-eccentricity* of a node $u \in B_P$ is $\text{ECCB}_{\text{D}}^{P}(u) = \max_{v\in A_P(u)} d_{\text{D}}(v, u)$.

The *pivot-diameter* is then defined as $\varnothing_{\text{D}}^{P} = \max_{(u,v)\in R(P)} d_{\text{D}}(u, v) = \max_{u\in A_P} \text{ECCF}_{\text{D}}^{P}(u) = \max_{u\in B_P} \text{ECCB}_{\text{D}}^{P}(u)$. In this section, we focus on the pivot-diameter computation problem, that is, *given a weighted link stream and a set $P \subseteq V \times T$, compute the pivot-diameter with respect to any $\text{D} \in \{EAT, LDT, FT, ST\}$* (note that thanks to Lemma 1, we will neglect the LDT distance, as this case can be reduced to the EAT distance case after a suitable transformation of the graph). Let us observe that the pivot-diameter computation problem is also hard

to compute in time sub-quadratic to the number of temporal edges: indeed, the very same reduction described in the proof of Theorem 2 can be used, by choosing $P = X \times \{1, 2, 3\}$. In the following, we will assume that $T \subseteq \mathbb{N}^+$, that is, all time instants are positive integers (in order to deal with the negative time instants introduced by the transformation in Lemma 1, it is sufficient to perform a "temporal shift" of the link stream).

A simple algorithm for computing the pivot-diameter (denoted as PIVOT-TB$_{\text{D}}^P$) first computes the sets A_P and B_P, by simply performing, for each pivot $p = (x, t) \in P$, a backward best path search in the interval $[t_\alpha, t]$ and a forward best path search in the interval $[t, t_\omega]$, both starting from x. Once computed A_P and B_P, if $|A_P| \leq |B_P|$, then the algorithm computes, for each node u in A_P, the value ECCF$_{\text{D}}^P(u)$, and returns the maximum among all such values. Otherwise (that is, $|B_P| < |A_P|$) the algorithm computes, for each node u in B_P, the value ECCB$_{\text{D}}^P(u)$, and return the maximum among all such values.

We now propose another algorithm (called PIVOT-IFUB$_{\text{D}}$) which is, once again, inspired by the approach used for computing the diameter of graphs (see Algorithm 2). This algorithm, once computed the sets A_P and B_P, sorts the nodes of the link stream, computes, for each node in the given order, the (forward or backward) pivot-eccentricity of the node, and updates a lower bound on the value of the pivot-diameter and an upper bound on the pivot-eccentricities of the remaining nodes, until the upper bound becomes less than or equal to the lower bound. Once again, a key ingredient of the algorithm is the order of the nodes, which must be able to guarantee an effective non-trivial upper bound. Moreover, this upper bound should be easily obtainable in order to do not burden the computation. In the case of PIVOT-IFUB$_{\text{D}}$, all these aspects are guided by the pivots. In particular, each pivot ensures an upper bound on the distance between pairs of nodes it connects, and, for all pairs of nodes (v, w) in $R(P)$, we have at least one upper bound on their distance, which is given by the temporal paths passing through a pivot. Hence, we first select the pivot M giving the worst upper bound (this upper bound becomes the new upper bound of the algorithm), and we then select the corresponding pair of nodes (v, w). We then compute the forward pivot-eccentricity of v and the backward pivot-eccentricity w, eventually improving the lower bound of the algorithm. It is worth noting that selecting the pivot M and the pair (v, w) can be done in a fast way, as we can, during a preprocessing phase of the algorithm, compute the distances from and to all the pivots in P and, for each pivot $p \in P$, we can sort the nodes in $A_{\{p\}}$ and in $B_{\{p\}}$ in non-increasing order with respect to their distance from and to p, respectively.

The above description clearly depends on the distance D. In the following, we first precisely instantiate, for each distance, the above bounds, we then prove properties on the lower bound and on the upper bound, in order to asses the correctness of the algorithm. Given a pivot $p = (x, t) \in P$ and two nodes u and v such that $d_{\text{EAT}}^{[t_\alpha, t]}(u, x) < \infty$ and $d_{\text{EAT}}^{[t, t_\omega]}(x, v) < \infty$, we define $\text{UB}_{\text{EAT}}(p, u, v) = d_{\text{EAT}}^{[t, t_\omega]}(x, v)$, $\text{UB}_{\text{FT}}(p, u, v) = d_{\text{LDT}}^{[t_\alpha, t]}(u, x) - t_\omega + d_{\text{EAT}}^{[t, t_\omega]}(x, u)$, and $\text{UB}_{\text{ST}}(p, u, v) = d_{\text{ST}}^{[t_\alpha, t]}(u, x) + d_{\text{ST}}^{[t, t_\omega]}(x, u)$. For any $\text{D} \in \{\text{EAT}, \text{FT}, \text{ST}\}$, $d_{\text{D}}(u, v) \leq \text{UB}_{\text{D}}(p, u, v)$. Indeed, $\text{UB}_{\text{D}}(p, u, v)$ is the "cost" of a path going from u to x and then from x to v, where the cost is computed accordingly to the distance D. This concatenation of paths is a valid temporal path as the path arrives in x at most at time t and leaves from x at least at time t. For completeness, if $d_{\text{EAT}}^{[t_\alpha, t]}(u, x) = \infty$ or $d_{\text{EAT}}^{[t, t_\omega]}(x, v) = \infty$, we define $\text{UB}_{\text{D}}(p, u, v) = -\infty$ (this maybe non-intuitive definition allows us to deal with the search of the maximum values in a more compact way). In Algorithm 2, $\hat{A}_P \subseteq A_P$ and $\hat{B}_P \subseteq B_P$ are the set of nodes for which a `ssbp`$_{\text{D}}$ and a `stbp`$_{\text{D}}$, respectively, has been executed. Moreover, in the algorithm $\text{UB}_{\text{D}}(p)$ denotes the value $\max_{z \in A_P \setminus \hat{A}_P, y \in B_P \setminus \hat{B}_P} \text{UB}(p, z, y)$.

▶ **Lemma 5.** *At any iteration of the* ***while*** *loop of Algorithm 2, if* $\text{D} \in \{\text{FT}, \text{ST}\}$*, then* $lb = \max\{\max_{v \in \hat{B}_P}\{ECCB_D^P(v)\}, \max_{v \in \hat{A}_P}\{ECCF_D^P(v)\}\}$*, otherwise* $lb = \max_{v \in \hat{B}_P}\{ECCB_{EAT}^P(v)\}$.

Proof. It immediately follows from the definition of the GETLOWERBOUND function. ◀

▶ **Lemma 6.** *Let* $\mathrm{D} \in \{FT, ST, EAT\}$. *At any iteration of the* ***while*** *loop of Algorithm 2, for any node* $u \in A_P \setminus \hat{A}_P$ *and* $v \in B_P \setminus \hat{B}_P$, $d_{\mathrm{D}}(u,v) \leq ub$.

Proof. We have already observed that, for any pivot $p \in P$, $d_{\mathrm{D}}(u,v) \leq \mathrm{UB}_{\mathrm{D}}(p,u,v)$. Hence, $d_{\mathrm{D}}(u,v) \leq \max_{p \in P} \mathrm{UB}(p,u,v)$. Since $u \in A_P$ and $v \in B_P$, then there is at least one pivot $p = (x,t)$ such that $d_{\mathrm{EAT}}^{[t_\alpha,t]}(u,x) < \infty$ and $d_{\mathrm{EAT}}^{[t,t_\omega]}(x,v) < \infty$. This implies that $\max_{p \in P} \mathrm{UB}(p,u,v) \neq -\infty$. Moreover, we have that

$$\max_{p \in P} \mathrm{UB}(p,u,v) \leq \max_{p \in P} \max_{z \in A_P \setminus \hat{A}_P, y \in B_P \setminus \hat{B}_P} \mathrm{UB}(p,z,y) = \max_{p \in P} \mathrm{UB}(p) = ub,$$

where the inequality holds since $u \in A_P \setminus \hat{A}_P$ and $v \in B_P \setminus \hat{B}_P$, while the remaining two equalities follow from the definition of $\mathrm{UB}(p)$ and from the assignment at Line 4 of Algorithm 2. The lemma thus follows. ◀

▶ **Lemma 7.** *At any iteration of the* ***while*** *loop of Algorithm 2, if* $\mathrm{D} \in \{FT, ST\}$, *then, for any node* $u \in B_P \setminus \hat{B}_P$ *(resp.* $A_P \setminus \hat{A}_P$*),* $ECCB_{\mathrm{D}}^P(u)$ *(resp.* $ECCF_{\mathrm{D}}^P(u)$*) is bounded by* $\max\{lb, ub\}$. *Otherwise, for any node* $u \in B_P \setminus \hat{B}_P$, $ECCB_{EAT}^P(u)$ *is bounded by* $\max\{lb, ub\}$.

Proof. Let us suppose there is a node $v \in B_P \setminus \hat{B}_P$ such that $\mathrm{ECCB}_{\mathrm{D}}^P(v) > ub$, with $\mathrm{D} \in \{\mathrm{EAT}, \mathrm{FT}, \mathrm{ST}\}$. Let u be the node in A_P such that $d_{\mathrm{D}}(u,v) = \mathrm{ECCB}_{\mathrm{D}}^P(v)$. Note that u must exist and it is such that $\mathrm{ECCF}_{\mathrm{D}}^P(u) \geq \mathrm{ECCB}_{\mathrm{D}}^P(v)$. If $u \in A_P \setminus \hat{A}_P$ (this is the only possible case when $\mathrm{D} = \mathrm{EAT}$, since $\hat{A}_P = \emptyset$), then from Lemma 6 it follows that $d_{\mathrm{D}}(u,v) \leq ub$, which is a contradiction. Otherwise (that is, $u \in \hat{A}_P$), from Lemma 5 it follows that $lb \geq \mathrm{ECCF}_{\mathrm{D}}^P(u) \geq \mathrm{ECCB}_{\mathrm{D}}^P(v)$. For $\mathrm{D} \in \{\mathrm{FT}, \mathrm{ST}\}$, the proof for nodes in $A_P \setminus \hat{A}_P$ is similar. The lemma is thus proved. ◀

▶ **Theorem 8.** *Algorithm 2 correctly computes the pivot-diameter.*

Proof. It immediately follows from Lemma 6 and Lemma 7. ◀

The proof of the next theorem is given in Appendix.

▶ **Theorem 9.** *Algorithm 2 computes the pivot-diameter in* $O(|A_P| \cdot \textsc{s-time}_{\mathrm{D}}(n,m) + |B_P| \cdot \textsc{t-time}_{\mathrm{D}}(n,m) + |P| \cdot (\textsc{s-time}_{\mathrm{D}}(n,m) + \textsc{t-time}_{\mathrm{D}}(n,m) + n \log n))$ *time, using space* $O(|P| \cdot n + \textsc{s-space}_{\mathrm{D}}(n,m) + \textsc{t-space}_{\mathrm{D}}(n,m))$.

As we will see in the next section, an effective choice of the cardinality of P is logarithmic in the number of nodes: hence, in the worst case, the time complexity of our algorithm is the same as the one of the TB_{D} algorithm (if poly-logarithmic factors are ignored). This time is clearly bounded by $O(n \cdot (\textsc{s-time}_{\mathrm{D}}(n,m) + \textsc{t-time}_{\mathrm{D}}(n,m)) + n \log^2 n)$, and, looking at the costs in Table 1, it does not contradict the computational lower bound.

5 Experimental Results

This section is devoted to show our experimental results for the different notions of distance we have considered. After introducing our experimental testbed, we organize the results as follows. We show the performance of the temporal double sweep described in the introduction. We then show the performance of our algorithm for computing $\varnothing_{\mathrm{EAT}}$ and $\varnothing_{\mathrm{LDT}}$, which is described in Section 3. We finally focus on the pivot-diameter, whose algorithm has been described in Section 4.

Table 2 Our dataset. The meaning of the columns is described in Section 5.

Network name	n	m	t_ω-t_α	$\mathcal{R}$	$\varnothing_{\text{EAT}}$	$\varnothing_{\text{LDT}}$	$\varnothing_{\text{FT}}$	$\varnothing_{\text{ST}}$	Ref.
PUBLIC TRANSPORT NETWORKS									
KUOPIO	549	30 574	73 920	216 630	60 120	73 260	53 940	36 240	[13]
RENNES	1 407	107 713	73 320	1 641 208	63 660	70 980	42 540	8 760	[13]
GRENOBLE	1 547	113 437	78 780	1 265 735	75 540	75 180	42 180	12 300	[13]
VENICE	1 874	113 933	89 160	2 354 707	79 080	85 020	67 020	8 040	[13]
BELFAST	1 917	121 195	67 920	3 040 354	66 360	66 900	61 560	9 360	[13]
CANBERRA	2 764	122 690	65 760	5 754 287	62 280	65 700	43 320	8 700	[13]
TURKU	1 850	131 684	75 625	2 966 925	65 615	75 545	53 225	10 795	[13]
LUXEMBOURG	1 367	178 052	72 780	1 818 761	62 160	72 720	50 820	4 920	[13]
NANTES	2 353	194 572	76 680	4 320 287	74 040	72 360	61 680	18 780	[13]
DETROIT	5 683	214 853	90 660	29 990 674	76 260	87 660	54 352	17 174	[13]
TOULOUSE	3 329	222 749	73 920	9 525 234	72 300	73 200	53 940	12 900	[13]
PALERMO	2 176	224 260	76 200	4 734 976	27 505	35 659	9 669	7 139	[13]
BORDEAUX	3 435	236 489	78 365	9 933 813	76 025	76 020	56 520	9 620	[13]
WINNIPEG	5 079	332 522	77 808	25 519 193	67 488	75 977	56 545	6 594	[13]
BRISBANE	9 645	386 175	76 860	70 598 864	74 220	75 060	64 320	17 700	[13]
DUBLIN	4 571	399 875	75 471	15 801 421	72 591	72 892	60 929	9 141	[13]
ADELAIDE	7 548	402 933	76 200	50 007 666	73 745	76 200	54 498	14 810	[13]
LISBON	7 073	525 114	87 424	14 065 989	82 088	84 578	68 582	12 434	[13]
PRAGUE	5 147	621 545	90 660	19 181 301	90 660	90 660	84 900	17 280	[13]
HELSINKI	6 986	664 507	87 960	41 724 039	86 520	85 320	72 720	27 720	[13]
BERLIN	4 601	1 019 012	91 200	20 931 583	83 880	90 300	79 920	7 260	[13]
ROME	7 869	1 049 202	90 382	56 395 585	80 648	89 677	72 017	11 470	[13]
MELBOURNE	19 493	1 089 555	86 400	294 973 451	79 260	82 500	61 260	34 595	[13]
SYDNEY	24 063	1 234 097	91 440	416 037 155	90 780	91 260	81 840	50 936	[13]
PARIS	11 950	1 807 200	81 660	90 427 303	80 340	79 620	71 460	19 560	[13]
SOCIAL NETWORKS									
TOPOLOGY	34 759	99 019	2 016 004	146 028 435	2 016 004	2 016 004	2 016 004	20	[17]
ELEC	7 119	103 675	119 088 241	5 736 331	119 084 221	119 088 241	119 038 621	10	[17]
FACEBOOK-WOSN-WALL	46 953	876 020	137 462 861	617 698 414	137 461 023	132 709 413	132 527 766	39	[17]
COLLEGE	1 900	59 835	16 736 182	1 794 245	16 736 043	16 621 304	16 113 324	17	[15]
SX-MATHOVERFLOW-A2Q	88 577	107 581	203 068 634	47 765 186	203 068 634	203 068 634	202 876 639	18	[15]
SX-MATHOVERFLOW-C2A	88 577	195 330	203 055 529	33 076 882	203 055 529	203 055 529	202 094 959	20	[15]
SX-MATHOVERFLOW-C2Q	88 581	203 639	202 990 935	24 548 566	202 983 123	202 778 885	201 772 254	17	[15]
EMAIL-EU-CORE	1 005	332 334	69 459 255	770 833	69 430 695	69 439 852	44 641 379	15	[15]
SX-ASKUBUNTU-A2Q	515 274	280 102	225 833 890	*210 462 953	225 833 890	225 833 890	≥208 943 421	≥19	[15]
SX-ASKUBUNTU-C2Q	515 281	327 513	176 894 703	*56 918 856	176 894 645	176 893 633	≥176 893 575	≥18	[15]
SX-ASKUBUNTU-C2A	515 255	356 822	208 942 104	*410 760 018	208 939 593	208 940 523	≥208 401 245	≥18	[15]
SX-SUPERUSER-A2Q	567 309	430 033	239 613 340	*305 607 123	239 613 340	239 613 340	≥237 851 331	≥25	[15]
SX-MATHOVERFLOW	88 581	506 550	203 069 368	*191 433 809	203 068 737	202 999 190	≥202 991 055	≥20	[15]
SX-SUPERUSER-C2Q	567 316	479 067	239 293 899	*83 514 878	239 293 899	239 293 899	≥225 626 623	≥22	[15]
SX-SUPERUSER-C2A	567 301	534 239	236 358 777	*967 867 967	236 354 503	236 358 777	≥235 553 110	≥20	[15]
SX-ASKUBUNTU	515 281	964 437	225 834 463	*4 098 469 692	225 834 443	223 600 507	≥223 599 719	≥23	[15]
SX-SUPERUSER	567 316	1 443 339	239 614 929	*7 720 583 649	239 614 929	239 614 929	≥239 425 973	≥25	[15]
WIKI-TALK-TEMPORAL	1 140 149	7 833 140	200 483 883	*80 939 303 499	200 483 883	196 754 851	≥186 398 389	≥24	[15]

Computing Platform and Source Code. Our computing platform is a machine with Intel(R) Xeon(R) CPU E5-2620 v3 at 2.40 GHz, 24 virtual cores, 128 GB RAM, and running Ubuntu Linux version 4.4.0-22-generic (machine available at Dipartimento di Informatica, Università di Pisa, Italy). The code has been written in Python 3 and it is available at `github.com/marcocalamai/Link-stream-diameter`.

Dataset. Table 2 reports the set of link streams we have used for our experiments. The upper part refers to public transport networks, while the lower part refers to social networks. The former link streams are weighted while the latter are unweighted. We report the number of nodes n, the number of temporal edges m, the spectrum $t_\omega - t_\alpha$ of times where edges appear, and the diameter $\varnothing_{\text{D}}$ for $\text{D} \in \{\text{EAT}, \text{LDT}, \text{FT}, \text{ST}\}$. We also report $\mathcal{R}$, which is the number of pairs (u, v) such that $d_{\text{D}}(u, v) < \infty$, as this quantity, together with the diameter values will be compared to our results concerning the pivot-diameter. The values of $\varnothing_{\text{D}}$ have been computed by making use of TB_{D}: whenever, in the table, we write $\varnothing_{\text{D}} \geq y$, it means that we are reporting as y the best lower bound computed so far during our experiments.

For computing $\mathcal{R}$, we have used TB_{EAT}, when possible. In the cases marked with *, the value of $\mathcal{R}$ has been estimated using the method in [9].

Table 3 Number of times the lower bound returned by $2\text{SW}_{\text{D}}(k)$ and $\text{RS}_{\text{D}}(k)$ is tight.

D	NETWORKS (Number of Nets whose $\varnothing_{\text{D}}$ is known)	Number of times the returned lower bound is tight							
		$k=4$		$k=4\log_2 n$		$k=8\log_2 n$		$k=16\log_2 n$	
		$2\text{SW}_{\text{D}}(k)$	$\text{RS}_{\text{D}}(k)$	$2\text{SW}_{\text{D}}(k)$	$\text{RS}_{\text{D}}(k)$	$2\text{SW}_{\text{D}}(k)$	$\text{RS}_{\text{D}}(k)$	$2\text{SW}_{\text{D}}(k)$	$\text{RS}_{\text{D}}(k)$
EAT	PUBLIC TRANSPORT (25)	10	3	15	6	19	9	22	10
LDT	PUBLIC TRANSPORT (25)	10	0	19	1	23	1	23	2
FT	PUBLIC TRANSPORT (25)	4	0	18	0	19	1	21	2
	SOCIAL (8)	1	0	6	0	7	0	7	0
ST	PUBLIC TRANSPORT (25)	13	0	22	1	23	2	24	3
	SOCIAL (8)	0	0	1	1	1	1	1	1

As it can be seen and as already pointed out in the introduction, in the case of social networks, $\varnothing_{\text{EAT}}$ and $\varnothing_{\text{LDT}}$ are both very close to $t_\omega - t_\alpha$. This is expected because of the meaning of the link streams in the case of this kind of networks, whose behaviour is induced by new users being added. This fact makes the computation of $\varnothing_{\text{EAT}}$ and $\varnothing_{\text{LDT}}$ very easy: for this reason, we decided to exclude social networks when reporting our experimental results concerning $\varnothing_{\text{EAT}}$ and $\varnothing_{\text{LDT}}$.

Methods. In the following, we summarize our methods and competitors used in the remainder of the section. The subscript D refers to distances in $\{\text{EAT}, \text{LDT FT}, \text{ST}\}$.

Lower bounds for the diameter The following methods compute *lower bounds* for $\varnothing_{\text{D}}$.

- $2\text{SW}_{\text{D}}(k)$: select a set of $k/4$ nodes randomly chosen and return the best lower bound found by a double sweep, which have been described in the introduction. As each double sweep requires 4 visits, $2\text{SW}_{\text{D}}(k)$ performs exactly k visits in total.
- $\text{RS}_{\text{D}}(k)$: select a set of k nodes randomly chosen $v_1, \ldots, v_k$, and return $\max_i \text{ECCF}_{\text{D}}(v_i)$. Also this methods requires exactly k visits in total.

Computing exactly the diameter The following methods compute $\varnothing_{\text{EAT}}$ and $\varnothing_{\text{LDT}}$ *exactly*.

- EAT-ALG: apply Algorithm 1, described in Section 3.
- LDT-ALG: apply the transformation in Lemma 1 and then Algorithm 1.
- TB_{D}: for each node $v \in V$, compute $\text{ECCF}_{\text{D}}(v)$ and return the maximum value found.

Computing exactly the pivot-diameter The following methods, given a set of pivots $P \subseteq V \times T$, compute the pivot-diameter introduced in Section 4.

- PIVOT-IFUB$_{\text{D}}$: apply Algorithm 2.
- PIVOT-TB$_{\text{D}}$: this algorithm has been described in Section 4.

In order to evaluate the considered methods independently from the used platform, their performance have been expressed in terms of number of visits. For the sake of completeness, however, a rough estimation of the running time in seconds (on our computing platform) can be easily obtained using the running times of the visits reported in Table 8.

Computing lower bounds

This section is devoted to show the performance of $2\text{SW}_{\text{D}}(k)$ compared to the one of $\text{RS}_{\text{D}}(k)$, for different values of k. We evaluate the performance of both methods for $k \in \{4, 4\log_2 n, 8\log_2 n, 16\log_2 n\}$ and we summarize our results in Table 3 and in Table 6 in Appendix. Table 3 reports the number of times the lower bound returned by each method is tight, i.e. it is equal to $\varnothing_{\text{D}}$. We were able to do this only for the link streams whose diameter is known, i.e. the 25 public transport networks and 8 social networks. Looking at Table 3, we can see that, as expected, the performance of both algorithms improves by increasing k, i.e. the number of performed visits.

For EAT and LDT we report the results only for the 25 public transport networks in our dataset as we have seen that in the case of social networks the diameter is easy to find. For the public transport networks, $2\text{SW}_{\text{EAT}}(k)$ (resp. $2\text{SW}_{\text{LDT}}(k)$) is able to get very often tight

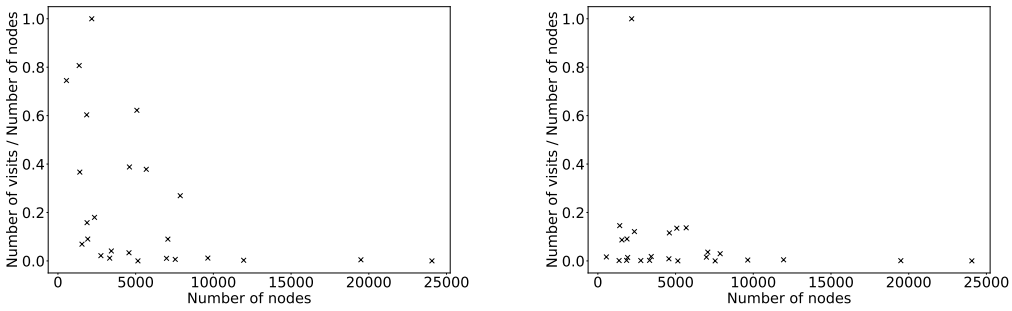
lower bounds, especially with $k = 16 \log_2 n$. For this value of k, in only 3 cases (resp. 2 cases) $2\text{SW}_{\text{EAT}}(k)$ (resp. $2\text{SW}_{\text{LDT}}(k)$) was not able to find a tight lower bound for $\varnothing_{\text{EAT}}$ (resp. $\varnothing_{\text{LDT}}$), namely for KUOPIO, MELBOURNE, and PARIS (resp. TOULOUSE and SYDNEY). Also in the case of FT, we can clearly see a very good performance of $2\text{SW}_{\text{FT}}(k)$. In the case $k = 16 \log_2 n$, it is able to find 21 tight lower bounds over the 25 public transport networks, and 7 tight lower bounds over the 8 social networks. The exceptional social network is EMAIL-EU-CORE, in which $2\text{SW}_{\text{FT}}(k)$ returns a lower bound 4% lower than the real value of the diameter.

In the case of $2\text{SW}_{\text{ST}}(k)$ we confirm the good performance of the double sweep in the case of public transport networks. However, we observe that its behaviour is worse in the case of social networks, as in only one case over 8, it returns a tight lower bound. In any case, we have verified that in the great majority of the cases, $2\text{SW}_{\text{ST}}(k)$ returns a lower bound greater than or equal to the one obtained by $\text{RS}_{\text{ST}}(k)$. This can be seen in Table 6 in Appendix, where we report such number of cases for each distance D. In this case, we were able to include all our 18 social networks, as we do not need to known the exact value of the diameter to perform the comparison. All in all, we can see that, fixing the number of visits, it is always more convenient to run $2\text{SW}_{\text{ST}}(k)$ instead of $\text{RS}_{\text{ST}}(k)$.

Computing $\varnothing_{\text{eat}}$ and $\varnothing_{\text{ldt}}$

In this section, we discuss the performance of EAT-ALG and LDT-ALG, i.e. the algorithms discussed in Section 3. These methods respectively compute $\varnothing_{\text{EAT}}$ and $\varnothing_{\text{LDT}}$, and have been compared respectively to TB_{EAT} and TB_{LDT}. The comparison is done in terms of number of visits performed: in particular, we report the ratio between the number of visits performed by our methods and n, which is indeed the number of visits required by TB_{D}. We report the results only for public transport networks.

Our results are summarized in Figure 1a, and further detailed in Table 7 in Appendix. In the case of EAT-ALG we can see that in 14 cases over 25 it performs less than 10% of the visits performed by TB_{EAT}, while in 20 cases over 25 it performs less than 50% of the visits. The cases where EAT-ALG had worst performance were KUOPIO, TURKU, LUXEMBOURG, WINNIPEG, and PALERMO. On the other hand, the cases where EAT-ALG performs better correspond to the three biggest link streams, namely MELBOURNE, SYDNEY, and PARIS, where it performs less than 1% visits. These results are even better if we look at the performance of LDT-ALG in Figure 1b. In this case we perform less than 15% of the visits required by TB_{LDT} for all the link streams except for PALERMO. In 16 cases over 25, we perform less than 3% of the visits. The reason behind these performances are deeply related to how the EAT-ALG and LDT-ALG work. Starting from the nodes with biggest $\delta(v)$, they perform one visit after the other, stopping when processing a node such that $\delta(w) = \varnothing_{\text{EAT}}$ (resp. $\delta(w) = \varnothing_{\text{LDT}}$). For this reason, both the methods need to compute the eccentricity of all the nodes v having $\delta(v) > \varnothing_{\text{D}}$. These nodes are relatively few in general, as shown in the case of ROME in Figures 4a and 4b in Appendix, respectively for EAT and LDT. In particular, Figure 4a and Figure 4b show for each δ the number of nodes in the link stream ROME having such δ. The values of δ are different for the plots, as the ones of Figure 4b refer to the values in the graph transformed applying Lemma 1. In both the cases, starting from the right of each plot, we need to perform the visit from all the nodes whose δ is at the right of the arrow marked as "diameter", whose number can be visually observed by the black mass at the right of the arrow. In the case of PALERMO, we have verified that the diameter is very low and all the nodes are at the right of the arrow both for EAT and LDT (see Figures 4c and 4d in Appendix).



(a) Visits performed by EAT-ALG wrt to n. **(b)** Visits performed by LDT-ALG wrt to n.

Figure 1 Ratio between the number of visits performed by EAT-ALG (resp. LDT-ALG) and n, where n is the number of visits required by TB_{EAT} (resp. TB_{LDT}), as a function of the number of nodes.

Computing the pivot-diameter

In the following, we focus on the computation of the pivot-diameter, using our algorithm PIVOT-IFUB$_{\text{D}}$ and the text-book algorithm PIVOT-TB$_{\text{D}}$. For the sake of brevity, in the following we discuss only the case where $\text{D} \in \{\text{FT}, \text{ST}\}$. For public transport networks, we have also computed the pivot-diameter for $\text{D} \in \{\text{LDT}, \text{EAT}\}$: we just report the results in Table 4.

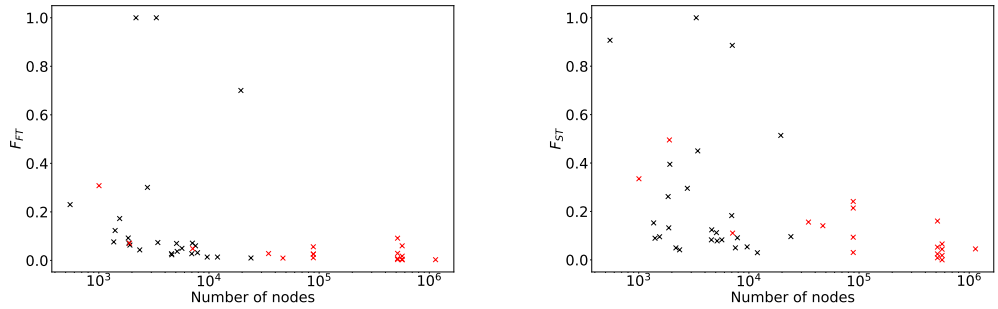
Choice of the pivots. We report our experiments in the case in which pivots are chosen as follows (we have analysed the performance of PIVOT-IFUB$_{\text{D}}$ with several choices of the pivots, with similar performance results: the choice we show here is a choice leading a sufficiently large coverage of the number of pairs analysed, corresponding to the case in which the pivot-diameter is more likely to be harder to compute). The set P is defined as $V' \times T'$, where V' are the top-$\log_2 n$ vertices with respect to the out-degree, i.e. number of temporal edges exiting from the vertex, and T' are 4 times equally spaced in the interval $[t_\alpha, t_\omega]$, i.e. $T' = \{t : \ t = t_\alpha + i/5 \cdot (t_\omega - t_\alpha), i \in \{1, 2, 3, 4\}\}$. For each graph in our dataset, we report the ratio $|P(H)|/|\mathcal{R}|$ in the second column of Table 4 and 5 (respectively, for public transport networks and social networks), that is the ratio between the number of pairs considered by the pivot-diameter and the number of pairs considered by diameter. As it can be seen, in the case of public transport networks, this choice of the pivots lead to a coverage of the pairs almost always very high, namely, for almost all the link streams, we cover more than 93% of the pairs. The exceptions are TOULOUSE, LISBON, and MELBOURNE, which are for this reason highlighted with a grey row. We have verified that this poor coverage is due to the fact that in these networks the nodes with the highest number of exiting temporal edges have a small out-degree in the underlying directed graph (see the concluding remarks). We also report the values of $|A_P|$ and $|B_P|$, which correspond to the number of vertices reaching (and reached by) the pivots (see Section 4). These values are crucial as $\min\{|A_P|, |B_P|\}$ is the number of visits needed by PIVOT-TB$_{\text{D}}$ (we report the ratio $\min\{|A_P|, |B_P|\}/n$ in Table 4 and 5). In the cases in which $\min\{|A_P|, |B_P|\}$ is constant or relatively small in practice, when compared to the number of nodes (like in the case of TOULOUSE and MELBOURNE), both PIVOT-TB$_{\text{D}}$ and PIVOT-IFUB$_{\text{D}}$ are effective, as they both spend linear time to find the pivot-diameter (see Theorem 9). In the case of social networks, the ratio $|R(B)|/|\mathcal{R}|$ seems to be in general smaller as also $\min\{|A_P|, |B_P|\}/n$ is very often below 10%. Even if one could be tempted to run PIVOT-TB$_{\text{D}}$, we will see that this choice is not convenient in any case, as PIVOT-IFUB$_{\text{D}}$ will perform much less visits to discover the pivot-diameter.

Pivot-diameter vs diameter. In the case in which pivots are chosen as above, the pivot-diameter is often very close to the diameter of the link stream (when we could verify it).

Table 4 Results for the pivot-diameter in the case of public transport networks (the choice of P is explained in the text). Rows in shadow gray correspond to graphs where the selected pivots lead to a small ratio $|R(P)|/|\mathcal{R}|$ (see also the concluding remarks).

PUBLIC TRANSPORT NETWORKS	$\frac{\lvert R(P)\rvert}{\lvert\mathcal{R}\rvert}$	$\lvert A_P\rvert$	$\lvert B_P\rvert$	$\frac{\min\{\lvert A_P\rvert,\lvert B_P\rvert\}}{n}$	$\frac{\varnothing^P_{\text{EAT}}}{\varnothing_{\text{EAT}}}$	F_{EAT}	$\frac{\varnothing^P_{\text{LDT}}}{\varnothing_{\text{LDT}}}$	F_{LDT}	$\frac{\varnothing^P_{\text{FT}}}{\varnothing_{\text{FT}}}$	F_{FT}	$\frac{\varnothing^P_{\text{ST}}}{\varnothing_{\text{ST}}}$	F_{ST}
KUOPIO	98.12%	452	472	82.33%	1	1	0.93	0.16	1	0.23	0.093	0.9
RENNES	98.51%	1298	1308	92.25%	1	0.23	0.94	0.07	0.91	0.12	0.97	0.09
GRENOBLE	93.56%	1 194	1 232	77.18%	0.72	0.3	0.94	0.11	1	0.17	1	0.1
VENICE	95.30%	1 531	1 512	80.68%	1	0.1	0.97	0.06	1	0.07	0.84	0.13
BELFAST	98.27%	1 707	1 780	89.04%	0.99	0.05	0.96	0.08	1	0.06	0.89	0.4
CANBERRA	98.59%	2 307	2 469	83.46%	0.99	0.05	0.98	0.04	1	0.3	1	0.3
TURKU	97.93%	1 722	1 690	91.35%	1	0.26	0.97	0.05	1	0.09	0.42	0.26
LUXEMBOURG	99.74%	1 336	1 358	97.73%	1	0.36	0.94	0.06	1	0.08	1	0.15
NANTES	97.14%	2 170	2 201	92.22%	0.91	0.05	0.98	0.04	1	0.04	1	0.04
DETROIT	99.04%	5 527	5 434	95.62%	0.98	0.38	0.95	0.1	1	0.05	1	0.08
TOULOUSE	0.00%	20	20	0.60%	0.41	1	0.07	1	0.03	1	0.13	1
PALERMO	100.00%	2 176	2 176	100%	1	1	1	1	1	1	1	0.05
BORDEAUX	98.50%	3 166	3 119	90.80%	0.9	0.44	0.89	0.86	1	0.07	0.97	0.45
WINNIPEG	97.81%	5 054	4 946	97.38%	1	0.4	1	0.02	1	0.07	1	0.11
BRISBANE	98.51%	8 228	8 657	85.31%	0.99	0.01	0.94	0.02	1	0.01	0.99	0.05
DUBLIN	99.15%	4 007	3 918	85.71%	0.97	0.03	0.97	0.03	1	0.03	1	0.08
ADELAIDE	98.82%	7 148	6 956	92.16%	0.9	0.47	0.93	0.06	1	0.06	0.90	0.05
LISBON	28.61%	2 052	2 059	29.01%	0.97	0.06	0.96	0.05	1	0.07	0.27	0.89
PRAGUE	98.27%	4 366	4 412	84.83%	0.99	0.02	0.99	0.02	1	0.04	0.96	0.08
HELSINKI	99.32%	6 375	6 552	91.25%	0.91	0.23	1	0.02	1	0.03	0.34	0.18
BERLIN	99.58%	4 573	4 561	99.13%	1	0.02	0.98	0.02	1	0.02	1	0.12
ROME	99.86%	7 511	7 499	95.30%	1	0.03	0.95	0.01	1	0.03	1	0.09
MELBOURNE	8.19%	17 545	1 459	7.48%	0.97	0.17	0.96	0.2	0.96	0.7	1	0.51
SYDNEY	93.94%	19 933	21 269	82.84%	0.97	0	0.99	0	1	0.01	0.53	0.1
PARIS	99.60%	9 878	9 858	82.49%	0.95	0.03	0.97	0.02	1	0.01	1	0.03

This is particularly evident for FT, while there are more exceptions for ST. To see this, for the cases in which we have the diameter of the link stream, in Table 4 and 5 we report, for each distance, the ratio between $\varnothing^P_{\text{D}}$ and $\varnothing_{\text{D}}$, where the former is computed by using our PIVOT-IFUB$_{\text{D}}$. In the case of public transport networks (Table 4), this can be easily explained by the fact that, very often, there is a large ratio $|R(P)|/|\mathcal{R}|$. In the case of social networks (Table 5), even though the ratio $|R(P)|/|\mathcal{R}|$ is lower, the ratio between $\varnothing^P_{\text{D}}$ and $\varnothing_{\text{D}}$ is not low, for FT and sometimes for ST.



(a) F_{FT} as a function of the number of nodes.

(b) F_{ST} as a function of the number of nodes.

Figure 2 For each link stream of n nodes, where the percentage of performed visits of PIVOT-IFUB$_{\text{D}}$ is F_{D}, we draw a cross (black for public transport networks and red for social networks) in position (n, F_{D}).

Table 5 Results for the pivot-diameter in the case of public transport networks (the choice of P is explained in the text). EAT and LDT distances are here neglected.

SOCIAL NETWORKS	$\frac{\lvert R(P)\rvert}{\lvert\mathcal{R}\rvert}$	$\lvert A_P\rvert$	$\lvert B_P\rvert$	$\frac{\min\{\lvert A_P\rvert,\lvert B_P\rvert\}}{n}$	$\frac{\varnothing^P_{\text{FT}}}{\varnothing_{\text{FT}}}$	F_{FT}	$\frac{\varnothing^P_{\text{ST}}}{\varnothing_{\text{ST}}}$	F_{ST}
TOPOLOGY	39.75%	14 402	4 316	12.42%	1	0.03	0.95	0.16
ELEC	63.86%	3 576	2 169	30.47%	1	0.05	0.9	0.11
FACEBOOK-WOSN-WALL	64.87%	13 720	32 591	29.22%	0.99	0.01	0.79	0.14
COLLEGE	80.52%	1 268	1 354	66.74%	1	0.07	0.71	0.5
SX-MATHOVERFLOW-A2Q	85.42%	4 866	11 753	5.49%	1	0.03	0.89	0.21
SX-MATHOVERFLOW-C2A	82.79%	7 993	5 082	5.74%	1	0.03	0.8	0.09
SX-MATHOVERFLOW-C2Q	86.40%	2 355	12 927	2.66%	1	0.06	0.88	0.24
EMAIL-EU-CORE	95.31%	818	924	81.39%	1	0.31	0.6	0.33
SX-ASKUBUNTU-A2Q	73.86%	6 572	52 406	1.27%		0.03		0.05
SX-ASKUBUNTU-C2Q	80.12%	1 671	52 590	0.32%		0.09		0.16
SX-ASKUBUNTU-C2A	72.49%	27 345	20 994	4.07%		0.01		0.02
SX-SUPERUSER-A2Q	76.27%	8 669	54 668	1.53%		0.02		0.02
SX-MATHOVERFLOW	87.79%	11 941	19 441	13.48%		0.01		0.03
SX-SUPERUSER-C2Q	89.36%	2 543	61 294	0.45%		0.06		0.07
SX-SUPERUSER-C2A	78.00%	39 088	31 982	5.64%		0.004		0.04
SX-ASKUBUNTU	75.10%	44 245	114 766	8.59%		0.003		0
SX-SUPERUSER	80.35%	67 470	136 455	11.89%		0.002		0.01
WIKI-TALK-TEMPORAL	54.93%	53 408	1 037 170	4.68%		0.003		0.04

Performance of the algorithms. In Figure 2, we report the ratio between the number of visits performed by PIVOT-IFUB$_{\text{D}}$ and $\min\{|A_P|, |B_P|\}$, which are the visits required by TB$_{\text{D}}$ (in the following, we denote this ratio as F_{D}). For each link stream of n nodes, we draw a cross in position (n, F_{D}) (black for public transport networks and red for social networks). The plot on the left refers to FT, while the plot on the right refers to ST. For the sake of completeness, the values of F_{D} are also reported in Table 4 and 5. As it can be seen, F_{FT} and F_{ST} indicates that PIVOT-IFUB$_{\text{D}}$ performs a number of visits which is very often much less than the ones performed by TB$_{\text{D}}$. In particular, for FT (Figure 2a), for public transport networks, F_{FT} is almost always less than 0.2 and for social networks it is less than 0.1. Exceptions correspond to TOULOUSE, MELBOURNE and EMAIL-EU-CORE, and this is not surprising as we can observe a relatively small $\min\{|A_P|, |B_P|\}$, which means that both PIVOT-IFUB$_{\text{D}}$ and TB$_{\text{D}}$ require linear time. In the case of ST (Figure 2b), the performance seems to be worse with respect to FT, but there is not doubt that running PIVOT-IFUB$_{\text{D}}$ is far more convenient than running TB$_{\text{D}}$. In the case of public transport networks F_{ST} is always smaller than 0.52, except for KUOPIO (the smallest graph), TOULOUSE, and LISBON, where both PIVOT-IFUB$_{\text{D}}$ and TB$_{\text{D}}$ are effective because of the few pairs in $R(P)$. In the case of social networks, F_{ST} seems to behave better, and, apart from COLLEGE, F_{ST} is always bounded by 0.34. In any case, the advantage of PIVOT-IFUB$_{\text{D}}$, for both FT and ST, is more evident when the number of nodes increases. Indeed, with social networks with more than 500 thousands nodes, the ratio F_{D} is always less than 0.07, apart from SX-ASKUBUNTU-C2Q (where F_{D} is bounded by 0.17).

6 Concluding remarks

In this paper, we have introduced the concept of pivot-diameter, we have given algorithms to compute it efficiently in practice, and we have seen that our choice of the pivots, i.e. choosing vertices with the maximum number of exiting temporal edges, leads very often to a large coverage of pairs. For the networks in our dataset with low coverage, we have additionally

verified that by simply choosing as pivots the top-degree vertices in the underlying directed static graph the coverage becomes higher than 98%. Even if a discussion about the many possible choices of pivots in order to maximize the coverage is outside the scope of this paper, we think that this problem deserves further (both theoretical and experimental) investigations to be addressed in a future work.

References

1 M. Borassi, D. Coudert, P. Crescenzi, and A. Marino. On computing the hyperbolicity of real-world graphs. In *Proc. 23rd Annual European Symposium on Algorithms*, pages 215–226, 2015.

2 M. Borassi, P. Crescenzi, and M. Habib. Into the square: On the complexity of some quadratic-time solvable problems. *Electron. Notes Theor. Comput. Sci.*, 322:51–67, 2016.

3 M. Borassi, P. Crescenzi, M. Habib, W. A. Kosters, A. Marino, and F. W. Takes. Fast diameter and radius bfs-based computation in (weakly connected) real-world graphs: With an application to the six degrees of separation games. *Theor. Comput. Sci.*, 586:59–80, 2015.

4 F. Brunelli, P. Crescenzi, and L. Viennot. On computing pareto optimal paths in weighted time-dependent networks. *Inf. Proc. Let.*, 168:106086, 2021.

5 A. Casteigts, J. G. Peters, and J. Schoeters. Temporal cliques admit sparse spanners. In *Proc. 46th ICALP*, pages 134:1–134:14, 2019.

6 P. Crescenzi, R. Grossi, M. Habib, L. Lanzi, and A. Marino. On computing the diameter of real-world undirected graphs. *Theor. Comput. Sci.*, 514:84–95, 2013.

7 P. Crescenzi, R. Grossi, C. Imbrenda, L. Lanzi, and A. Marino. Finding the diameter in real-world graphs - experimentally turning a lower bound into an upper bound. In *Proc. 18th Annual European Symposium on Algorithms*, pages 302–313, 2010.

8 P. Crescenzi, R. Grossi, L. Lanzi, and A. Marino. On computing the diameter of real-world directed (weighted) graphs. In *Proc. 11th International Symposium on Experimental Algorithms*, pages 99–110. Springer, 2012.

9 P. Crescenzi, C. Magnien, and A. Marino. Approximating the temporal neighbourhood function of large temporal graphs. *Algorithms*, 12(10):211, 2019.

10 P. Crescenzi, C. Magnien, and A. Marino. Finding top-k nodes for temporal closeness in large temporal graphs. *Algorithms*, 13(9):211, 2020.

11 IMDb. IMDb Datasets. `http://www.imdb.com/interfaces`.

12 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

13 R. Kujala, C. Weckström, R. Darst, M. Madlenocić, and J. Saramäki. A collection of public transport network data sets for 25 cities. *Sci. Data*, 5, 2018.

14 M. Latapy, T. Viard, and C. Magnien. Stream graphs and link streams for the modeling of interactions over time. *Soc. Netw. Anal. Min.*, 8(1):61:1–61:29, 2018.

15 J. Leskovec. Stanford Large Network Dataset Collection (SNAP). `http://snap.stanford.edu/data/index.html`.

16 M. Ley. DBLP - some lessons learned. *Proc. VLDB Endow.*, 2(2):1493–1500, 2009.

17 Institute of Web Science and Technologies. The Koblenz Network Collection. Available online: `http://konect.uni-koblenz.de`.

18 F. W. Takes and W. A. Kosters. Determining the diameter of small world networks. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management*, pages 1191–1196, 2011.

19 F. W. Takes and W. A. Kosters. Computing the eccentricity distribution of large graphs. *Algorithms*, 6(1):100–118, 2013.

20 Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *Proc. 14th KDD*, pages 990–998, 2008.

21 V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE FOCS*, pages 645–654, 2010.

22 H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. *Proc. of the VLDB Endowment*, 7(9):721–732, 2014.

23 H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu. Efficient algorithms for temporal path computation. *IEEE Trans. on Knowl. and Data Eng.*, 28(11):2927–2942, 2016.

24 B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. of Found. of Comp. Sci.*, 14(02):267–285, 2003.

A Proofs

A.1 Proof of Lemma 1

In order to prove the first assertion, it suffices to show that there exists a $[t_\alpha, t_\omega]$-compatible path from u to v in $(V, \mathbb{E})$ whose duration (respectively, travel time) is τ if and only if there exists a $[-t_\omega, -t_\alpha]$-compatible path from v to u in $(V, \mathbb{F})$ whose duration (respectively, travel time) is τ. Let $P = e_1 e_2 \dots e_k$ be a $[t_\alpha, t_\omega]$-compatible path from u to v in $(V, \mathbb{E})$, where $e_i = (u_i, v_i, t_i, \lambda_i)$ for any i with $1 \leq i \leq k$, $u_1 = u$, $v_k = v$, $t_1 \geq t_\alpha$, $t_k + \lambda_k \leq t_\omega$, and, for each i with $1 < i \leq k$, $u_i = v_{i-1}$ and $t_i \geq t_{i-1} + \lambda_{i-1}$. Since $-t_k - \lambda_k \geq -t_\omega$ and $-t_1 - \lambda_1 + \lambda_1 \leq -t_\alpha$, and since $t_i \geq t_{i-1} + \lambda_{i-1}$ if and only if $-t_{i-1} - \lambda_{i-1} \geq -t_i - \lambda_i + \lambda_i$, we have that $\rho(P) = \rho(e_k)\rho(e_{k-1}) \dots \rho(e_1)$ is a $[-t_\omega, -t_\alpha]$-compatible path from v to u in $(V, \mathbb{F})$. Since the travel times of the temporal edges have not been changed, we have that the travel time of P is equal to the travel time of $\rho(P)$. Moreover, the duration of P is equal to $t_k + \lambda_k - t_1$: since $t_k + \lambda_k - t_1 = -t_1 - \lambda_1 + \lambda_1 - (-t_k - \lambda_k)$, we have that P and $\rho(P)$ have also the same duration. The opposite direction can be proved similarly.

In order to prove the second assertion, it suffices to show that there exists a $[t_\alpha, t_\omega]$-compatible path from u to v in $(V, \mathbb{E})$ whose arrival (respectively, latest departure) time is τ if and only if there exists a $[-t_\omega, -t_\alpha]$-compatible path from v to u in $(V, \mathbb{F})$ whose departure (respectively, arrival) time is $-\tau$. As before, let $P = e_1 e_2 \dots e_k$ be a $[t_\alpha, t_\omega]$-compatible path from u to v in $(V, \mathbb{E})$, and let $\rho(P) = \rho(e_k)\rho(e_{k-1}) \dots \rho(e_1)$ be the corresponding $[-t_\omega, -t_\alpha]$-compatible path from v to u in $(V, \mathbb{F})$. The arrival (respectively, departure) time of P is $t_k + \lambda_k$ (respectively, t_1), while the departure (respectively, arrival) time of P is $-t_k - \lambda_k = -(t_k + \lambda_k)$ (respectively, $-t_1 - \lambda_1 + \lambda_1 = -t_1$). The opposite direction can be proved similarly, and this concludes the proof of the lemma.

A.2 Proof of Theorem 9

In the worst case, the number of iterations of the while loop is $O(n)$. The computation of the lower bound requires $O(\text{S-TIME}_\text{D}(n, m) + \text{T-TIME}_\text{D}(n, m))$ and it is the dominant part of the **while** loop, if we can speed up the computation of M at Line 3. To this aim we can perform the following precomputation. Let us define $\overline{\text{EAT}} = \overline{\text{FT}} = \text{LDT}$, $\underline{\text{EAT}} = \underline{\text{FT}} = \text{EAT}$, $\overline{\text{ST}} = \underline{\text{ST}} = \text{ST}$. For $\text{D} \in \{\text{EAT}, \text{FT}, \text{ST}\}$ and for each $p = (x, t) \in P$, we define π_p as the sequence of nodes $v \in A_{\{p\}}$ sorted in non-increasing order with respect to $d_{\overline{\text{D}}}^{[t_\alpha, t]}(v, x)$, and γ_P as the sequence of nodes $v \in B_{\{p\}}$ sorted in non-decreasing order with respect to $d_{\underline{\text{D}}}^{[t_i, t_\omega]}(x, v)$. This precomputation can be performed in $O(|P| \cdot (\max\{\text{S-TIME}_\text{D}(n, m), \text{T-TIME}_\text{D}(n, m)\} + n \log n))$. When Line 3 is performed, it is sufficient to consider, for each pivot p, the pair of nodes (u, v), where u is the leftmost element of π_p not in $\hat{A}_P$ and v is the leftmost element of γ_p not in $\hat{B}_P$. Hence, this line costs $O(|P|)$ time. Once M has been selected, the other lines, i.e. Line 4 and Line 6, cost $O(1)$ time. As a result, we obtain the time and space bounds of the theorem, where the space overhead $O(|P| \cdot n)$ is due to the space required for maintaining the result of the preprocessing.

B Tables and figures

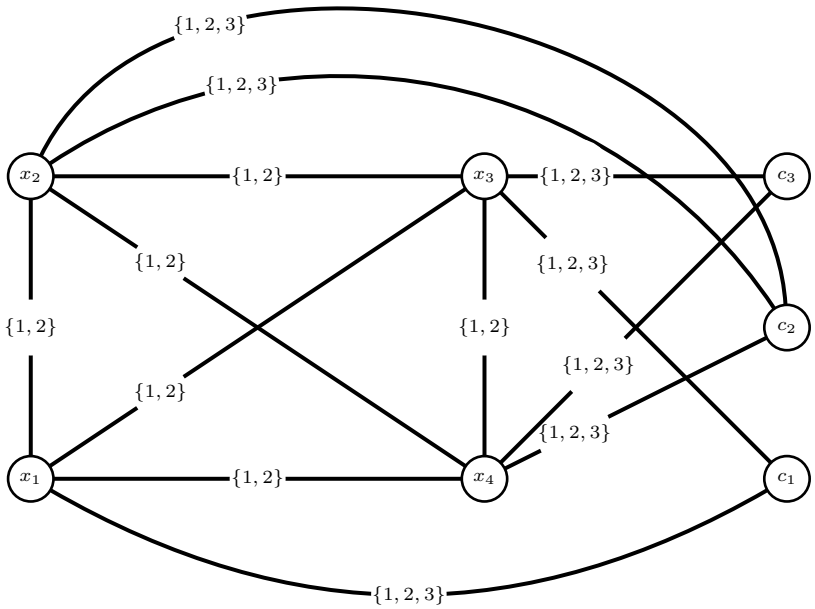
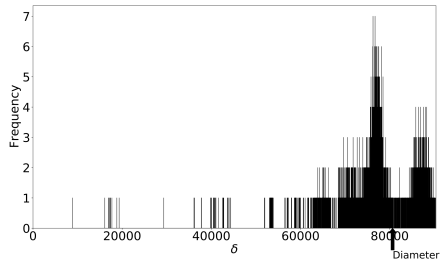


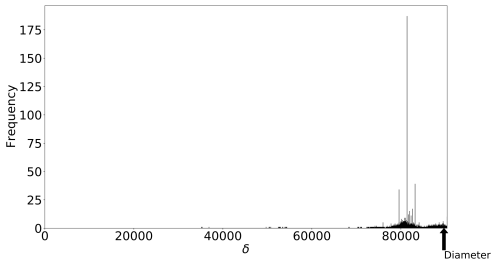
Figure 3 The reduction from disjoint sets to diameter computation. In this case, $c_1 = \{x_1, x_3\}$, $c_2 = \{x_2, x_4\}$, and $c_3 = \{x_3, x_4\}$. All temporal edges have travel time equal to 1. For any distance, the diameter is 3, and, indeed, c_1 and c_2 are disjoint.

Table 6 Number of times the lower bound returned by $2\text{SW}_\text{D}(k)$ is at least the one of $\text{RS}_\text{D}(k)$.

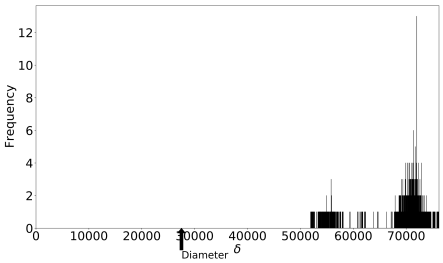
D	NETWORKS (Number of nets)	Number of times the lower bound returned by $2\text{SW}_\text{D}(k)$ is $\geq$ than the one of $\text{RS}_\text{D}(k)$			
		$k = 4$	$k = 4 \log_2 n$	$k = 8 \log_2 n$	$k = 16 \log_2 n$
EAT	PUBLIC TRANSPORT (25)	23	25	25	25
LDT	PUBLIC TRANSPORT (25)	23	25	25	25
FT	PUBLIC TRANSPORT (25)	25	25	25	25
	SOCIAL (18)	15	16	18	18
ST	PUBLIC TRANSPORT (25)	22	25	25	25
	SOCIAL (18)	16	16	18	18



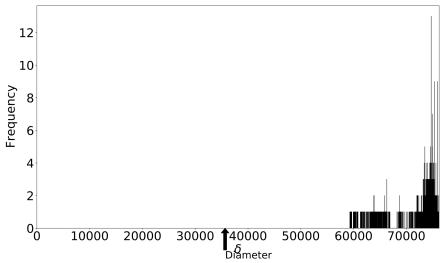
(a) Distribution of δ values for EAT in ROME.



(b) Distribution of δ values for LDT in ROME.



(c) Distribution of δ values for EAT in PALERMO.



(d) Distribution of δ values for LDT in PALERMO.

Figure 4 Distribution of δ values for EAT and LDT, for ROME and PALERMO. For each x the amount of vertices v having $\delta(v) = x$.

Table 7 Number of visits performed by EAT-ALG and LDT-ALG wrt to n, where n is the number of visits required by TB_{EAT} and TB_{LDT}. These values are plot in Figure 1a and Figure 1b as a function of n.

NETWORK	visits/n		NETWORK	visits/n	
	EAT	LDT		EAT	LDT
KUOPIO	74.50%	1.64%	WINNIPEG	62.20%	13.51%
RENNES	36.67%	14.57%	BRISBANE	1.17%	0.36%
GRENOBLE	6.92%	8.66%	DUBLIN	3.37%	0.88%
VENICE	15.80%	9.07%	ADELAIDE	0.62%	0.01%
BELFAST	9.02%	1.41%	LISBON	8.99%	3.62%
CANBERRA	2.17%	0.14%	PRAGUE	0.02%	0.02%
TURKU	60.32%	0.22%	HELSINKI	1.06%	1.55%
LUXEMBOURG	80.69%	0.15%	BERLIN	38.80%	11.56%
NANTES	17.98%	12.11%	ROME	26.93%	2.97%
DETROIT	37.83%	13.67%	MELBOURNE	0.47%	0.10%
TOULOUSE	1.14%	0.21%	SYDNEY	0.02%	0.02%
PALERMO	100.00%	100.00%	PARIS	0.27%	0.46%
BORDEAUX	4.10%	1.80%			

Table 8 Running time of our implementations of the visits reported in Table 1 for each of the networks in our dataset (mean in seconds and variance, over a random sample of 100 visits).

NETWORK	SINGLE SOURCE						SINGLE TARGET					
	EAT		LDT		FT		EAT		LDT		FT	
	MEAN(S)	VAR	MEAN(S)	VAR	MEAN(S)	VAR	MEAN(S)	VAR	MEAN(S)	VAR	MEAN(S)	VAR
KUOPIO	0.237	0.004	0.213	0.001	0.177	0.002	0.198	0.001	0.232	0.001	0.098	0.001	0.21	0.001
RENNES	0.776	0.039	0.759	0.004	0.637	0.016	0.658	0.005	0.778	0.003	0.336	0.001	0.732	0.01
GRENOBLE	0.754	0.085	0.721	0.002	0.528	0.016	0.53	0.009	0.73	0.002	0.351	0.001	0.6	0.014
VENICE	0.698	0.058	0.785	0.004	0.675	0.02	0.691	0.014	0.804	0.003	0.366	0.001	0.668	0.018
BELFAST	0.821	0.058	0.816	0.003	0.749	0.021	0.717	0.016	0.849	0.003	0.383	0.001	0.764	0.018
CANBERRA	0.629	0.028	0.838	0.005	0.793	0.035	0.797	0.02	0.893	0.002	0.381	0.001	0.847	0.027
TURKU	0.798	0.081	0.991	0.004	0.979	0.031	0.991	0.033	1.006	0.005	0.412	0.001	0.964	0.048
LUXEMBOURG	1.216	0.13	1.347	0.003	1.445	0.022	1.243	0.013	1.419	0.003	0.578	0.001	1.438	0.018
NANTES	1.315	0.136	1.374	0.008	1.192	0.053	1.175	0.028	1.391	0.006	0.603	0.001	1.159	0.045
DETROIT	1.565	0.142	1.666	0.007	1.653	0.026	1.536	0.019	1.693	0.012	0.67	0.001	1.661	0.066
TOULOUSE	1.431	0.158	1.598	0.012	1.444	0.082	1.408	0.056	1.621	0.011	0.694	0.001	1.492	0.02
PALERMO	1.961	0.089	1.843	0.007	1.835	0.01	1.718	0.017	1.836	0.012	0.718	0.001	1.898	0.02
BORDEAUX	1.743	0.16	1.747	0.009	1.666	0.095	1.771	0.087	1.801	0.015	0.736	0.001	1.761	0.055
WINNIPEG	2.782	0.206	2.636	0.028	2.985	0.08	3.062	0.169	2.674	0.017	1.045	0.001	2.884	0.21
BRISBANE	1.732	0.261	2.797	0.05	2.68	0.472	2.598	0.366	2.881	0.031	1.207	0.001	2.669	0.34
DUBLIN	1.566	0.017	2.992	0.062	3.241	0.392	3.118	0.391	3.097	0.06	1.281	0.001	3.002	0.578
ADELAIDE	1.598	0.014	2.975	0.026	3.387	0.311	3.675	0.278	3.268	0.038	1.261	0.001	3.297	0.411
LISBON	1.816	0.023	3.433	0.054	2.723	0.408	2.696	0.226	3.644	0.047	1.653	0.001	2.841	0.47
PRAGUE	2.504	0.057	4.595	0.112	4.46	1.142	4.038	0.404	4.891	0.143	2.097	0.021	4.671	1.181
HELSINKI	2.8	0.066	4.889	0.134	5.248	0.953	5.026	0.65	5.256	0.105	2.139	0.001	5.328	0.642
BERLIN	4.481	0.039	7.744	0.101	8.784	0.357	7.471	0.416	8.286	0.191	3.249	0.004	8.702	0.873
ROME	4.451	0.039	7.954	0.114	9.861	1.236	7.235	0.751	8.654	0.26	3.292	0.001	9.591	2.471
MELBOURNE	4.223	0.12	7.936	0.222	8.182	3.522	9.109	4.072	8.324	0.245	3.419	0.001	8.123	1.798
SYDNEY	4.739	0.2	9.164	0.369	8.641	4.41	9.529	3.859	9.402	0.61	3.963	0.001	8.749	3.64
PARIS	6.691	0.274	14.891	0.893	12.154	6.056	11.798	7.146	12.42	0.769	5.694	0.006	12.167	6.889
TOPOLOGY	0.185	0.001	0.371	0.001	0.258	0.007	0.291	0.008	0.312	0.001	0.19	0.001	0.263	0.006
ELEC	0.167	0.001	0.346	0.001	0.173	0.001	0.187	0.001	0.307	0.001	0.16	0.001	0.177	0.001
FACEBOOK-WOSN-WALL	1.436	0.008	3.338	0.061	1.597	0.117	1.82	0.197	2.752	0.031	1.37	0.001	1.91	0.14
COLLEGE	0.099	0.001	0.202	0.001	0.121	0.001	0.124	0.001	0.196	0.001	0.094	0.001	0.137	0.001
SX-MATHOVERFLOW-A2Q	0.164	0.001	0.498	0.001	0.218	0.001	0.23	0.001	0.417	0.001	0.168	0.001	0.229	0.001
SX-MATHOVERFLOW-C2A	0.294	0.001	0.789	0.001	0.362	0.001	0.386	0.002	0.659	0.001	0.305	0.001	0.369	0.002
SX-MATHOVERFLOW-C2Q	0.299	0.001	0.806	0.001	0.363	0.002	0.395	0.001	0.711	0.003	0.313	0.001	0.388	0.002
EMAIL-EU-CORE	0.541	0.001	1.078	0.004	0.746	0.031	0.843	0.05	0.96	0.005	0.509	0.001	0.876	0.017
SX-ASKUBUNTU-A2Q	0.43	0.001	1.915	0.006	0.783	0.001	0.88	0.003	1.605	0.005	0.441	0.001	0.85	0.008
SX-ASKUBUNTU-C2Q	0.5	0.001	2.01	0.009	0.847	0.001	0.948	0.002	1.727	0.007	0.514	0.001	0.92	0.005
SX-ASKUBUNTU-C2A	0.554	0.001	2.09	0.008	0.947	0.004	1.017	0.006	1.816	0.003	0.561	0.001	0.951	0.006
SX-SUPERUSER-A2Q	0.658	0.001	2.639	0.011	1.16	0.004	1.138	0.003	2.165	0.005	0.676	0.001	1.161	0.008
SX-MATHOVERFLOW	0.771	0.001	1.845	0.008	0.91	0.025	0.992	0.032	1.565	0.009	0.785	0.001	0.946	0.032
SX-SUPERUSER-C2Q	0.731	0.001	2.596	0.016	1.129	0.002	1.273	0.006	2.254	0.014	0.755	0.001	1.203	0.006
SX-SUPERUSER-C2A	0.814	0.001	2.875	0.012	1.331	0.017	1.385	0.008	2.424	0.011	0.826	0.001	1.346	0.016
SX-ASKUBUNTU	1.481	0.007	4.212	0.032	1.989	0.071	2.108	0.054	3.587	0.029	1.522	0.001	2.058	0.056
SX-SUPERUSER	2.288	0.022	6.013	0.077	2.801	0.102	3.126	0.19	5.102	0.073	2.262	0.002	3.035	0.26
WIKI-TALK-TEMPORAL	12.373	0.391	29.842	2.129	14.759	13.015	15.014	6.032	24.993	0.973	12.309	0.009	17.823	6.574

ST	
MEAN(S)	VAR
0.202	0.001
0.671	0.006
0.573	0.008
0.68	0.015
0.718	0.011
0.769	0.015
0.957	0.025
1.279	0.015
1.14	0.023
1.576	0.034
1.353	0.023
1.692	0.014
1.744	0.111
3.002	0.207
2.491	0.222
3.174	0.493
3.592	0.55
2.792	0.441
3.987	0.42
4.933	0.531
7.428	0.739
7.528	0.651
7.694	2.435
8.852	3.057
11.752	5.771
0.313	0.009
0.212	0.001
2.255	0.235
0.169	0.001
0.26	0.001
0.436	0.003
0.449	0.002
1.034	0.032
0.899	0.005
0.995	0.009
1.053	0.009
1.286	0.016
1.104	0.039
1.321	0.011
1.51	0.024
2.458	0.116
3.466	0.324
20.561	8.103