Faster maximal clique enumeration in large real-world link streams

Alexis Baudin[1], Clémence Magnien[1], and Lionel Tabourier[1]

[1]Sorbonne Université, CNRS, LIP6, F-75005 Paris

Abstract

Link streams offer a good model for representing interactions over time. They consist of links (b, e, u, v), where u and v are vertices interacting during the whole time interval $[b, e]$. In this paper, we deal with the problem of enumerating maximal cliques in link streams. A clique is a pair $(C, [t_0, t_1])$, where C is a set of vertices that all interact pairwise during the full interval $[t_0, t_1]$. It is maximal when neither its set of vertices nor its time interval can be increased. Some main works solving this problem are based on the famous Bron-Kerbosch algorithm for enumerating maximal cliques in graphs. We take this idea as a starting point to propose a new algorithm which matches the cliques of the instantaneous graphs formed by links existing at a given time t to the maximal cliques of the link stream. We prove its correctness and compute its complexity, which is better than the state-of-the art ones in many cases of interest. We also study the output-sensitive complexity, which is close to the output size, thereby showing that our algorithm is efficient. To confirm this, we perform experiments on link streams used in the state of the art, and on massive link streams, up to 100 million links. In all cases our algorithm is faster, mostly by a factor of at least 10 and up to a factor of 10^4. Moreover, it scales to massive link streams for which the existing algorithms are not able to provide the solution.

1 Introduction

The analysis of real-world interaction networks has recently made significant progress as the field evolved from static to dynamic representations. Indeed, the availability of temporal data, as well as the development of tools to describe and analyze them, have revealed the importance of the events' temporality. They allow understanding the structure and functioning of complex interacting systems such as computer networks [30], online social networks [46], human communication networks [38], or mobility networks [1, 21].

Static representations of interacting systems are usually based on graphs. Concerning dynamical networks, several formalisms have been developed at the same time. Some choose to represent a dynamical network as a sequence of static images of equal duration – see for instance [28]. This allows to use standard graph tools, but also raises issues about the existence of an adequate time window of analysis [19, 25]. In the same vein, *time varying graphs* [9] represent a dynamical network as a graph where links have attributes corresponding to the time-stamps at which they exist in the network. Some use a representation – often simply called *temporal network* [18] – which aims at gathering the different existing models of dynamical interacting systems, without making a strict choice of formalism. In this paper, we use link streams [24], which associate a time interval to each interaction. It proposes a rigorous formalism and allows accounting for the temporal aspect of the data without any parameter or restricting choice of timescale. Yet, it is simple to go from one mode of representation to another.

Listing all maximal cliques in a graph is known to be NP-hard. However, this problem is of utmost importance to analyze real-world graphs because it reveals high-density subgraphs and is thus a keystone to understand their structure. For instance, clique mining is used to detect relevant dense subgraphs [16, 14], define communities [35, 3] or compress graphs [8]. Efficient listing algorithms have been designed for large graphs representing real-world interaction networks. In particular, the Bron-Kerbosch algorithm [7] is used on large and sparse instances, especially by using the pivot improvement, which allows to considerably reduce the search space. Currently, one of the most efficient versions of this algorithm is the pivot proposed by Tomita *et al.* [41] and its implementation by Eppstein *et al.* [12], which uses an adequate node ordering method and an efficient implementation for large sparse real-world graphs.

In link streams, cliques are sets of vertices interacting with all others during a time interval, and maximal cliques are maximal both in number of nodes and in duration. Their listing has attracted interest in the last few years because it brings a powerful analysis tool to the field. To the best of our knowledge, the first algorithm to achieve this task was proposed by Viard *et al.* [43] and later improved in [44]. In the meantime, Himmel *et al.* proposed a new version based on the adaptation of the Bron-Kerbosch algorithm to a dynamical context [17], which was later generalized to the notion of k-plex by Bentert *et al.* [5]. Depending on the experimental settings, one or the other method can be faster. However, these methods are still limited to relatively small dynamical networks, not allowing the enumeration on very large datasets. Therefore, there is a need for more efficient algorithms to list maximal cliques in link streams.

The contributions of this paper are the following:

- We propose a new algorithm for listing maximal cliques in link streams, which scales to massive real-world datasets,

- We analyze the complexity of this algorithm and provide two complexities: one that depends on the input and an output-sensitive complexity that is close to the output size, thereby showing that our algorithm is efficient. We also show that the memory requirements are close to optimal.

- We show experimentally that it significantly outperforms the state of the art and allows enumerating cliques in networks that are two orders of magnitude larger than what was previously feasible in the time and memory limits of the protocol, which allows us to step up from link streams with less than half a million links to link streams with more than 100 million links.

- Finally, we provide two implementations of the algorithm: one in `Python`, used for comparison to the state of the art and another in `C++`, which is the most efficient one currently available and also allows a parallel enumeration. The code is publicly available [1].

The rest of this paper is organized as follows. Section 2 gives the basic definitions and notations that we use throughout the paper. Section 3 presents work in the literature related to maximal clique enumeration in link streams. Then, our new algorithm is presented in Section 4. In Section 5, we analyze this algorithm, giving a proof of correctness as well as two theoretical complexities: one as a function of input parameters and one as a function of output parameters of the enumeration. Finally, in Section 6, we make an extensive experimental assessment of the performances of this algorithm, and briefly show the results of the parallel implementation.

[1] `https://gitlab.lip6.fr/baudin/maxcliques-linkstream`

2 Basic definitions and notations

2.1 Cliques in a graph

We briefly remind a few classic definitions in the context of graphs that are useful for this work. We only consider simple undirected graphs. Such a graph is defined as a pair $G = (V, E_G)$, where V is a set of vertices, and E_G a set of edges; the edges of E_G are of the form $\{x, y\}$, where $x, y \in V$ and $x \neq y$. For a given vertex $u \in V$, the neighborhood of u, denoted $\Gamma_G(u)$, is the set of vertices adjacent to u in G, that is to say $\Gamma_G(u) = \{v \in V \mid \{u, v\} \in E_G\}$.

A clique C of a graph $G = (V, E_G)$ is a set of vertices of V which are all connected to each other, or more formally

$$\forall u, v \in C \textit{ with } u \neq v, \{u, v\} \in E_G.$$

A clique is *maximal* if it is not included in any other. If C is a clique of a graph G, then the *neighborhood* of C in G, denoted $\mathcal{N}_C$, is the set of vertices which are neighbors of all vertices of C in G: $\mathcal{N}_C = \bigcap\limits_{v \in C} \Gamma_G(v)$.

2.2 Cliques in a link stream

We recall here a few definitions on link streams which are needed for the rest of this paper.

Definition 1 (Link stream)**.** *A link stream is a triplet $L = (T, V, E)$ where $T = [\alpha, \omega] \subset \mathbb{R}$ is a time interval, V a set of nodes and $E \subseteq T \times T \times V \times V$ a set of links such that for all links (b, e, u, v) in E, the beginning and ending times of the link b and e are such that $e \geq b$. We call $e - b$ the duration of the link.*

In the rest of this article, the link streams that we use are undirected, *i.e.* there is no distinction between $(b, e, u, v) \in E$ and $(b, e, v, u) \in E$. They are also simple, *i.e.* for all (b, e, u, v) in E, $u \neq v$ and for all (b, e, u, v) and (b', e', u, v) in E, $[b, e] \cap [b', e'] = \emptyset$.

A clique of a link stream $L = (T, V, E)$ is defined as follows:

Definition 2 (Clique of a link stream)**.** *A clique is a pair $(C, [t_0, t_1])$ where $t_0, t_1 \in T$ are respectively called the start and end times of the clique, and $C \subseteq V$ is the set of vertices in the clique, with $|C| \geq 2$. Each pair of vertices of C is connected by a link existing during the whole interval $[t_0, t_1]$.*

Formally, $(C, [t_0, t_1])$ is such that:

$$\forall u, v \in C \textit{ with } u \neq v,\ \exists (b, e, u, v) \in E \textit{ s.t. } [t_0, t_1] \subseteq [b, e].$$

Note that for the sake of simplicity, we use the term clique both to designate a clique C in a graph and a clique $(C, [t_0, t_1])$ in a link stream, while these objects are different in nature. This is because in general the context allows removing the ambiguity on the kind of object considered. As for a clique in a graph, a clique in a link stream can contain other cliques. During the exhaustive enumeration, we are only interested in those that are maximal. In a link stream, the notion of maximality applies both to time and to vertices. The following definitions formalize this maximality. In this paper, we are interested in cliques that are maximal both in terms of time and vertices. We start by defining these two notions that will be used in the description of our algorithm before defining maximal cliques formally.

Definition 3 (Time-maximal clique)**.** *A time-maximal clique $(C, [t_0, t_1])$ is a clique which cannot be extended in time: there is no clique $(C, [t'_0, t'_1])$ with $[t_0, t_1] \subsetneq [t'_0, t'_1]$.*

Definition 4 (Vertex-maximal clique)**.** *A vertex-maximal clique $(C, [t_0, t_1])$ is a clique which cannot be extended in vertices: there is no clique $(C', [t_0, t_1])$ with $C \subsetneq C'$.*

Definition 5 (Maximal Clique)**.** *A maximal clique $(C, [t_0, t_1])$ in a link stream is a clique which is time-maximal and vertex-maximal.*

To illustrate these definitions, the left panel of Figure 1 shows a link stream with its maximal cliques.

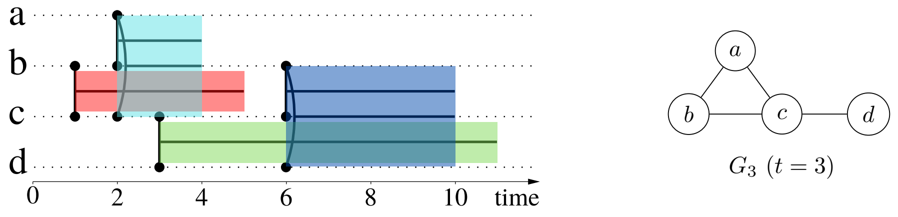


Figure 1: **Left:** a link stream with interaction time on the abscissa and vertices on the ordinate. For example, there is a link between b and c during the time interval $[1, 5]$. The maximal cliques are represented in color, *e.g.*, on the interval $[2, 4]$, the three vertices a, b, c are linked together, and form a maximal clique $(\{a, b, c\}, [2, 4])$. **Right:** the instantaneous graph G_3 of this link stream at $t = 3$.

2.3 Instantaneous graph of a link stream

We now give a few definitions that characterize a link stream $L = (T, V, E)$ at a given time $t \in T$. The edges existing at t can be seen as the edges of a graph that we call the instantaneous graph of L at time t. This graph contains the vertices in V and the edges existing at time t, or more formally:

Definition 6 (Instantaneous graph G_t associated to a link stream at time t)**.** *Given a time $t \in T$, the instantaneous graph G_t at time t associated to the link stream L is the graph $G_t = (V, E_{G_t})$ such that*

$$E_{G_t} = \{\{u, v\} \mid \exists (b, e, u, v) \in E,\ \ t \in [b, e]\}.$$

To avoid any confusion, we use the term *edge* for the elements of E_G in a graph $G = (V, E_G)$ and *link* for the elements of E in a link stream $L = (T, V, E)$. The edges of G_t are induced by the links of L, the corresponding link thus has an end time in T. We formalize below the notion of end time associated with an edge of G_t and the final time of a clique of G_t that follows.

Definition 7 (End time $\mathcal{E}_t(u, v)$ of an edge $\{u, v\}$ of G_t)**.** *Let $\{u, v\}$ be an edge of G_t. By definition of G_t, there is a unique link $(b, e, u, v) \in E$ such that $t \in [b, e]$. We call e the end time of the edge $\{u, v\}$ in G_t and denote it $\mathcal{E}_t(u, v)$.*

Definition 8 (Final time $\mathcal{E}_t(C)$ of a clique C of G_t)**.** *Let C be a clique of G_t. The final time $\mathcal{E}_t(C)$ of clique C is the minimum of the end times of the edges of C in G_t. Formally:*

$$\mathcal{E}_t(C) = \min_{u,v \in C} \{\mathcal{E}_t(u, v)\}.$$

For instance, on the right panel of Figure 1, the instantaneous graph at time $t = 3$ is the graph $G_3 = (\{a, b, c, d\}, E_3)$, with $E_3 = \{\{a, b\}, \{a, c\}, \{b, c\}, \{c, d\}\}$. The end times of its edges are $\mathcal{E}_3(a, b) = \mathcal{E}_3(a, c) = 4$, $\mathcal{E}_3(b, c) = 5$ and $\mathcal{E}_3(c, d) = 11$. G_3 contains the clique $\{a, b, c\}$, which also exists in G_2 and in G_4, and whose final time is 4, corresponding to the minimum of the end times of the three edges.

3 Related work

3.1 Maximal cliques in graphs

We first describe the Bron-Kerbosch algorithm [7], denoted BK, which is widely used to detect maximal cliques in graphs representing real-world networks, and serves as a basis for the design of the algorithm that we propose in this work. It is formally described in Algorithm 1.

Algorithm 1: Bron-Kerbosch algorithm (without pivot) on a graph G

```
Input: Graph $G = (V, E)$
Output: All maximal cliques of $G$ without duplicates
1 BK($\emptyset$, $V$, $\emptyset$)
2 Function BK($R$, $P$, $X$):
3     if $P \cup X = \emptyset$ then
4         output $R$ maximal clique
5     for $u \in P$ do
6         BK($R \cup \{u\}$, $P \cap \Gamma_G(u)$, $X \cap \Gamma_G(u)$)
7         $P \leftarrow P \setminus \{u\}$
8         $X \leftarrow X \cup \{u\}$
```

Broadly speaking, it is a recursive backtracking algorithm. The central idea is that each call maintains a clique R and the neighbors of all the vertices of R, which are distributed in two sets: P and X. Each vertex of $P \cup X$ can potentially expand the clique R. P corresponds to the candidate nodes that are actually used to expand R, while X corresponds to the vertices that are forbidden for expanding R to avoid the enumeration of duplicate cliques. Note that a clique R is maximal if and only if there is no vertex which is a neighbor of all its vertices, that is to say if and only if $P \cup X = \emptyset$.

In terms of complexity, this version of the BK algorithm makes exactly one recursive call per clique of the graph, maximal or not. Note that there can be many such cliques, as a given maximal clique of size q contains 2^q sub-cliques. To reduce this search complexity, Bron and Kerbosch have introduced the idea of selecting a pivoting vertex to prune the recursive call tree. Indeed, by choosing a vertex $p \in P \cup X$, we do not need to make recursive calls for the vertices in $\Gamma_G(p)$ because any maximal clique must include either p or a node which is not in $\Gamma_G(p)$. So we do not miss any maximal clique by cutting these branches of the tree. The corresponding modification is:

```
Replace Line 5 of Algorithm 1 by:
    Choose pivot $p \in P \cup X$
    for $u \in P \setminus \Gamma_G(p)$ do
```

The strategies to select the pivot and achieve an efficient pruning have been discussed in various works, *e.g.* [22]. In particular, the one proposed by Tomita *et al.* [41] maximizes $|P \cap \Gamma_G(p)|$ over all $p \in P \cup X$ and this guarantees that the worst-case time complexity is in $\emptyset 3^{n/3}$ with $n = |V|$. Eppstein *et al.* [12] use this strategy and also propose to make the first call of the BK function on each vertex according to the degeneracy ordering. The underlying idea is that when the vertices are processed in a random order, $|P|$ can be as large as the maximum degree, while when using the degeneracy ordering, $|P|$ cannot be larger than the graph degeneracy δ, which is smaller than the maximum degree and usually low in graphs representing real-world networks. They prove that their method ensures a time complexity in $\O \delta n 3^{\delta/3}$. Combined with adequate data structures, their implementation is currently considered to be one of the most efficient versions available of the BK algorithm.

3.2 Maximal cliques in link streams

The question of listing maximal cliques in dynamical networks has emerged relatively recently as an important research question to describe the structure of interaction data with temporal information. This problem has been considered in different ways: some see a dynamical network as a sequence of graph snapshots, which leads to tackle the problem as the evolution of cliques in graphs through time. This is for instance the point of view adopted in [40, 11]. Other works model the dynamical network as a link stream, thus acknowledging the intrinsically temporal aspect of the clique itself and thus look at it as an object which should be redefined in the dynamical context [43, 17, 44, 5]. We focus on this second point of view.

In this paper, we consider works that define cliques in link streams. This issue has been addressed with slightly different formalisms: some consider the link streams formalism [43, 44] and others use the temporal networks one [17, 5]. All these works nonetheless consider cliques as a set of vertices interacting during a given time interval, as discussed in Section 2.2. Both representations contain the same information, and it is simple to go from one representation to the other, making it possible to compare the efficiency of the different methods.

Viard *et al.* [43] proposed the first algorithm to list maximal cliques in link streams. In that work, links do not have duration, but the cliques are themselves parameterized with a value Δ, and thus called Δ-cliques. In this work Δ-cliques are defined such that all pairs of nodes in the Δ-clique interact at least once during each sub-interval of duration Δ. Their algorithm has since then been both simplified and extended to the more general formalism of link streams that allow link duration [44]. The idea underlying this algorithm is to extend either in terms of time or nodes a clique in a link stream, starting from a specific link of the stream. While this method indeed allows finding all maximal cliques in the link stream, it relies on memoization in order to decide whether a given clique has already been processed and this induces a high memory usage, which is prohibitive in many cases.

In parallel, Himmel *et al.* [17] proposed another algorithm to enumerate Δ-cliques in link streams (named temporal graphs in these papers). It adapts the BK algorithm to this dynamical setting, and implements different pivoting strategies. This version regularly outperforms the algorithm in [43], especially on larger Δ values. The results are more contrasted when compared to the algorithm in [44]: depending on the experimental settings and the data under study, one or the other method may be more efficient. More recently, Bentert *et al.* [5] have developed a generalization of this algorithm to list all maximal Δ-k-plexes in link streams. A Δ-k-plex in a link stream is defined by analogy with k-plexes in a graph, which is a maximal subgraph of size s such that any vertex in the k-plex is adjacent to at least $s-k$ vertices in the subgraph. So a Δ-k-plex in a link stream is a subset of s nodes and a time interval Δ in which each vertex interacts with at least $s-k$ vertices of the Δ-k-plex during each sub-interval of duration Δ. Note that a Δ-1-plex ($k=1$) is equivalent to a Δ-clique, which allows comparing this algorithm to the others described above. Here also, the algorithm is based on the structure of the BK algorithm, but it contains a few implementation improvements on [17], which makes it more efficient in terms of practical computation times. These two methods obtain running time bounds depending on a temporal variant of the degeneracy of the input graph, similarly to Eppstein *et al.* [12] in the static case.

Banerjee and Pal have proposed the notion of maximal (Δ, γ)-cliques [2], which is a generalization of maximal Δ-clique: a (Δ, γ)-clique is defined in the same way as a Δ-clique, but each link must appear at least γ times in each sub-interval of size Δ. Note that for $\gamma = 1$, their definition is equivalent to maximal Δ-cliques. Molter *et al.* also proposed a generalization of maximal clique enumeration in link streams [33], by enumerating *isolated* maximal cliques, that are cliques with few edges connected to the rest of the temporal network. The isolation is quantified by a factor c that is the maximal number of links between the clique and the rest of the network. We can recover the definition of maximal clique with an isolation condition that makes all cliques isolated (*e.g.* take $c = n^2$).

Finally, some methods which were originally designed to update cliques in graphs that evolve with time may be exploited in the context of clique enumeration in link streams. In particular, Das *et al.* [11] apply the Tomita *et al.* [41] clique enumeration method in this context. This is done by enumerating the cliques that contain at least one edge that is new at the current time. While the formal problem is different from ours, this method can be directly adapted to our problem, as we will see in Section 4.1.

4 Algorithm

The algorithm that we propose is based on an adaptation of the BK algorithm to a dynamical setting, following the same logic as the one proposed in Himmel *et al.* [17]. We use the BK procedure to enumerate exactly once all sets of vertices C that form a time-maximal clique (C, I), and then filter these time-maximal cliques by keeping only those that are vertex-maximal. The main difference of our approach is that we work on neighborhoods which are limited to the interactions actually occurring at time t and not to the complete link stream.

In what follows, we consider a simple link stream $L = (T, V, E)$. We describe the general structure of our algorithm, then we detail the two main points: first, enumerating the time-maximal cliques via the enumeration of cliques of instantaneous graphs G_t, second, testing their vertex-maximality to filter out those which are not. Finally, we improve the computation times by pruning the search tree with a *pivot* that is similar to the one used in [17].

4.1 General structure of the algorithm

The algorithm that we propose first enumerates exactly once each time-maximal clique and then filters the ones which are vertex-maximal. For the first step, we use the equivalence given by Lemma 1.

Lemma 1 (Time-maximality of a clique)**.** *$(C, [t_0, t_1])$ is a time-maximal clique if and only if:*

(i) C is a clique of G_{t_0};

(ii) There is an edge $\{u, v\}$ in G_{t_0} with $u, v \in C$ that originates from a link $(t_0, e, u, v) \in E$ that begins at t_0;

(iii) $t_1 = \mathcal{E}_{t_0}(C)$.

Proof. We suppose that $(C, [t_0, t_1])$ is a time-maximal clique. Then C is trivially a clique of G_{t_0} as all pairs of nodes of C are connected at time t_0. Moreover, Viard *et al.* [44] proved in their Lemma 3 that there must exist an edge $\{u, v\}$ with $u, v \in C$ that originates from a link $(t_0, e, u, v) \in E$ that begins at t_0. Indeed, if all the links in the clique begin before t_0, its time interval can be extended to the left, and it is therefore not time-maximal. Finally, as the clique $(C, [t_0, t_1])$ cannot be extended in time, it implies that the final time associated with the clique must be the maximum time when all links are present in the link stream, in other words $\mathcal{E}_{t_0}(C) = t_1$.

Reciprocally, if C is a clique of G_{t_0} and $t_1 = \mathcal{E}_{t_0}(C)$, it means that all edges in C originates from links of the link stream that are present over the whole interval $[t_0, t_1]$, so $(C, [t_0, t_1])$ is a clique of L. Moreover, the time interval of this clique is maximal, as (1) at least one link is no longer present after t_1; (2) if there is an edge $\{u, v\}$ with $u, v \in C$ that originates from a link $(t_0, e, u, v) \in E$ beginning at t_0, it means that at least one edge is not present before t_0. So the clique $(C, [t_0, t_1])$ is time-maximal. □

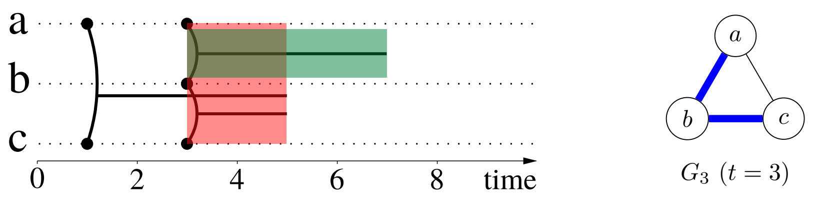


Figure 2: A link stream with the two maximal cliques that start at $t = 3$ in color, and its instantaneous graph G_3 with the edges that start at $t = 3$ in blue. There are three time-maximal cliques that start at this instant, and each contains a new edge (in blue): $(\{a, b\}, [3, 7])$, $(\{b, c\}, [3, 5])$ and $(\{a, b, c\}, [3, 5])$. The clique $(\{b, c\}, [3, 5])$ is not vertex-maximal, while the two others are.

We deduce from Lemma 1 that we can enumerate the time-maximal cliques by going through T: for each instant $t \in T$, we enumerate the time-maximal cliques that start at time t. To do so, we enumerate each (graph) clique C of G_t that contains an edge $\{u, v\}$ of a link $(t, e, u, v) \in E$ starting at t. Each of these graph cliques is then associated to the time-maximal clique $(C, [t, \mathcal{E}_t(C)])$ of the link stream. The lemma ensures that we list all time-maximal cliques in this way. Then, we only have to find those that are also vertex-maximal. Figure 2 gives an illustration of this procedure at time $t = 3$: the time-maximal cliques that begin at $t = 3$ are $(\{a, b\}, [3, 7])$, $(\{b, c\}, [3, 5])$ and $(\{a, b, c\}, [3, 5])$, and they all contain at least one new link and their end time corresponds to the minimum of the end times of their links. Then, only $(\{a, b\}, [3, 7])$ and $(\{a, b, c\}, [3, 5])$ are output, since $(\{b, c\}, [3, 5])$ is not vertex-maximal. This framework is summarized in Figure 3, and is implemented in Algorithm 2.

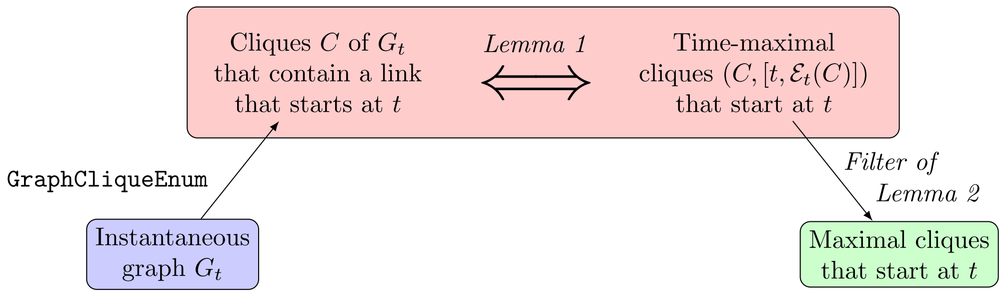

Cliques C of G_t that contain a link that starts at t
Lemma 1
$\Longleftrightarrow$
Time-maximal cliques $(C, [t, \mathcal{E}_t(C)])$ that start at t
GraphCliqueEnum
Filter of Lemma 2
Instantaneous graph G_t
Maximal cliques that start at t

Figure 3: General structure of Algorithm 2: for each time $t \in T$, it enumerates the maximal cliques that start at t.

We first describe the high-level features of Algorithm 2. The details will be discussed in the rest of this section. For each time $t \in T$ (Line 1), the algorithm follows two steps:

Algorithm 2: Algorithm of maximal clique enumeration in link streams.

```
Input: A link stream $L = (T, V, E)$
Output: All maximal cliques of $L$ without duplicates
1  for $t \in T$ do
2      $Cliques \leftarrow \emptyset$
3      $ForbidEdges \leftarrow \emptyset$
4      for $(t, \_, u, v) \in E$ do                    // Loop on links starting at time $t$
           // Enumerating cliques in $G_t$ containing $\{u,v\}$:
5          $Cliques \leftarrow Cliques \cup$ GraphCliqueEnum($\{u, v\}$, $\Gamma_{G_t}(u) \cap \Gamma_{G_t}(v)$, $\emptyset$,
             $ForbidEdges$, $t$)
6          $ForbidEdges \leftarrow ForbidEdges \cup \{\{u, v\}\}$
7      for $(C, \mathcal{N}_C) \in Cliques$ do                // $\mathcal{N}_C$ = neighborhood of $C$
8          if $\forall u \in \mathcal{N}_C,\ \ \mathcal{E}_t(C \cup \{u\}) < \mathcal{E}_t(C)$ then      // Vertex maximality test
9              output $(C, [t, \mathcal{E}_t(C)])$

10 Function GraphCliqueEnum($R$, $P$, $X$, $ForbidEdges$, $t$):
11     output $(R, P \cup X)$                          // $P \cup X$ = neighborhood of $R$
12     $Q \leftarrow \{u \in P \mid \exists \{u, v\} \in ForbidEdges,\ v \in R\}$
13     for $u \in P \setminus Q$ do
14         GraphCliqueEnum($R \cup \{u\}$, $P \cap \Gamma_{G_t}(u)$, $X \cap \Gamma_{G_t}(u)$, $ForbidEdges$, $t$)
15         $P \leftarrow P \setminus \{u\}$
16         $X \leftarrow X \cup \{u\}$
```

- Lines 2 to 6 enumerate exactly once each time-maximal clique beginning at t. To do this, the cliques of G_t that contain an edge from a link starting at t are enumerated by processing every link $(t, \_, u, v) \in E$ that starts at t (Line 4). This enumeration is made by a call to the function `GraphCliqueEnum` (Line 5). Note the use of the set of edges $ForbidEdges$ that allows to avoid listing a same clique several times, which will be discussed in further details in Section 4.2.

- Lines 7 to 9 test whether the time-maximal cliques enumerated above are vertex-maximal or not. To do so, we use the neighborhood $\mathcal{N}_C$ of C for each time-maximal clique $(C, [t_0, t_1])$. The maximality test (Line 8) will be discussed in Section 4.3.

Note also that the iterations of the loop at Line 1 of the algorithm are independent of each other and can therefore be computed in parallel. We describe this parallelization process and its results in Section 6.

4.2 Clique enumeration in graphs G_t

In this section, we discuss the procedure `GraphCliqueEnum`($R, P, X, ForbidEdges, t$) which enumerates all graph cliques C of G_t satisfying $R \subseteq C \subseteq R \cup P$ and containing no edge of $ForbidEdges$. It is called on Line 5 of Algorithm 2 and detailed from Line 10 to 16. This procedure is a variant of the BK algorithm, described in Section 3.1: it is a backtracking recursive function which explores the neighborhood of the clique under construction to extend it. Here, the clique under construction is R, its neighborhood is $P \cup X$, and the candidates for increasing R without enumerating duplicates are the vertices of P. The fact that $P \cup X$ is the neighborhood of clique R is formally proved in Section 5.1 by Lemma 3. There are three major differences with the BK procedure described in Algorithm 1 that should be noted:

- each clique R is output with its neighborhood $P \cup X$ (Line 11) because it is necessary for the vertex-maximality test of Line 8, as discussed in Section 4.3;
- a clique is output whether it is maximal in G_t or not (Line 11): there is no test equivalent to Line 3 of Algorithm 1;
- the output cliques must not contain any edge of *ForbidEdges*; this is ensured by Line 12 which prevents to add a vertex u to R if it implies the addition of an edge $\{u, v\}$ of *ForbidEdges* to the resulting clique $R \cup \{u\}$.

Using the set *ForbidEdges* guarantees to list each clique at most once, as will be proved in Section 5. Indeed, if a clique C contains two edges $\{u_1, v_1\}$ and $\{u_2, v_2\}$, then it is in the set of cliques of G_t which contain $\{u_1, v_1\}$, but also in the set of cliques which contain $\{u_2, v_2\}$. Thus, if `GraphCliqueEnum` is called at t on $\{u_1, v_1\}$ and $\{u_2, v_2\}$ without *ForbidEdges*, C would be enumerated twice. This idea is illustrated by the example in Figure 4: `GraphCliqueEnum` is called first on $\{a, c\}$ and output cliques $\{a, c\}$, $\{a, b, c\}$ $\{a, c, d\}$ and $\{a, b, c, d\}$. Then $\{a, c\}$ is added to *ForbidEdges* and `GraphCliqueEnum` is called on $\{b, d\}$, outputting cliques $\{b, d\}$, $\{a, b, d\}$ and $\{b, c, d\}$ but not $\{a, b, c, d\}$ because it contains the edge $\{a, c\}$ which is in *ForbidEdges*. Thus, all cliques containing at least one red edge are listed exactly once.

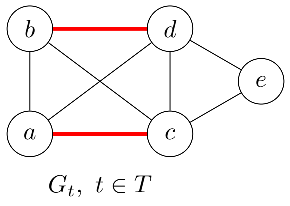


Figure 4: Example of a clique enumeration using *ForbidEdges*. An instantaneous graph G_t of a link stream L at time t is represented. The thick red edges correspond to a link that begins at t, while the others correspond to links that have begun earlier. The clique $\{a, b, c, d\}$ contains the two new edges $\{a, c\}$ and $\{b, d\}$, which is why there is a need for the set *ForbidEdges* to avoid enumerating it twice.

4.3 Vertex maximality test of time-maximal cliques

In this section, we explain the vertex maximality test used in Line 8 of Algorithm 2. The following lemma allows filtering the vertex-maximal cliques among the time-maximal cliques.

Lemma 2 (Vertex-maximality of a time-maximal clique). *Let $(C, [t, \mathcal{E}_t])$ be a time-maximal clique and $\mathcal{N}_C$ the neighborhood of clique C in G_t (*i.e. $\mathcal{N}_C = \bigcap_{v \in C} \Gamma_{G_t}(v)$*). Then*

$$(C, [t, \mathcal{E}_t(C)]) \text{ is vertex-maximal } \Leftrightarrow \forall u \in \mathcal{N}_C,\quad \mathcal{E}_t(C \cup \{u\}) < \mathcal{E}_t(C).$$

In other words, a time-maximal clique $(C, [t, \mathcal{E}_t(C)])$ is vertex-maximal (and thus maximal) if and only if we cannot add any node to C without reducing its final time. Note that if the neighborhood of C is empty, *i.e.* $\mathcal{N}_C = \emptyset$, then $(C, [t, \mathcal{E}_t(C)])$ is always vertex-maximal.

Proof. Let $(C, [t, \mathcal{E}_t(C)])$ be a maximal clique and suppose that there exists some $u \in \bigcap_{v \in C} \Gamma_{G_t}(v)$ such that $\mathcal{E}_t(C \cup \{u\}) \geq \mathcal{E}_t(C)$. Then, by definition of $\mathcal{E}_t(C \cup \{u\})$, for any pair of nodes $x, y \in C \cup \{u\}$, $x \neq y$, there must be a link $(b, e, x, y) \in E$ such that $[t, \mathcal{E}_t(C)] \subseteq [b, e]$. In consequence, $(C \cup \{u\}, [t, \mathcal{E}_t(C)])$ is a clique of L, which contradicts our hypothesis that $(C, [t, \mathcal{E}_t(C)])$ is maximal. This proves that a time-maximal clique $(C, [t, \mathcal{E}_t(C)])$ is vertex-maximal implies that $\forall u \in \bigcap_{v \in C} \Gamma_{G_t}(v)$, $\mathcal{E}_t(C \cup \{u\}) < \mathcal{E}_t(C)$.

Reciprocally, suppose that $(C, [t, \mathcal{E}_t(C)])$ is not vertex-maximal, then by definition there must be a vertex $u \in \bigcap_{v \in C} \Gamma_{G_t}(v)$ such that $(C \cup \{u\}, [t, \mathcal{E}_t(C)])$ is a clique of L. This implies in turn that $\mathcal{E}_t(C) \leq \mathcal{E}_t(C \cup \{u\})$. So, if $\forall u \in \bigcap_{v \in C} \Gamma_{G_t}(v)$, $\mathcal{E}_t(C \cup \{u\}) < \mathcal{E}_t(C)$ then $(C, [t, \mathcal{E}_t(C)])$ is vertex-maximal. □

Therefore, the filtering test performed at Line 8 does correspond to the maximality test of the above lemma. For example, in Figure 2, the time-maximal clique $(\{b, c\}, [3, 5])$ is not vertex-maximal, because we can add vertex a to it without reducing its final time. This is not the case for $(\{a, b\}, [3, 7])$ and $(\{a, b, c\}, [3, 5])$ which are both time-maximal and vertex-maximal, and thus maximal.

4.4 Pivoting strategy to improve clique enumeration in G_t graphs

As reported in Section 3.1, the BK algorithm in graphs is usually implemented with a pivot to prune the recursion tree. A similar process can be implemented in the dynamical context to eliminate some recursive calls when enumerating time-maximal cliques, as proposed in [17]. We adapt this improvement to the function `GraphCliqueEnum`, which corresponds to the following modification of Algorithm 2:

```
Replace Line 13 of Algorithm 2 by:
    Choose pivot $p \in P \cup X$
    $Del \leftarrow \{u \in P \cap \Gamma_{G_t}(p) \mid \mathcal{E}_t(R \cup \{u\}) = \mathcal{E}_t(R \cup \{u, p\})\}$
    for $u \in (P \setminus Del) \setminus Q$ do
```

According to this procedure, vertices in the set *Del* are not considered in the recursive calls. Indeed, not making recursive calls for these vertices does not remove any maximal clique from the enumeration. This statement is formally proved by Theorem 2, in Section 5.1. This is justified by the observation that at a given time t, for any maximal clique $(C, [t, \mathcal{E}_t(C)])$ such that $R \subseteq C \subseteq R \cup P$, and given a pivot $p \in P \cup X$, there is always a vertex $u \in P \setminus Del$ which allows extending R towards C. To observe that, there are two cases:

- if there exists a vertex $u \in C$ that is not a neighbor of p, then $u \notin Del$ since $Del \subseteq \Gamma_{G_t}(p)$, and $u \in P$ since $R \subseteq \Gamma_{G_t}(p)$. Then, a recursive call with u extends R towards C – note that if $p \in C$ then $u = p$ as $p \notin \Gamma_{G_t}(p)$,

- else, all the vertices in C are neighbors of p, which means on the one hand that $p \notin C$, and on the other hand that $C \cup \{p\}$ is a clique of G_t. Since $(C, [t, \mathcal{E}_t(C)])$ is maximal, this implies that $\mathcal{E}_t(C \cup \{p\}) < \mathcal{E}_t(C)$. Therefore there must exist a vertex $u \in P \cap C$ such that $\mathcal{E}_t(R \cup \{u\}) > \mathcal{E}_t(R \cup \{u, p\})\}$, *i.e.* u does not belong to *Del* and allows extending R towards C.

Besides, we mentioned in Section 3.1 that the Tomita *et al.* pivoting strategy for clique enumeration in graphs [41] chooses a pivot that maximizes the set of vertices to be removed from the candidates. Similarly, we select a pivot that allows to cut as many calls as possible: for each potential $p \in P \cup X$, we choose the one that maximizes $|Del|$. It has been shown experimentally in [17] that this choice does indeed achieve the most efficient pruning. We evaluate in Section 6.4 the pruning efficiency by comparing the calculations made with and without a pivot.

5 Analysis

In this section, we show that Algorithm 2 is correct: it allows enumerating once and only once each maximal clique of a link stream. Then, we study its complexity, first as a function of the global parameters of the link stream, then as a function of the output of the algorithm.

5.1 Correctness

To show that Algorithm 2 is correct, we need the following preliminary lemmas: Lemma 3 which characterizes the output of the calls to `GraphCliqueEnum` in the algorithm, and Lemma 4 which shows that all the maximal cliques starting at time t are enumerated by the procedure from Lines 2 to 6. Finally, the correctness of the algorithm is proved in Theorem 1.

Lemma 3. *In Algorithm 2, each call to* `GraphCliqueEnum(`$R,P,X,ForbidEdges,t$`)` *satisfies:*

- *R is a clique of G_t;*
- *$P \cup X = \bigcap\limits_{v \in R} \Gamma_{G_t}(v)$.*

Proof. `GraphCliqueEnum` being a recursive function, we prove this lemma by induction on the tree of the recursive calls made by Algorithm 2.

Initialization. The first call of a series of recursive calls is made at Line 5 of Algorithm 2, and is of the form `GraphCliqueEnum(`$\{u,v\}, \Gamma_{G_t}(u) \cap \Gamma_{G_t}(v), \emptyset, ForbidEdges, t$`)`, for some $u, v \in V$. At this point, $R = \{u,v\}$, and u and v are connected in G_t because they come from the link $(t, \_, u, v)$ of E (Line 4). So R is indeed a clique of G_t. Besides, $P = \Gamma_{G_t}(u) \cap \Gamma_{G_t}(v)$ and $X = \emptyset$, so $P \cup X = \bigcap\limits_{v \in R} \Gamma_{G_t}(v)$.

Induction. Suppose `GraphCliqueEnum(`$R, P, X, ForbidEdges, t$`)` satisfies that R is a clique of G_t and $P \cup X = \bigcap\limits_{v \in R} \Gamma_{G_t}(v)$. Let us show that the recursive calls made at Line 14 induced by this call maintain these properties. On the one hand, $P \cup X = \bigcap\limits_{v \in R} \Gamma_{G_t}(v)$ is an invariant of the loop starting at Line 13: indeed, after the operations of Lines 15 and 16, $P \leftarrow P \backslash \{u\}$ and $X \leftarrow X \cup \{u\}$, so $P \cup X$ is not modified. On the other hand, at a given loop iteration associated with vertex u, the recursive call is made on $R' = R \cup \{u\}$, $P' = P \cap \Gamma_{G_t}(u)$ and $X' = X \cap \Gamma_{G_t}(u)$. First, $u \in P$, so by the induction hypothesis, u is neighbor of all vertices in R, thus R' is a clique of G_t. Second, $\bigcap\limits_{v \in R'} \Gamma_{G_t}(v) = \bigcap\limits_{v \in R} \Gamma_{G_t}(v) \cap \Gamma_{G_t}(u) = (P \cup X) \cap \Gamma_{G_t}(u) = (P \cap \Gamma_{G_t}(u)) \cup (X \cap \Gamma_{G_t}(u))$, so $\bigcap\limits_{v \in R'} \Gamma_{G_t}(v) = P' \cup X'$. Thus, the properties are indeed true at each recursive call.

□

Lemma 4. *For $t \in T$, the set Cliques at Line 7 of Algorithm 2 contains exactly one pair $(C, \mathcal{N}_C)$ per time-maximal clique $(C, [t, \mathcal{E}_t(C)])$ of L that begins at t.*

Proof. Let $(C, [t, \mathcal{E}_t(C)])$ be a time-maximal clique of L, that begins at t. We show by induction on $2 \leq k \leq |C|$ the following property $\mathcal{P}(k)$: *"There is a call* `GraphCliqueEnum(`*R,P,X,ForbidEdges,t*`)` *within Algorithm 2 such that R satisfies $|R| = k$, $R \subseteq C \subseteq R \cup P$ and ForbidEdges contains no edge of C".*

Initialization. From Lemma 1 *(ii)*, there exists at least one link (t, e, u, v) in the link stream that begins at t with $u, v \in C$. Consider the first such link processed by the loop starting at Line 4. Let us then consider the call to `GraphCliqueEnum` of this iteration, at Line 5. In this call, *ForbidEdges* does not yet contain any edge of C. Moreover, $R = \{u, v\}$ and $P = \Gamma_{G_t}(u) \cap \Gamma_{G_t}(v)$, and as all the elements of C must be neighbors of u and v, we have $R \subseteq C \subseteq R \cup P$. Thus, $\mathcal{P}(2)$ is true.

Induction. Let k be such that $2 \leq k \leq |C| - 1$, we assume $\mathcal{P}(k)$ to be true, and show $\mathcal{P}(k+1)$. Let `GraphCliqueEnum(`*R,P,X,ForbidEdges,t*`)` be a call of Algorithm 2 corresponding to $\mathcal{P}(k)$. From $\mathcal{P}(k)$, we know that *ForbidEdges* contains no edge of C, therefore $C \cap Q = \emptyset$ (Line 12) and the vertices of $C \setminus R$ (which is not empty) that may extend R are in $P \setminus Q$. Without loss of generality, we consider the first such vertex $u \in C \setminus R$ processed by the loop starting at Line 13. In the recursive call `GraphCliqueEnum(`$R \cup \{u\}, P \cap \Gamma_{G_t}(u), X \cap \Gamma_{G_t}(u), ForbidEdges, t$`)`, *ForbidEdges* is unchanged, so it contains no edge of C; moreover, by construction we have $R \cup \{u\} \subseteq C \subseteq P \cap \Gamma_{G_t}(u)$, and $|R \cup \{u\}| = k + 1$. Therefore, $\mathcal{P}(k+1)$ is true.

When $k = |C|$, we have shown that there is a call in Algorithm 2 that outputs $(C, \mathcal{N}_C)$ at time t. We prove now that there cannot exist another call that outputs $(C, \mathcal{N}_C)$ at time t. At Line 6, $\{u, v\}$ is added to *ForbidEdges* after the first call involving (t, e, u, v) with $u, v \in C$. Then, as Q (Line 12) prevents the clique under construction from containing an edge of *ForbidEdges*, no call can enumerate C in the following iterations. Finally, if we consider this iteration, the call to `GraphCliqueEnum` cannot enumerate $(C, \mathcal{N}_C)$ twice, because each recursive call in the loop starting at Line 13 enumerates different cliques, as u is deleted from P at Line 15. ☐

Theorem 1 (Correctness). *Algorithm 2 lists once and only once each maximal clique of the input link stream L and nothing else.*

Proof. We prove that for any $t \in T$, the related iteration of the loop starting at Line 1 enumerates all the maximal cliques that start at t, without duplication. Indeed, according to Lemma 4, the loop starting at Line 7 iterates over all pairs $(C, \mathcal{N}_C)$ such that $(C, [t, \mathcal{E}_t(C)])$ is a time-maximal clique starting at t without duplication. Moreover, according to Lemma 3, $\mathcal{N}_C = P \cup X = \bigcap_{v \in C} \Gamma_{G_t}(v)$ so, according to the vertex-maximality test of Lemma 2, the time-maximal cliques output after the test at Line 8 are all of those which are vertex-maximal. Thus, at iteration t, the algorithm outputs once all maximal cliques that start at t and nothing else. ☐

We proposed in Section 4.4 to introduce a pivot in order to prune the recursive call tree of `GraphCliqueEnum`. By cutting off branches corresponding to redundant computations, it improves the practical running times without changing the output of the algorithm. We now prove that the algorithm with a pivot is correct.

Theorem 2 (Correct pivoting)**.** *Algorithm 2 with pivoting is correct: adding a pivot as described in Section 4.4 does not change the output of the algorithm.*

Proof. It is clear that adding a pivot to `GraphCliqueEnum` as described in Section 4.4 reduces the set of candidate nodes to add to the clique R under construction. Thus, the pivot can only reduce the set *Cliques* on Line 7. So we can assert that all cliques output by the algorithm with a pivot are still maximal cliques of L and that they are all distinct.

It remains to show that each maximal clique is effectively output by the algorithm with a pivot. Let $(C, [t, \mathcal{E}_t(C)])$ be a maximal clique. Following the same scheme as the one that we used for the proof of Lemma 4, we show by induction on $2 \leq k \leq |C|$ the property $\mathcal{P}(k)$: *"There is a call* `GraphCliqueEnum(`*R,P,X,ForbidEdges,t*`)` *within Algorithm 2 with a pivot, such that R satisfies $|R| = k$ with $R \subseteq C \subseteq R \cup P$ and ForbidEdges contains no edge of C."* Then, the call corresponding to $k = |C|$ would demonstrate the result. The initialization step is the same as in the proof of Lemma 4. Now assuming $\mathcal{P}(k)$ is true, given $k < |C|$ and `GraphCliqueEnum(`*R,P,X,ForbidEdges,t*`)` the corresponding call, we only need to show that there is a vertex of $C \setminus R$ to extend R into C which is in $(P \setminus Del) \setminus Q$. Let us suppose that it is not the case, then $((P \setminus Del) \setminus Q) \cap C = \emptyset$ and since *ForbidEdges* does not have any edge in C, $C \cap Q = \emptyset$, so we must have $C \subseteq R \cup Del$. Now, by definition of Del, we have $R \cup Del \subseteq \Gamma_{G_t}(p)$, so $C \subseteq \Gamma_{G_t}(p)$. This implies two things: $p \notin C$, and $C \cup \{p\}$ is a clique of G_t. Moreover, $C \subseteq R \cup Del$ implies that $\mathcal{E}_t(C) = \mathcal{E}_t(C \cup \{p\})$ which contradicts the fact that $(C, [t, \mathcal{E}_t(C)])$ is a maximal clique. Hence, $\mathcal{P}(k+1)$ is true, and we deduce the expected result. □

5.2 Complexity

We compute here the complexity as a function of the input link stream parameters, and then as a function of the features of the output. Throughout this section, we refer to the following characteristics of the link stream:

- $|T|$: number of different time instants at which links begin or end, *i.e.*:

$$|T| = |\{t \in T \mid \exists (b, e, u, v) \in E,\ t = b \text{ or } t = e\}|$$

- $m = |E|$: the number of links; note that each link (b, e, u, v) in the link stream corresponds to two times $b, e \in T$, therefore $|T| \leq 2m$;

- d: the maximal degree of a vertex in any static graph G_t;

- α_T: the number of time-maximal cliques in L; notice that not all time-maximal cliques are maximal and that each link induces a time-maximal clique, therefore $m \leq \alpha_T$;

- α: the number of maximal cliques (*i.e.* both time-maximal and vertex-maximal) in L; note that $\alpha \leq \alpha_T$;

- q: the maximal number of vertices in a clique.

5.2.1 Complexity as a function of the input link stream parameters

To compute the time complexity of Algorithm 2, we need the preliminary Lemma 5, which expresses it as a function of the number of time-maximal cliques α_T. This complexity implicitly takes into account the factors $|T|$ and m of the link stream, since $\frac{1}{2}|T| \leq m \leq \alpha_T$. Then, we will estimate an upper bound on α_T to deduce a general expression of this complexity.

Lemma 5. *The time complexity of Algorithm 2 is in $\emptyset d^2 \cdot \alpha_T$.*

Proof. We recall that, by Lemma 4, for each $t \in T$, there is exactly one pair $(C, \mathcal{N}_C)$ enumerated in the set *Cliques* for each time-maximal clique of the link stream that begins at t. Thus, there are a total of α_T couples $(C, \mathcal{N}_C)$ enumerated by the whole global loop of Line 1.

First, let us analyze the complexity of a recursive call to `GraphCliqueEnum`. Let us consider each operation associated to such a recursive call and show that they never exceed $\emptyset d^2$:

- computing its arguments (at Line 5 or 14) is done by intersection of sets of size at most d, so it is in $\emptyset d$;
- the sets in the couple $(R, P \cup X)$ output at Line 11 are of size at most d, so the output is in $\emptyset d$;
- computation of Q at Line 12 corresponds, for each element u of P, to the intersection between R and the neighbors of u in *ForbidEdges*. Each of the three sets mentioned are of size at most d, so this calculation is done in $\emptyset d^2$;
- each iteration of the loop starting on Line 13 is constituted of the computation of the arguments of another recursive call (Line 14) counted in this one's complexity, and constant time operations (Lines 15 and 16).

Finally, each pair $(C, \mathcal{N}_C)$ is the output of a recursive call to `GraphCliqueEnum`, and each recursive call outputs one such pair. Since there are α_T recursive calls following the initial remark, the complexity of the corresponding operations is therefore in $\emptyset d^2 \cdot \alpha_t$.

Second, the processing of each pair $(C, \mathcal{N}_C)$ by the maximality test of Lines 8 and 9 is done in $\emptyset d^2$ and thus the complexity of all maximality tests done by the entire algorithm is in $\emptyset d^2 \cdot \alpha_T$. Indeed, each test requires computing the final times $\mathcal{E}_t(C)$, and $\mathcal{E}_t(C \cup \{u\})$ for u in $\mathcal{N}_C$. Notice that the link stream data structure allows direct access to the end time of each link, so during successive calls to `GraphCliqueEnum`, it is possible to maintain the end time of the current clique R in $\emptyset d$. This is because when we add a vertex u to R at Line 14 of Algorithm 2, then $\mathcal{E}_t(R \cup \{u\}) = \min\{\mathcal{E}_t(R), \min_{v \in R}\{\mathcal{E}_t(u, v)\}\}$, therefore this operation can be carried out in $\emptyset |R| \subseteq \emptyset d$. Then, access to $\mathcal{E}_t(C)$ is possible in $\emptyset d$ operations. Now, $\mathcal{E}_t(C \cup \{u\})$ is obtained by adding one vertex to C, so with the same procedure in $\emptyset |C| \subseteq \emptyset d$. Since $\mathcal{N}_C$ is at most of size d, it is computed in $\emptyset d$. Moreover, the output of Line 9 has also a complexity in $\emptyset d$. Thus, each pair $(C, \mathcal{N}_C)$ is processed in $\emptyset d^2$, so the total complexity of the maximality tests made by the entire algorithm is in $\emptyset d^2 \cdot \alpha_T$.

Finally, the overall complexity, which is the sum of the two above, is in $\emptyset d^2 \cdot \alpha_T$. $\square$

We can now show the overall time complexity of our algorithm.

Theorem 3 (Time complexity). *The time complexity of Algorithm 2 is in* $\emptyset m \cdot 3^{d/3} \cdot 2^q \cdot d^2$.

Proof. Moon and Moser showed in [34] that the number of maximal cliques in a graph with n vertices is in $\emptyset 3^{n/3}$. Now, a call to `GraphCliqueEnum` in Line 5 outputs pairs $(C, \mathcal{N}_C)$ where each C is a clique (different from the others) of the sub-graph composed of the vertices of $\Gamma_{G_t}(u) \cap \Gamma_{G_t}(v) \cup \{u, v\}$. As there are at most $d + 2$ vertices in this sub-graph, the number of pairs $(C, \mathcal{N}_C)$ output by this call such that C is a maximal clique of G_t, is in $\emptyset 3^{d/3}$. Now, each of these maximal cliques is of size at most q by definition, so it contains at most 2^q different sub-cliques. Therefore, the total number of pairs $(C, \mathcal{N}_C)$ enumerated by such a call is in $\emptyset 2^q \cdot 3^{d/3}$. Finally, such a call is made for each link of the link stream, so the total number of pairs $(C, \mathcal{N}_C)$ enumerated during Algorithm 2 is in $\emptyset m \cdot 2^q \cdot 3^{d/3}$. In other words, α_T is in $\emptyset m \cdot 2^q \cdot 3^{d/3}$. Finally, from Lemma 5, the time complexity of the algorithm is in $\emptyset m \cdot 3^{d/3} \cdot 2^q \cdot d^2$. $\square$

By contrast, the complexity of the algorithm in Viard *et al.* [44] is in $\emptyset 2^n \cdot n^2 \cdot m^2 \cdot (n + \log(m))$ with $n = |V|$; in Himmel *et al.* [17] it is in $\emptyset m \cdot 3^{c/3} \cdot 2^c \cdot n \cdot |T|$ with c the maximal degeneracy of a graph G_t (notice that $q - 1 \leq c \leq d$); in Bentert *et al.* [5], it is in $\emptyset 2^c \cdot \min(m^2, |T|^2) \cdot n^4$. While not directly comparable to ours, these three complexities are all products of at least two of the parameters n, m and $|T|$ of the link stream, while there is only one single factor m among those in ours (we recall that $|T| \leq 2m$). Note that all these complexities also depend on c, q or d, but these parameters are known to vary in much smaller ranges than n, m and $|T|$ when considering real data. Thus, this observation suggests that our algorithm might be more able to scale up to larger link streams, which we confirm experimentally in Section 6.

Regarding the complexity of the algorithm with a pivot, notice that in the worst case, each time-maximal clique is a maximal clique of the link stream. In that case, the pivot cannot prune any recursive call. Thus, the worst-case complexity is the same as the one of the version without a pivot. Nevertheless, we will study in more details its impact in Section 5.2.2 in which we study the output-sensitive complexity.

Finally, we show in Theorem 4 that the memory requirement of our algorithm is close to the size m of the link stream. It is possible to show that the algorithms proposed in [17] and [5] have the same memory complexity as ours, while the one in [44] is exponential.

Theorem 4 (Memory complexity)**.** *The spatial complexity of Algorithm 2 is in* $\emptyset m + q \cdot d$.

Proof. In practice, we store the m links of the link stream. We also need to store $ForbidEdges$ which is at most of size m. Notice that it is possible to perform the maximality test of Line 8 inside the function `GraphCliqueEnum`, which eliminates the need to store more than one clique at any given time, it can thus be done using a memory space in $\emptyset q \subseteq \emptyset m$. Finally, we have to compute the memory cost of the calls to `GraphCliqueEnum`. Each call at Line 5 generates a stack of recursive calls of size at most q and for each of them, we store R, P, X and Q, each of these sets being of size at most d. The stack therefore demands a space complexity in $\emptyset q \cdot d$. All these data structures add up to a memory complexity in $\emptyset m + q \cdot d$. □

5.2.2 Output-sensitive complexity

We formulate here the time complexity as a function of α, the number of maximal cliques output by the algorithm, and q the maximum number of vertices in a clique. To do this, we consider the search trees of recursive calls to `GraphCliqueEnum`. The internal nodes of these trees correspond to the calls for which the loop on Line 13 is not empty (i.e. which generates other child calls), while the leaves correspond to the calls that do not generate any subsequent call. Note that all leaves correspond to time maximal cliques.

Inspired by the work of Conte *et al.* [10], in what follows, we focus on the leaves of these search trees, which we separate into two categories: those which output a pair $(C, \mathcal{N}_C)$ that corresponds to a maximal clique of the link stream, and those whose output pair does not correspond to a maximal clique. The latter create unnecessary computations because they do not contribute to the enumeration. An optimal pivot strategy would cut off the branches of the search forest so that each leaf always outputs a maximal clique. We denote l the total number of leaves of the search forest, ℓ_{max} the number of leaves that correspond to maximal cliques, and $\ell_{\neg max}$ the ones which are not, so that $l = \ell_{max} + \ell_{\neg max}$. We are then interested in the ratio of "good" leaves:

$$r = \frac{\ell_{max}}{l} = \frac{\ell_{max}}{\ell_{max} + \ell_{\neg max}}.$$

This ratio can be computed either for the algorithm without a pivot or the one with a pivot. In the first case, it quantifies the maximal possible efficiency of the pivot: if r is lower than 1 this means that there are recursive calls which are not necessary and might be pruned by a pivot. Comparing the ratios obtained by the algorithms with and without the pivot shows to what extent the pivoting strategy has successfully pruned unnecessary calls.

Using this ratio allows us to describe the time complexity of Algorithm 2 as a function of the output:

Theorem 5 (Output-sensitive complexity)**.** *With the above definition of r, we have $1 \leq \frac{1}{r} \leq 2^q$, and the output-sensitive complexity of Algorithm 2 is in $\emptyset \frac{1}{r} \cdot d^2 \cdot q \cdot \alpha$.*

Proof. Let us show first that $1 \leq \frac{1}{r} \leq 2^q$. For this, we introduce c_{max}: the number of pairs $(C, \mathcal{N}_C)$ enumerated by `GraphCliqueEnum` such that C is a maximal clique of the associated graph G_t. These pairs can only be enumerated by calls corresponding to search tree leaves, because if C is a maximal clique of G_t, then it cannot be expanded by a recursive call. Moreover, since it cannot be expanded, its associated time-maximal clique is always vertex-maximal. Thus, c_{max} is such that $c_{max} \leq \ell_{max}$. Besides, each maximal clique of a graph G_t contains at most 2^q sub-cliques, so there are at most $2^q \cdot c_{max}$ pairs $(C, \mathcal{N}_C)$ listed in total. Now, there is at least one pair listed per leaf. Therefore, $l \leq 2^q \cdot c_{max}$, so $\frac{1}{r} \leq \frac{2^q \cdot c_{max}}{\ell_{max}} \leq 2^q$.

Let us now prove the expression for the time complexity. By definition of q, we know that the depth of a search tree is at most q. Then, there are at most $q \cdot l$ nodes in the whole search forest. We have seen in the proof of Lemma 5 that there is exactly one node per time-maximal clique. So, there are α_T nodes, which implies that $\alpha_T \leq q \cdot l$. So, using the expression of the time complexity in Lemma 5, we have that the complexity of Algorithm 2 is in $\emptyset d^2 \cdot q \cdot l$. As $l = \frac{1}{r} \cdot \ell_{max}$ and $\ell_{max} \leq \alpha$, the complexity is in $\emptyset d^2 \cdot q \cdot \frac{1}{r} \cdot \alpha$. ☐

Note that the bounds given by Theorem 5 are almost tight. First, $\frac{1}{r}$ can be equal to 1 for example when all the time-maximal cliques are also vertex-maximal, as in this way $\ell_{\neg max} = 0$. This is the case for instance when all the links end times are distinct. However, $\frac{1}{r}$ can also be exponential in q as suggested by its upper bound. To illustrate this, consider the example of a link stream that is equivalent to a graph because all its links have the same existence interval. Suppose that this link stream forms a clique. Then, it contains only one maximal clique, so that $\frac{1}{r} = l$. With the `BK` procedure on graphs (without pivot) described in Section 3.1, it is possible to show that the call tree has $l = 2^{q-1} - 1$ leaves of size ≥ 2. Thus, Algorithm 2 on this link stream can reach $\frac{1}{r} = 2^{q-1} - 1$.

Nevertheless, we will see in Section 6 that $\frac{1}{r}$ is experimentally small with a pivot: it is lower than 2 in all experiments except one. This means that it does not have an exponential behavior on the real world link streams that we study, by contrast with its theoretical upper bound. According to this observation and the computed complexity, we can state that the running time of our algorithm with a pivot is close to the best that we can expect. Indeed, since the running time must process all vertices of all maximal cliques (of which there are α) and the size of the largest ones is q, it is expected to have a factor $q \cdot \alpha$ in the complexity. Our algorithm only adds a multiplicative factor $\frac{1}{r} \cdot d^2$ to this, thus explaining its good performances.

6 Experimental evaluation

In this section, we compare the implementations of the algorithm that we have presented in Section 4 to the best ones provided in the literature. These are those of Viard *et al.* [44] [2], Himmel *et al.* [17] [3], and Bentert *et al.* [5] [4]. We do not compare our algorithm to the one of

[2] `https://bitbucket.org/tiph_viard/cliques`
[3] `https://fpt.akt.tu-berlin.de/temporalcliques/`
[4] `https://fpt.akt.tu-berlin.de/temporalkplex/`

Banerjee and Pal [2] as they study a generalization of the maximal clique enumeration and their implementation for the special case corresponding to our specific problem is not more efficient than the other algorithms mentioned. We do not compare either with the work of Molter *et al.* [33], as the authors explain that their algorithm is slower in all cases than that of Bentert *et al.*

Then, we show that our algorithm scales to massive real-world link streams of more than 100 million links, we study the impact of the pivot strategy on the computational time, and finally we present results of a parallel implementation.

6.1 Experimental setup

Hardware. We carried out the experiments on a machine equipped with 2 processors Intel Xeon Silver 4216 with 32 cores each and 384 GB of RAM.

Implementations. We have made two implementations of Algorithm 2 which are available online [5]: one in `Python` and another in `C++`. We use the `Python` implementation for comparison with the state-of-the-art methods which are coded in `Python`, and the `C++` one to scale up to more massive link streams. The `C++` implementation is inspired by the efficient implementation of the BK algorithm by Eppstein *et al.* [12] that enumerates maximal cliques on graphs. Since using a pivot reduces the running time, we use by default the algorithm with a pivot (as detailed in Section 4.4), except in Section 6.4, in which we analyze the efficiency of the pivot strategy.

Data and pre-processing. Following the work of Viard *et al.* in [44], we use datasets corresponding to *instantaneous* link streams in which all links have a duration equal to 0: we associate a duration Δ to each link (t, u, v) by replacing it by $(t, t + \Delta, u, v)$. Note that this may create overlapping links (b, e, u, v) and (b', e', u, v) with $[b, e] \cap [b', e'] \neq \emptyset$; we resolve this by iteratively replacing such pairs of links by $(\min(b, b'), \max(e, e'), u, v)$. Notice that this transformation implies that the number of links may decrease as Δ increases. This allows us to study the behavior of our algorithm with varied input, while still allowing to compare it to the ones that study Δ-cliques in instantaneous link streams, since Viard *et al.* [44] showed that both problems are equivalent. This pre-processing is performed before the main algorithm, and is therefore not taken into account in the computation times. Nevertheless, its running time is linear in the number of links in the instantaneous link streams and is negligible in practice when compared to the enumeration time of maximal cliques.

We use two different families of datasets. The first one, whose main characteristics are detailed in Table 1, corresponds to the datasets used in Bentert *et al.* [5] to compare their own algorithm to the literature. In this case, Δ values are chosen identical to those used in [5], for comparison purposes. The second one corresponds to a set of more massive datasets, whose characteristics are given in Table 2. We use these datasets to show that our algorithm scales to link streams up to several tens of millions of links. In this family, Δ values are chosen as functions of Θ, the total duration of the link stream: either 0 (instantaneous), $\Theta/10000$, or $\Theta/100$. The results of the corresponding experiments are detailed respectively in Section 6.2 and Section 6.3.

These two tables illustrate the effect of increasing Δ: the number of links m decreases (as explained above), while the maximum degree d and maximum clique size q increase. Indeed, increasing link duration naturally makes instantaneous graphs denser overall, which leads to an increase of d and q. However, the effect on the number of maximal cliques α is not the same in all cases. In some cases, α increases, probably because the increased density in instantaneous

[5]`https://gitlab.lip6.fr/baudin/maxcliques-linkstream`

graphs induces more cliques. In other cases α decreases, possibly because cliques involving the same vertices at different times are merged.

Dataset	$\boldsymbol{\Delta}$	$\boldsymbol{m}$	$\boldsymbol{d}$	$\boldsymbol{\alpha}$	$\boldsymbol{q}$	**Dataset**	$\boldsymbol{\Delta}$	$\boldsymbol{m}$	$\boldsymbol{d}$	$\boldsymbol{\alpha}$	$\boldsymbol{q}$
	0	8,111	72	8,111	2		***0***	125,773	4	106,879	5
dnc [23]	***125***	7,967	99	8,038	4	***primaryschool [15]***	***125***	49,530	16	67,820	6
	3125	7,454	108	7,834	4		***3125***	19,513	50	194,231	14
	0	20,818	9	19,037	6		***0***	188,508	4	172,035	5
hypertext [20]	***125***	6,323	14	6,859	7	***highschool13 [29]***	***125***	36,277	14	41,534	6
	3125	4,082	48	6,308	7		***3125***	15,764	30	28,357	8
	0	28,539	8	26,384	5		***0***	312,164	7	294,269	3
highschool11 [13]	***125***	6,472	19	7,732	7	***london [45]***	***125***	27,595	7	28,683	3
	3125	3,636	34	7,500	10		***3125***	768	7	778	3
	0	32,424	7	27,835	5		***0***	353,226	8	342,540	3
hospital-ward [42]	***125***	7,971	12	9,731	6	***paris [45]***	***125***	50,248	8	51,084	3
	3125	3,033	25	9,856	9		***3125***	1,080	8	1,093	3
	0	45,047	5	42,105	5		***0***	415,000	134	229,144	19
highschool12 [13]	***125***	11,329	10	12,115	5	***flights [45]***	***125***	415,000	134	229,144	19
	3125	5,691	18	7,268	7		***3125***	37,862	139	368,756	19
	0	59,795	31	59,795	2		***0***	415,912	4	338,815	5
facebooklike [36]	***125***	50,056	78	50,080	3	***infectious [20]***	***125***	100,329	15	138,670	7
	3125	34,116	92	34,342	4		***3125***	44,767	43	150,883	16
	0	110,581	1,458	97,687	10		***0***	468,897	12	442,341	3
as-733 [26]	***125***	32,485	1,568	41,965	12	***ny [45]***	***125***	113,651	12	117,481	3
	3125	21,466	1,851	49,293	18		***3125***	661	12	674	3

Table 1: Characteristics of the link stream datasets investigated by Bentert *et al.* [5]. $\boldsymbol{\Delta}$ (in seconds) corresponds to the duration added to each link of the stream, $\boldsymbol{m}$ is the number of links, $\boldsymbol{d}$ the maximal degree, $\boldsymbol{\alpha}$ the number of maximal cliques, and $\boldsymbol{q}$ the maximal number of nodes of a clique.

Dataset	$\boldsymbol{\Delta}$	$\boldsymbol{m}$	$\boldsymbol{d}$	$\boldsymbol{\alpha}$	$\boldsymbol{q}$
	0	1,108,715	3	1,108,715	2
stackexchange [37]	***23,961***	870,128	47	894,317	5
	2,396,149	751,974	671	1,030,023	10
	0	6,092,321	22	6,092,321	2
wikitalk [37]	***20,048***	4,123,960	12,205	4,207,362	7
	2,004,838	3,078,861	29,543	4,500,754	10
	0	12,223,774	28,714	12,253,571	17
youtube [31]	***1,944***	12,223,774	28,714	12,253,571	17
	194,400	10,310,419	28,714	10,656,065	17
	0	18,613,039	79	131,251	80
copresence-Thiers [39]	***38***	3,857,645	100	412,553	80
	3,804	147,125	217	5,572,145	102
	0	39,949,279	3,463	39,935,611	13
wikipedia [31]	***19,318***	39,246,821	7,216	40,898,684	28
	1,931,869	38,737,308	36,121	45,440,988	28
	0	47,902,566	4	47,902,566	2
stackoverflow [37]	***23,970***	33,948,538	90	35,298,283	6
	2,397,055	29,583,489	1,443	43,989,692	13
	0	122,378,012	3,769	119,172,078	219
soc-bitcoin [39]	***15,653***	93,897,987	28,144	787,519,128	237
	1,565,366	86,668,193	170,760	-	-

Table 2: Characteristics of large link stream datasets used to investigate the scaling properties of the algorithm. Δ values (in seconds) are set to 0, $\Theta/10000$ and $\Theta/100$, where Θ is the total duration of the link stream. A hyphen ("-") means that none of the implementations available allows enumerating all maximal cliques.

6.2 Comparison to the state of the art

In Table 3 we present the computation times (in seconds) for the datasets of Table 1, which are those studied by Bentert *et al.* [5]. As the state-of-the-art algorithms are implemented in `Python`, we first compare them with our `Python` implementation to evaluate the improvement brought by our algorithm. We observe that our algorithm is the fastest in all tested cases. Among the three state-of-the-art algorithms, **HMNS** [17] is clearly the least competitive, while **BHM+** [5] and **VML** [44] algorithms yield comparable results in terms of running times: depending on the link stream, one or the other is faster, while remaining in the same order of magnitude. Besides, we also observe that our implementation in `C++` is very efficient on these datasets: in most cases, the computation time is less than 1 second, and never exceeds 3 seconds, even on the *flights* dataset, for which it is typically 1,000 times faster than the state of the art, and 30 times faster than our `Python` implementation.

To study more precisely the difference of performance, we visualize the computation times of the different methods on all link streams in Figure 5 (left panel). The different link streams are ordered on the X-axis by increasing number of links m. Note that the scales are logarithmic. There is one experiment per value of Δ for each dataset, and three values are displayed, corresponding to the running times of the fastest state-of-the-art method, our `Python` implementation, and our `C++` implementation. The three horizontal lines at the top of the figure correspond to the runs for which the computation is interrupted, *i.e.* exceeds 24 hours or 380 GB of RAM. We also placed a vertical line to show the limit above which the state-of-the-art algorithms fail to provide a result. We observe three distinct layers of points, which illustrates that the `C++` implementation is more efficient than the `Python` implementation, itself more efficient than the state of the art. Finally, we notice a trend to have slightly higher gain with larger link streams.

Then, to better evaluate the gain, we display the speed-up factor achieved by our two implementations in the right panel of Figure 5, compared to the most efficient algorithm of the state of the art. We display a point for each dataset where at least one state-of-the-art algorithm is able to provide a result in less than 24 hours and 380 GB of RAM. The speed-up factor is defined as the state-of-the-art execution time divided by the time of our `Python` and `C++` implementations. The position of the line $y = 1$ at the bottom shows that this factor is always larger than 1. Thus, both `Python` and `C++` implementations are more efficient than the other implementations available, in all the experiments performed. In most experiments, the speed-up factor is larger than 10, even for the `Python` implementation. Concerning the `C++` implementation the speed-up factor is always larger than 10 except for the 3 smallest link streams, and it can reach up to 10^4 for larger link streams.

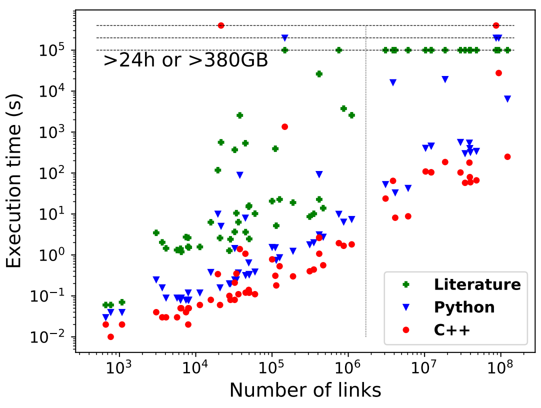
6.3 Scaling up to massive real-world link streams

In the previous section, we have seen that our algorithm is more efficient than those of the state of the art. It is thus relevant to see to what extent it can be used on larger datasets. Table 4 presents the running times of the maximal clique enumerations performed on the massive link streams described in Table 2. In this table, a "-" symbol means that the enumeration does not finish within 24 hours for the algorithm under examination and a "$\times$" symbol indicates that it requires more than 380 GB of RAM.

We observe that the state-of-the-art implementations do not allow enumerating the maximal cliques on these datasets, except for **VML** [44] on the dataset *stackexchange* with $\Delta = 0s$ and $\Delta = 23,961s$, which is much less efficient than both our implementations. In all other cases, the experiments are interrupted either for time or for memory consumption reasons. The `Python` implementation of our algorithm allows enumerating maximal cliques for all experiments except for 3. The `C++` implementation allows scaling the computation further, as it completes

Dataset	Δ	C++	Python	BHM+	VML	HMNS
dnc	*0*	0.05	0.08	24	1.5	178
	125	0.05	0.08	24	2.6	155
	3,125	0.04	0.08	23	2.7	80
hypertext	*0*	0.06	0.16	2.6	2.8	65
	125	0.05	0.08	1.4	1.8	6.9
	3,125	0.03	0.09	1.4	2.0	3.6
highschool11	*0*	0.08	0.20	2.4	2.6	95
	125	0.05	0.09	1.2	1.6	5.4
	3,125	0.03	0.16	2.0	4.1	3.1
hospital-ward	*0*	0.08	0.25	3.7	4.3	271
	125	0.02	0.12	1.6	2.6	17
	3,125	0.04	0.25	3.5	7.8	5.0
highschool12	*0*	0.12	0.32	3.6	4.2	183
	125	0.06	0.12	1.6	2.1	12
	3,125	0.03	0.09	1.3	1.5	3.8
facebooklike	*0*	0.11	0.39	27	10	129
	125	0.12	0.34	26	15	96
	3,125	0.35	0.25	25	10	59
as-733	*0*	0.31	1.5	569	392	-
	125	0.21	1.5	368	1,581	528
	3,125	0.11	5.1	562	×	572
primaryschool	*0*	0.53	0.87	22	30	716
	125	0.14	0.65	15	37	125
	3,125	0.34	10	204	854	117
highschool13	*0*	0.30	1.2	19	24	1,420
	125	0.11	0.37	6.3	11	54
	3,125	0.08	0.38	6.3	11	14
london	*0*	0.40	1.8	8.6	11	3,712
	125	0.10	0.20	2.1	1.3	45
	3,125	0.01	0.04	1.5	0.06	1.6
paris	*0*	0.44	2.0	10	13	4,260
	125	0.12	0.33	2.8	2.5	101
	3,125	0.02	0.04	1.7	0.07	1.8
infectious	*0*	1.1	3.1	665	22	1,206
	125	0.78	1.6	634	20	945
	3,125	1.1	8.1	818	534	1,004
flights	*0*	2.7	93	36,859	×	26,109
	125	2.6	93	37,076	×	26,411
	3,125	1.4	89	13,420	×	2,555
ny	*0*	0.56	2.8	13	18	6,084
	125	0.18	0.75	5.2	6.9	304
	3,125	0.02	0.03	2.3	0.06	2.5

Table 3: Comparison of the computation times (in seconds) of our implementations (**C++** and **Python**) to the state-of-the-art implementations: **BHM+** [5], **VML** [44] and **HMNS** [17] on the link streams listed in [5] described in Table 1. A "-" symbol means that the computation time exceeds 24 hours, and a "×" symbol means that the memory needed for the computation exceeds 380 GB.



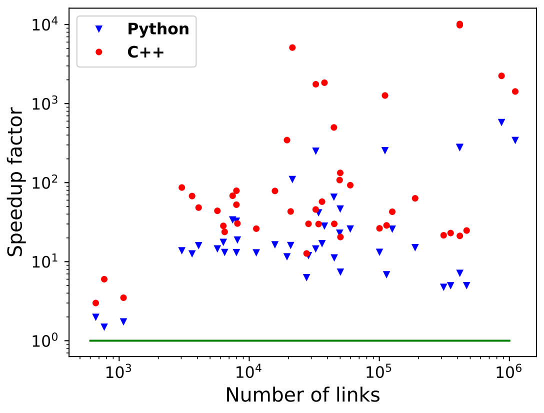


Figure 5: **Left**: Summary of the computation times of maximal clique enumerations as a function of the number m of links for all link stream datasets in Tables 3 and 4. The three lines at the top represent enumerations that are interrupted because they exceed 24 hours or 380 GB of RAM. **Right**: Speed-up factor of our implementations with respect to the fastest state-of-the-art method, as a function of the number m of links. There is one point per dataset where at least one state-of-the-art algorithm finishes in less than 24 hours and using less than 380 GB RAM.

the enumeration for all experiments except for 1: the *soc-bitcoin* link stream with link duration $\Delta = 1,565,366s$. We observe a significant gain of the `C++` implementation compared to the `Python` implementation.

Dataset	**Δ**	**C++**	**Python**	**BHM+**	**VML**	**HMNS**
	0	1.8	7.5	-	2,576	-
stackexchange	***23,961***	1.7	6.4	-	3,747	-
	2,396,149	1.9	10	-	×	-
	0	8.8	42	-	×	-
wikitalk	***20,048***	8.1	33	-	×	-
	2,004,838	23	53	-	×	-
	0	103	459	-	×	-
youtube	***1,944***	104	461	-	×	-
	194,400	109	408	-	×	-
	0	185	19,525	-	×	-
copresence-Thiers	***38***	64	16,305	-	×	-
	3,804	1,347	-	-	×	-
	0	59	322	-	×	-
wikipedia	***19,318***	79	403	-	×	-
	1,931,869	179	547	-	×	-
	0	66	343	-	×	-
stackoverflow	***23,970***	57	302	-	×	-
	2,397,055	103	568	-	×	-
	0	249	6,505	-	×	-
soc-bitcoin	***15,653***	27,660	-	-	×	-
	1,565,366	-	-	-	×	-

Table 4: Comparison of the computation times (in seconds) of our implementations (**C++** and **Python**) to the state-of-the-art implementations: **BHM+** [5], **VML** [44] and **HMNS** [17] on the massive link streams described in Table 2. A "-" symbol means that the computation time exceeds 24 hours, and a "×" symbol means that the memory needed for the computation exceeds 380 GB.

6.4 Efficiency of the pivot

In this section, we analyze and comment on the gain of computation time achieved by using a pivot, which was presented in Section 4.4. We recall that the complexity of Algorithm 2, as given by Theorem 5, is in Ø$\frac{1}{r} \cdot d^2 \cdot q \cdot \alpha$, where d is the maximal degree in an instantaneous graph G_t, q is the maximal size of a clique, α is the number of maximal cliques and r is the ratio of leaves that corresponds to maximal cliques in the tree of calls of `GraphCliqueEnum`. r can be computed for the enumeration performed without or with the pivot. In the first case, r quantifies the potential gain that the pivot can bring; in the second case, it quantifies how efficient the pivot actually is: the closer r gets to 1, the fewer useless branches remain in the tree.

In Table 5 we give the computation times of the `C++` implementation with and without a pivot for the massive link streams of Table 2. In both cases we report the enumeration times t, the ratio r mentioned above and the factor $1/r$, as it appears in the complexity expression of Theorem 5.

We first observe that the pivot allows to achieve the enumeration of maximal cliques faster. The speed-up factor may vary a lot from a dataset to another and in 5 cases, the pivot version terminates within our time limit while the version without pivot does not. The ratio r is larger than 0.9 in 18 experiments out of 21 with the pivot, while it is lower in most experiments without it. This indicates that the pivot allows to cut off almost all unnecessary branches of recursive calls, except for the *copresence-Thiers* dataset. Notice that this dataset with a link duration $\Delta = 0s$ clearly stands out, as there are only 8.7% of the leaves of the call trees of `GraphCliqueEnum` which correspond to maximal cliques for the enumeration with the pivot.

We can also see that the factor $1/r$, which can be very large in theory since it is in Ø2^q according to Theorem 5, remains relatively small in the experiments. This factor never exceeds

2 with the pivot, except in the case of *copresence-Thiers* with $\Delta = 0s$ mentioned above. Thus, from this observation, the complexity with the pivot can often be considered in practice as in $\emptyset d^2 \cdot q \cdot \alpha$, which is within a d^2 factor of the output size (in $\emptyset q \cdot \alpha$).

Finally, even with a pivot, the computation does not succeed for the largest value of Δ on the *soc-bitcoin* dataset within the boundaries of the experimental procedure. There are other larger datasets on which no method is able to produce a result, as for instance the link stream *flickr* [32] which has very dense instantaneous graphs. Based on observations not reported here, we suggest that this is not due to the size of the output, but rather to the fact that the pivot does not succeed in pruning a sufficient number of branches.

Link stream		**With pivot**			**Without pivot**		
Dataset	**Δ**	***t***	***r***	**$1/r$**	***t***	***r***	**$1/r$**
	0	1.8	1.000	1	1.8	1.000	1
stackexchange	***23,961***	1.7	1.000	1	1.7	1.000	1
	2,396,149	1.9	0.997	1.003	1.7	0.928	1.078
	0	8.8	1.000	1	7.7	1.000	1
wikitalk	***20,048***	8.1	1.000	1	7.7	1.000	1
	2,004,838	23	0.995	1.005	23	0.891	1.122
	0	103	0.907	1.103	135	0.332	3.012
youtube	***1,944***	104	0.907	1.103	135	0.332	3.012
	194,400	109	0.900	1.111	137	0.298	3.356
	0	185	0.087	11.49	-	-	-
copresence-Thiers	***38***	64	0.757	1.321	-	-	-
	3,804	1,347	0.854	1.171	-	-	-
	0	59	1.000	1	78	0.987	1.013
wikipedia	***19,318***	79	1.000	1	1,143	0.043	23.26
	1,931,869	179	0.999	1.001	1,618	0.034	29.41
	0	66	1.000	1	90	1.000	1
stackoverflow	***23,970***	57	1.000	1	73	1.000	1
	2,397,055	103	0.996	1.004	122	0.881	1.135
	0	249	0.972	1.029	-	-	-
soc-bitcoin	***15,653***	27,660	0.911	1.098	-	-	-
	15,653,366	-	-	-	-	-	-

Table 5: Comparison of the computation times, in seconds, of the `C++` implementation using a pivot to the one without a pivot. The factor r, defined in Section 5.2.2, is equal to the ratio of leaves in the call trees of `GraphCliqueEnum` that correspond to a maximal clique of the link stream.

6.5 Parallel experiments

In this section, we study a parallel version of the code to evaluate the speed-up that it brings. Algorithm 2 is indeed easily parallelizable, as the iterations of the loop on T at Line 1 are independent of each other. Our procedure consists in splitting the total time interval of duration Θ of the link stream into n_{th} sub-intervals, where n_{th} is the number of threads on which we perform the parallelization. For each sub-interval in parallel, the corresponding thread enumerates all maximal cliques that start during this interval, following the loop of Line 1. We choose to split the total time interval of the link stream in such a way that approximately the same number of links starts within each sub-interval. Thus, each thread processes approximately $\frac{m}{n_{th}}$ links. According to the expression established by Theorem 3, the sequential complexity is in $\emptyset m \cdot 3^{d/3} \cdot 2^q \cdot d^2$, so by dividing the number of links by n_{th}, the theoretical complexity is now in $\emptyset \frac{1}{n_{th}} \cdot m \cdot 3^{d/3} \cdot 2^q \cdot d^2$ per thread.

Figure 6 illustrates the results of the parallelization process on the massive link stream datasets detailed in Table 2. It reports the execution time for the enumeration of maximal cliques as a function of the number of threads used. On the left are the link streams for which parallelization offers an interesting reduction in computation times, while we show on the right the link streams for which parallelization does not yield any significant improvement. We observe different behaviors depending on the link stream under study: for example, the parallelization of *wikipedia* with $\Delta = \Theta\ /\ 100$ leads to a division by up to three of the enumeration time and even more for *copresence-Thiers* with $\Delta = 0s$, while the same process on link streams on the right brings very little gains. Also, we observe that for almost all datasets, the computation times do not decrease significantly when using more than 4 threads, by contrast with the theoretical expression of the complexity established above. This low speed-up can be explained by the distribution of the link stream density through time. Indeed, the density of the link stream is the maximal density of G_t at each instant t, which is not affected by the parallelization process. Even a short sub-interval of time can be associated to a dense link stream and thus generate many maximal cliques. As the overall computation time is set by the sub-interval that takes the longest time to compute, a dense sub-stream creates a bottleneck and entails a low speed-up.

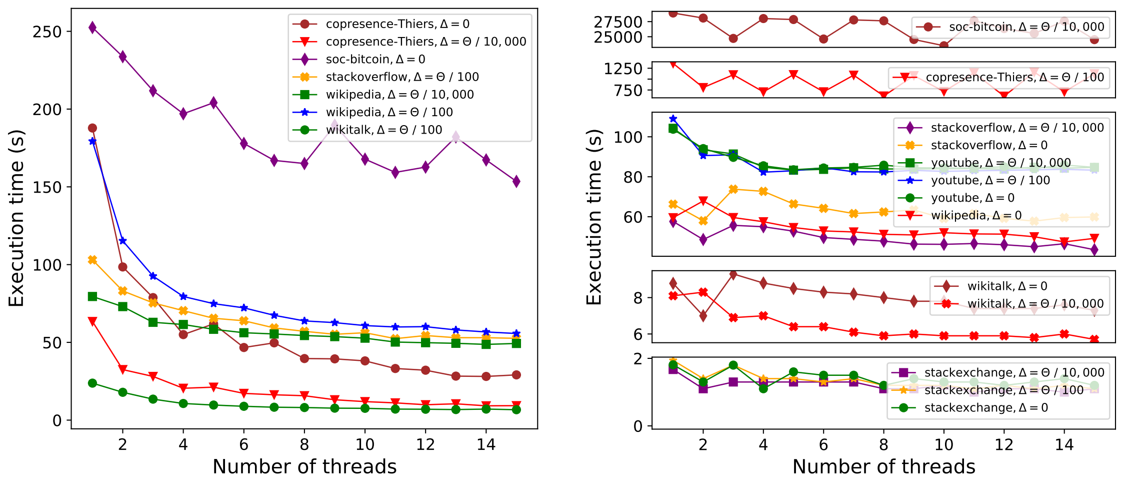


Figure 6: Computation times (in seconds) as a function of the number of threads for the parallel version of the `C++` implementation on the massive link stream datasets detailed in Table 2. The duration Δ of the links is expressed as a function of Θ, the total duration of the link stream. **Left:** Link streams for which parallelization offers an interesting reduction in computation time. **Right:** Link streams for which parallelization does not work.

7 Conclusion

In this paper, we have addressed the problem of maximal clique enumeration in link streams. We propose a new algorithm to solve this problem that scales to massive real-world link streams. We analyze its complexity as a function of the characteristics of the input and as a function of the characteristics of the output of the algorithm We provide two implementations (in `Python` and in `C++`) and perform an experimental protocol on various datasets from real interactions over time. It shows that our algorithm allows a performance gain of several orders of magnitude with respect to the state of the art.

The work done in this paper can be pursued along several directions. One is to investigate the reasons why the enumeration takes so long on some link streams, especially in the cases where the time limit of our protocol is reached. For instance, as the cliques are output on the fly, we could compute the $\frac{1}{r}$ factor appearing in the output-sensitive complexity on the fly and find out whether the computation stalls because it explores many unnecessary branches, or if it comes from the sheer number of maximal cliques. Another direction would be to improve the parallelization process by refining the partitioning of the link stream based on its structural properties and balance the computation more fairly on the different threads. For instance, we can study the structure of the link stream to anticipate the instants $t \in T$ where there are more or less maximal cliques, which would allow the construction of more suitable sub-intervals and thus a better speedup. Also, it would be interesting to study the impact of the link duration on the enumeration of maximal cliques, both in terms of computation times and number and size of cliques output.

More broadly, listing other kinds of dense sub-streams in link streams is a relevant problem. In this direction, we proposed in another work [4] an algorithm to extend to the context of links streams the well-known clique percolation method which defines communities in graphs [35]. It demands to list k-cliques in link streams, which is a different problem from the one of listing maximal cliques (and computationally more tractable for small values of k). We showed that using fast k-clique enumeration processes allows obtaining communities more efficiently than the other extension to the dynamical context of the clique percolation method [6]. As other types of dense sub-streams have been recently proposed [5, 33, 2], it would be interesting to generalize our enumeration method to these cases.

Acknowledgements

This work is financed by the ANR (French National Agency of Research) through the ANR FiT LabCom. The authors thank the SocioPatterns [6], Konect [23] and SNAP [27] communities, for making their datasets available.

References

[1] P. Bajardi, C. Poletto, J. J. Ramasco, M. Tizzoni, V. Colizza, and A. Vespignani. Human mobility networks, travel restrictions, and the global spread of 2009 h1n1 pandemic. *PloS one*, 6(1):e16591, 2011.

[2] S. Banerjee and B. Pal. An efficient updation approach for enumerating maximal (δ, γ)-cliques of a temporal network. *Journal of Complex Networks*, 10(5):cnac027, 2022.

[3] A. Baudin, M. Danisch, S. Kirgizov, C. Magnien, and M. Ghanem. Clique percolation method: memory efficient almost exact communities. In *International Conference on Advanced Data Mining and Applications*, pages 113–127. Springer, 2022.

[4] A. Baudin, L. Tabourier, and C. Magnien. Lscpm: Communities in massive real-world link streams by clique percolation method. In *30th International Symposium on Temporal Representation and Reasoning (TIME 2023).* Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[5] M. Bentert, A.-S. Himmel, H. Molter, M. Morik, R. Niedermeier, and R. Saitenmacher. Listing all maximal k-plexes in temporal graphs. *Journal of Experimental Algorithmics (JEA)*, 24:1–27, 2019.

[6] `http://www.sociopatterns.org`

[6] S. Boudebza, R. Cazabet, F. Azouaou, and O. Nouali. Olcpm: An online framework for detecting overlapping communities in dynamic social networks. *Computer Communications*, 123:36–51, 2018.

[7] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[8] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 international conference on web search and data mining*, pages 95–106, 2008.

[9] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[10] A. Conte and E. Tomita. On the overall and delay complexity of the cliques and bron-kerbosch algorithms. *Theoretical Computer Science*, 899:1–24, 2022.

[11] A. Das, M. Svendsen, and S. Tirthapura. Incremental maintenance of maximal cliques in a dynamic graph. *The VLDB Journal*, 28(3):351–375, 2019.

[12] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*, pages 403–414. Springer, 2010.

[13] J. Fournet and A. Barrat. Contact patterns among high school students. *PloS one*, 9(9):e107878, 2014. Data available at: `http://www.sociopatterns.org/datasets/high-school-dynamic-contact-networks`.

[14] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

[15] V. Gemmetto, A. Barrat, and C. Cattuto. Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC infectious diseases*, 14(1):1–10, 2014. Data available at: `http://www.sociopatterns.org/datasets/primary-school-temporal-network-data`.

[16] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases*, pages 721–732, 2005.

[17] A.-S. Himmel, H. Molter, R. Niedermeier, and M. Sorge. Adapting the bron–kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):1–16, 2017.

[18] P. Holme and J. Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

[19] T. Hossmann, F. Legendre, and T. Spyropoulos. From contacts to graphs: Pitfalls in using complex network analysis for dtn routing. In *IEEE INFOCOM Workshops 2009*, pages 1–6. IEEE, 2009.

[20] L. Isella, J. Stehlé, A. Barrat, C. Cattuto, J. Pinton, and W. Van den Broeck. What's in a crowd? analysis of face-to-face behavioral networks. *Journal of Theoretical Biology*, 271(1):166–180, 2011. Data hypertext: `http://www.sociopatterns.org/datasets/hypertext-2009-dynamic-contact-network`. Data infectious: `http://www.sociopatterns.org/datasets/infectious-sociopatterns/`.

[21] D. M. Jacoby and R. Freeman. Emerging network-based tools in movement ecology. *Trends in Ecology & Evolution*, 31(4):301–314, 2016.

[22] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1-2):1–30, 2001.

[23] J. Kunegis. KONECT – The Koblenz Network Collection. In *Proc. Int. Conf. on World Wide Web Companion*, pages 1343–1350, 2013. Data available at: `http://konect.cc/networks`.

[24] M. Latapy, T. Viard, and C. Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, 8(1):1–29, 2018.

[25] Y. Léo, C. Crespelle, and E. Fleury. Non-altering time scales for aggregation of dynamic networks into series of graphs. *Computer Networks*, 148:108–119, 2019.

[26] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187, 2005. Data available at: `https://snap.stanford.edu/data/as-733.html`.

[27] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

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

[29] R. Mastrandrea, J. Fournet, and A. Barrat. Contact patterns in a high school: a comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PloS one*, 10(9):e0136497, 2015. Data available at: `https://journals.plos.org/plosone/article?id=10.1371%2Fjournal.pone.0136497#sec012`.

[30] D. Miorandi and F. De Pellegrini. K-shell decomposition for dynamic complex networks. In *8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 488–496. IEEE, 2010.

[31] A. Mislove. *Online Social Networks: Measurement, Analysis, and Applications to Distributed Information Systems.* PhD thesis, Rice University, Department of Computer Science, May 2009. Data available at: `https://socialnetworks.mpi-sws.org/data-wosn2008.html`.

[32] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Growth of the flickr social network. In *Proceedings of the first workshop on Online social networks*, pages 25–30, 2008. Data available at: `http://konect.cc/networks/flickr-growth`.

[33] H. Molter, R. Niedermeier, and M. Renken. Isolation concepts applied to temporal clique enumeration. *Network Science*, 9(S1):S83–S105, 2021.

[34] J. W. Moon and L. Moser. On cliques in graphs. *Israel journal of Mathematics*, 3(1):23–28, 1965.

[35] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *nature*, 435(7043):814–818, 2005.

[36] P. Panzarasa, T. Opsahl, and K. M. Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2009. Data available at: `http://snap.stanford.edu/data/CollegeMsg.html`.

[37] A. Paranjape, A. R. Benson, and J. Leskovec. Motifs in temporal networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 601–610, 2017. Data available at: `http://snap.stanford.edu/data#temporal`.

[38] F. Peruani and L. Tabourier. Directedness of information flow in mobile phone communication networks. *PloS one*, 6(12):e28860, 2011.

[39] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. Data available at: `https://networkrepository.com/temporal-networks.php`.

[40] S. Sun, Y. Wang, W. Liao, and W. Wang. Mining maximal cliques on dynamic graphs efficiently by local strategies. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 115–118. IEEE, 2017.

[41] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science*, 363(1):28–42, 2006.

[42] P. Vanhems, A. Barrat, C. Cattuto, J.-F. Pinton, N. Khanafer, C. Régis, B.-a. Kim, B. Comte, and N. Voirin. Estimating potential infection transmission routes in hospital wards using wearable proximity sensors. *PloS one*, 8(9):e73970, 2013. Data available at: `http://www.sociopatterns.org/datasets/hospital-ward-dynamic-contact-network/`.

[43] T. Viard, M. Latapy, and C. Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.

[44] T. Viard, C. Magnien, and M. Latapy. Enumerating maximal cliques in link streams with durations. *Information Processing Letters*, 133:44–48, 2018.

[45] M. J. Williams and M. Musolesi. Spatio-temporal networks: reachability, centrality and robustness. *Royal Society open science*, 3(6):160196, 2016. Data available at: `https://datadryad.org/stash/dataset/doi:10.5061/dryad.3p27r`.

[46] X. Zhao, A. Sala, C. Wilson, X. Wang, S. Gaito, H. Zheng, and B. Y. Zhao. Multi-scale dynamics in a massive online social network. In *Proceedings of the 2012 Internet Measurement Conference*, pages 171–184, 2012.