Computing maximal cliques in link streams

Tiphaine Viard, Matthieu Latapy, Clémence Magnien

Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris

Abstract

A link stream is a collection of triplets (t, u, v) indicating that an interaction occurred between u and v at time t. We generalize the classical notion of cliques in graphs to such link streams: for a given Δ, a Δ-clique is a set of nodes and a time interval such that all pairs of nodes in this set interact at least once during each sub-interval of duration Δ. We propose an algorithm to enumerate all maximal (in terms of nodes or time interval) cliques of a link stream, and illustrate its practical relevance on a real-world contact trace.

Keywords: link streams, temporal networks, time-varying graphs, cliques, graphs, algorithms

1. Introduction

In a graph $G = (V, E)$ with $E \subseteq V \times V$, a clique $C \subseteq V$ is a set of nodes such that $C \times C \subseteq E$. In addition, C is maximal if it is included in no other clique. In other words, a maximal clique is a set of nodes such that all possible links exist between them, and there is no other node linked to all of them. Enumerating maximal cliques of a graph is one of the most fundamental problems in computer science, and it has many applications [9, 10].

A link stream $L = (T, V, E)$ with $T = [\alpha, \omega]$ and $E \subseteq T \times V \times V$ models interactions over time: $l = (t, u, v)$ in E means that an interaction occurred between $u \in V$ and $v \in V$ at time $t \in T$. Link streams, also called temporal networks or time-varying graphs depending on the context, model many real-world data like contacts between individuals, email exchanges, or network traffic [2, 6, 12, 14].

For a given duration Δ, a Δ-clique C of L is a pair $C = (X, [b, e])$ with $X \subseteq V$ and $[b, e] \subseteq T$ such that $|X| \geq 2$, and for all $\{u, v\} \subseteq X$ and $\tau \in [b, \max(e-\Delta, b)]$ there is a link (t, u, v) in E with $t \in [\tau, \min(\tau + \Delta, e)]$. Notice that Δ-cliques necessarily have at least two nodes.

More intuitively, all nodes in X interact at least once with all others at least every Δ from time b to time e. Δ-clique C is maximal if it is included in no other Δ-clique, (i.e. there exists no Δ-clique $C' = (X', [b', e'])$ such that $C' \neq C$, $X \subseteq X'$ and $[b, e] \subseteq [b', e']$). See Figure 1 for an example.

In real-world situations like the ones cited above, Δ-cliques are signatures of meetings, discussions, or distributed applications for instance. Moreover, just

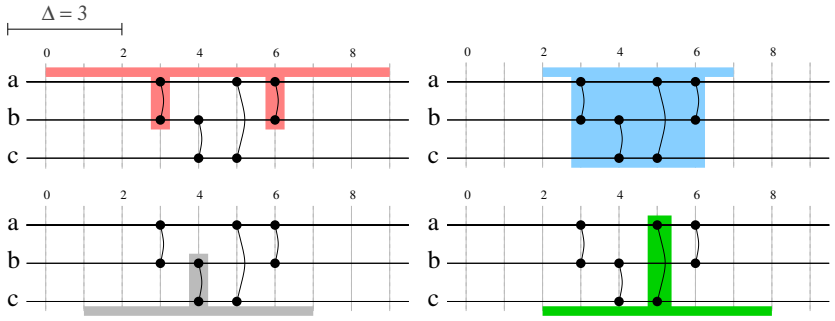

Figure 1: Examples of Δ-cliques. We consider the link stream $L = ([0, 9], \{a, b, c\}, E)$ with $E = \{(3, a, b), (4, b, c), (5, a, c), (6, a, b)\}$ and $\Delta = 3$. There are four maximal 3-cliques in L: $(\{a, b\}, [0, 9])$ (top left), $(\{a, b, c\}, [2, 7])$ (top right), $(\{b, c\}, [1, 7])$ (bottom left), and $(\{a, c\}, [2, 8])$ (bottom right). Notice that $(\{a, b, c\}, [1, 7])$ is not a Δ-clique since during time interval $[1, 4]$ of duration $\Delta = 3$ there is no interaction between a and c. Notice also that $(\{a, b\}, [1, 9])$, for instance, is not maximal: it is included in $(\{a, b\}, [0, 9])$.

like cliques in a graph correspond to its subgraphs of density 1, Δ-cliques in a link stream correspond to its substreams of Δ-density 1, as defined in [12]. Therefore, Δ-cliques in link streams are natural generalizations of cliques in graphs.

In this paper, we propose the first algorithm for listing all maximal Δ-cliques of a given link stream. We illustrate the relevance of the concept and algorithm by computing maximal Δ-cliques of a real-world dataset.

Before entering in the core of the presentation, notice that we consider here undirected links only: given a link stream $L = (T, V, E)$, we make no distinction between $(t, u, v) \in E$ and $(t, v, u) \in E$. Likewise, we suppose that there is no loop (t, v, v) in E, and no isolated node $(\forall v \in V,\ \exists (t, u, v) \in E)$.

We finally define the first occurrence time of (u, v) after b as the smallest time $t \geq b$ such that $(t, u, v) \in E$, and we denote it by f_{buv}. Conversely we denote the last occurrence time of (u, v) before e by l_{euv}. We say that a link (t, u, v) is in $C = (X, [b, e])$ if $u \in X$, $v \in X$ and $t \in [b, e]$.

2. Algorithm

One may trivially enumerate all maximal cliques in a graph as follows. One maintains a set M of previously found cliques (maximal or not), as well as a set S of candidate cliques. Then for each clique C in S, one removes C from S and searches for nodes outside C connected to all nodes in clique C, thus obtaining new cliques (one for each such node) larger than C. If one finds no such node, then clique C is maximal and it is part of the output. Otherwise, if the newly found cliques have not already been found (i.e., they do not belong to M), then one adds them to S and M. The set S is initialized with the trivial cliques containing only one node, and all maximal cliques have been found when S is empty. The set M is used for memorization, and ensures that one does not

examine the same clique more than once. In [7] the authors use this framework to enumerate all maximal cliques of a graph in lexicographic order.

Our algorithm for finding Δ-cliques in link stream $L = (T, V, E)$ (Algorithm 1) relies on the same scheme. We initialize the set S of candidate Δ-cliques and the set M of all found Δ-cliques with the trivial Δ-cliques $(\{a, b\}, [t, t])$ for all (t, a, b) in E (Line 2). Then, until S is empty (*while* loop of Lines 3 to 24), we pick an element $(X, [b, e])$ in S (Line 4) and search for nodes v outside X such that $(X \cup \{v\}, [b, e])$ is a Δ-clique (Lines 6 to 10). We also look for a value $b' < b$ such that $(X, [b', e])$ is a Δ-clique (Lines 11 to 16), and likewise a value $e' > e$ such that $(X, [b, e'])$ is a Δ-clique (Lines 17 to 22). If we find such a node, such a b' or such an e', then Δ-clique C is not maximal and we add to S and M the new Δ-cliques larger than C we just found (Lines 10, 16 and 22), on the condition that they had not already been seen (i.e., they do not belong to M). Otherwise, C is maximal and is part of the output (Line 24).

Algorithm 1 Maximal Δ-cliques of a link stream

input: a link stream $L = (T, V, E)$ and a duration Δ
output: the set of all maximal Δ-cliques in L

```
1:  S ← ∅, R ← ∅, M ← ∅
2:  for (t, u, v) ∈ E: add ({u, v}, [t, t]) to S and to M
3:  while S ≠ ∅ do
4:      take and remove (X, [b, e]) from S
5:      set isMax to True
6:      for v in V \ X do
7:          if (X ∪ {v}, [b, e]) is a Δ-clique then
8:              set isMax to False
9:              if (X ∪ {v}, [b, e]) not in M then
10:                 add (X ∪ {v}, [b, e]) to S and M
11:     f ← max_{u,v∈X} f_{buv}          ▷ latest first occurrence time of a link in (X, [b, e])
12:     set b' to f − Δ
13:     if b ≠ b' then
14:         set isMax to False
15:         if (X, [b', e]) not in M then
16:             add (X, [b', e]) to S and M
17:     l ← min_{u,v∈X} l_{euv}          ▷ earliest last occurrence time of a link in (X, [b, e])
18:     set e' to l + Δ
19:     if e ≠ e' then
20:         set isMax to False
21:         if (X, [b, e']) not in M then
22:             add (X, [b, e']) to S and M
23:     if isMax then
24:         add (X, [b, e]) to R
25: return R
```

Let us explain the choice of b' (Lines 11 to 16) in details, the choice of e'

(Lines 19 to 22) being symmetrical. For a given Δ-clique $(X, [b, e])$, we set b' to $f - \Delta$, which is the smallest time such that we are sure that $(X, [b', e])$ is a Δ-clique without inspecting any link outside of $(X, [b, e])$. Indeed, all links in $X \times X$ appear at least once in the interval $[f - \Delta, f]$: f is the latest of the first occurrence times of all links in this Δ-clique, and so all links appear at least once in $[b, f] \subseteq [f - \Delta, f]$. If $b' \neq b$, then the Δ-clique $(X, [b', e])$ is added to S (Line 13).

We display in Figure 2 an example of a sequence of such operations from an initial trivial Δ-clique to a maximal Δ-clique in an illustrative link stream. The algorithm builds this way a set of Δ-cliques of L, which we call the *configuration space*; we display the configuration space for this simple example in Figure 3 together with the relations induced by the algorithm between these Δ-cliques.

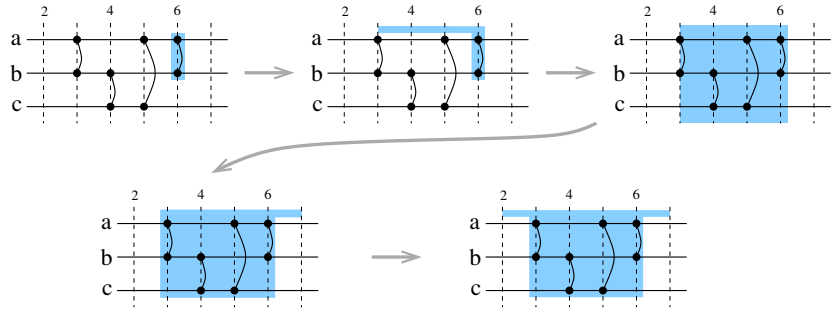

Figure 2: A sequence of Δ-cliques built by our algorithm to find a maximal Δ-clique (bottom-right) from an initial trivial Δ-clique (top-left) in the link stream of Figure 1 when $\Delta = 3$. From left to right and top to bottom: the algorithm starts with $(\{a, b\}, [6, 6])$, and finds $(\{a, b\}, [3, 6])$ thanks to Lines 11 to 16 of the algorithm. It then finds $(\{a, b, c\}, [3, 6])$ thanks to Lines 6 to 10. It finds $(\{a, b, c\}, [3, 7])$ from Lines 17 to 22, and finally $(\{a, b, c\}, [2, 7])$ from Lines 11 to 16.

To prove the validity of Algorithm 1, we must show that all the elements it outputs are Δ-cliques, that they are maximal, and that all maximal Δ-cliques are in its output.

Lemma 1. *In Algorithm 1, all elements of S are Δ-cliques of L.*

Proof. We prove the claim by induction on the iterations of the *while* loop (Lines 3 to 24).

Initially, all elements of S are Δ-cliques (Line 2). Let us assume that all the elements of S are Δ-cliques at the i-th iteration of the loop (induction hypothesis). The loop may add new elements to S at Lines 10, 16 and 22. In all cases, the added element is built from an element $C = (X, [b, e])$ of S (Line 4), which is a Δ-clique by induction hypothesis.

It is trivial (from the test at Line 7) that Line 10 only adds Δ-cliques.

Let us show that $(X, [b, l + \Delta])$, where l is computed in Line 17, necessarily is a Δ-clique. As $(X, [b, e])$ is a Δ-clique all links in $X \times X$ appear at least once every Δ from b to $l \leq e$. Moreover, since l is the earliest last occurrence

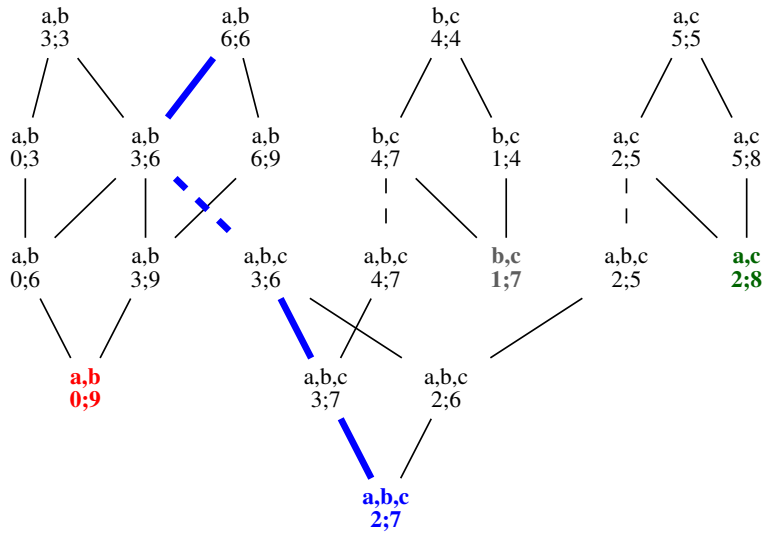

Figure 3: The configuration space built by our algorithm from the link stream of Figures 1 and 2 when $\Delta = 3$. Each element is a Δ-clique and it is linked to the Δ-cliques the algorithm builds from it (links are implicitly directed from top to bottom). Plain links indicate Δ-cliques discovered by Lines 11 to 16 or Lines 17 to 22 of the algorithm, which change the time span of the clique. Dotted links indicate Δ-cliques discovered by Lines 6 to 10, which change the set of nodes involved in the clique. The bold path is the one detailed in Figure 2. Colors correspond to the maximal Δ-cliques displayed in Figure 1.

time of a link in C, for all u and v in X there is necessarily a link (t, u, v) in E with $l \leq t \leq e$. Notice also that $l \geq e - \Delta$, otherwise $(X, [b, e])$ would not be a Δ-clique. Therefore a link between u and v occurs at least once between l and $l + \Delta$ for all u and v in X. Finally, $(X, [b, l + \Delta])$ is a Δ-clique.

The same arguments hold for Line 11.

Finally, at the end of the $(i + 1)$-th iteration of the loop, all the elements of S are Δ-cliques, which ends the proof. □

Lemma 2. *All the elements of the set returned by Algorithm 1 are maximal Δ-cliques of L.*

Proof. Let $C = (X, [b, e])$ be an element of R returned by the algorithm. Only elements of S are added to R (at Line 24), and so according to Lemma 1 C is a Δ-clique. Assume it is not maximal; then we are in one of the three following situations.

There exists v in $V \backslash X$ such that $(X \cup \{v\}, [b, e])$ is a Δ-clique. Then v is found at Lines 6–7, and Line 8 sets the boolean *isMax* to *false*. Therefore, Line 23 ensures that $C = (X, [b, e])$ is not added to R, and we reach a contradiction.

There exists $e' > e$ such that $(X, [b, e'])$ is a Δ-clique and we assume without loss of generality that there is no link between nodes in X from e to e'. Then, let us consider $l \in [b, e]$, computed in Line 17, which is the earliest last occurrence

time of a link in C. We necessarily have $l \geq e' - \Delta$ because $(X, [b, e'])$ is a Δ-clique. Since $e' > e$, this implies $l > e - \Delta$. As a consequence, the test in Line 19 of the algorithm is satisfied, and Line 20 sets the boolean *isMax* to *false*. Like above, we reach a contradiction.

If there exists $b' < b$ such that $(X, [b', e])$ is a Δ-clique, then similarly to the previous case we reach a contradiction.

Finally, C necessarily is maximal, which proves the claim. □

Before proving our main result, which is that all maximal Δ-cliques are returned by the algorithm, we need the following two intermediate results.

Lemma 3. *Let $C = (X, [b, e])$ be a maximal Δ-clique of L, and let s be the earliest occurrence time of a link in C. Then $e \geq s + \Delta$.*

Proof. Since C is a Δ-clique and by definition of s, for all u, v in X there exists at least one link (t, u, v) such that $s \leq t \leq e$. Assume $e < s + \Delta$; then for all u, v in X there also exists a link (t, u, v) such that $s \leq t \leq e < s + \Delta$. Therefore $(X, [b, s + \Delta])$ is a Δ-clique and C is included in it, which means that C is not maximal and we reach a contradiction. □

Lemma 4. *Let $C = (X, [b, e])$ be a maximal Δ-clique of L and let s be the earliest occurrence time of a link in C. If $(X, [s, s + \Delta])$ is in S at some stage of Algorithm 1, then C is in the set returned by the algorithm.*

Proof. Assume $C_0 = (X, [s, s + \Delta])$ is in S and consider the longest sequence of steps of Algorithm 1 of the form: $C_0 \to C_1 \to \cdots \to C_k$ such that for all i $C_i = (X, [s, e_i])$ with $e_{i+1} > e_i$. In other words, the algorithm builds C_{i+1} from C_i in Lines 17 to 22 (notice that $e \geq s + \Delta$ from Lemma 3 and so C_0 is included in C).

We prove that $C_k = (X, [s, e])$ by contradiction. Assume this is false, and so that $e_k \neq e$. As C is maximal, we then necessarily have $e_k < e$. In addition, $e_k = l + \Delta$ where l is the earliest last occurrence time of a link in C_{k-1} computed at Line 17. Since C_k is the last Δ-clique in the sequence, l is also the earliest last occurrence time of a link in C_k (otherwise there would be a clique C_{k+1} satisfying the constraints of the sequence above). Therefore there exist $u, v \in X$ such that $(l, u, v) \in E$ and such that there is no occurrence of a link (u, v) between l and $e_k = l + \Delta$. This ensures that there exists an ϵ such that $l + \Delta + \epsilon < e$ and such that there is no link between u and v from $l + \epsilon$ to $l + \Delta + \epsilon$, which contradicts the assumption that C is a Δ-clique.

We now show that the algorithm builds C from C_k to end the proof. Since C is maximal, there exists $u, v \in X$ such that $(b + \Delta, u, v) \in E$ and such that there is no other link between u and v from b to $b + \Delta$. By definition of s, $b + \Delta \geq s$. Therefore the latest first occurrence time of a link in C_k, f, is equal to $b + \Delta$ and Lines 11 to 16 build C from C_k. □

Lemma 5. *All maximal Δ-cliques of L are in the set returned by Algorithm 1.*

Proof. It is easy to check that if S contains a maximal Δ-clique then it is added to the set R returned by the algorithm, and only these Δ-cliques are added to R. We therefore show that all maximal Δ-cliques are in S at some stage.

Let $C = (X, [b, e])$ be a maximal Δ-clique of L, let s be the earliest occurrence time of a link in C, and let $u, v \in X$ be two nodes such that there exists a link between them at s (i.e., $(s, u, v) \in E$). We show that there is a sequence of steps of the algorithm that builds C from Δ-clique $C_0 = (\{u, v\}, [s, s])$ (which is placed in S at the beginning of the algorithm, Line 2).

Lines 17 to 22 builds $C_1 = (\{u, v\}, [s, s + \Delta])$ from C_0.

Notice that for all subset Y of X, $(Y, [s, s + \Delta])$ is a Δ-clique. Therefore the algorithm iteratively adds all elements of X at Lines 6 to 10, finally obtaining $C' = (X, [s, s + \Delta])$ from C_1.

We finally apply Lemma 4 to conclude that the algorithm builds C from C'. □

From these lemmas, we finally obtain the following result.

Theorem 1. *Given a link stream L and a duration Δ, Algorithm 1 computes the set of all maximal Δ-cliques of L.*

In order to investigate the complexity of our algorithm, let us denote by $n = |V|$ the number of nodes and $m = |E|$ the number of links in L. First notice that the number of elements in the configuration space built by the algorithm is bounded by the number of subsets of V times the number of sub-intervals of T.

Moreover, for all Δ-clique $C = (X, [b, e])$ in the configuration space, there exists a link (b, u, v) or a link $(b + \Delta, u, v)$ in E. Indeed, the initial trivial Δ-cliques are in the first case, and all Δ-cliques obtained from them are also in this case until Line 16 is applied. The Δ-cliques built after this are in the second case. Likewise, there exists a link (e, u, v) or a link $(e - \Delta, u, v)$ in E. Therefore, the number of possible values for b and e for any Δ-clique in the configuration space is proportional to the number of time instants at which a link occurs, which is bounded by the number of links m. The number of sub-intervals of T corresponding to a Δ-clique in the configuration space is therefore in $\mathcal{O}(m^2)$. This bound is reached in the worst case, for instance if the stream is a sequence of links occurring once every Δ time interval.

The trivial bound $\mathcal{O}(2^n)$ for the number of subsets of V is also reached in the worst case, for instance if there is a link between all pairs of nodes at the same time: the algorithm will enumerate all subsets of V.

Therefore, the number of elements in the configuration space is in $\mathcal{O}(2^n m^2)$. This leads to the space complexity of our algorithm: it is proportional to the space needed to store the configuration space, which is in $\mathcal{O}(2^n n m^2)$ since each element may be stored in $\mathcal{O}(n)$ space.

We estimate the time complexity by studying the complexity of operations performed on each element of the configuration space, (i.e., the complexity of each iteration of the *while* loop at Lines 3 to 24). Let us consider a Δ-clique $C = (X, [b, e])$ picked from S by the algorithm at Line 4.

The *while* loop is composed of three blocks: (1) searching for Δ-cliques of the form $(X \cup \{v\}, [b, e])$ larger than C (Lines 6 to 10); (2) searching for a Δ-clique $(X, [b', e])$ larger than C (Lines 11 to 16); and (3) searching for a Δ-clique $(X, [b, e'])$ larger than C (Lines 17 to 22). The third block has the same complexity as the second one, so we focus on the time complexity of the two first blocks.

Given a node $v \notin X$, Line 7 tests whether for all nodes u in X there is a link $(t, v, u) \in E$ in each time interval of duration Δ. This requires at most $|X| \cdot m$ tests, and so it is in $\mathcal{O}(nm)$. Then, Line 9 searches for the found Δ-clique in M, which has a size in $\mathcal{O}(2^n m^2)$. Since the comparison between two Δ-cliques can be performed in $\mathcal{O}(n)$ time, this search therefore is in $\mathcal{O}(n \log(2^n m^2)) = \mathcal{O}(n^2 + n \log m)$ time. The algorithm repeats these operations for all nodes $v \in V \setminus X$, and thus less than n times, hence the complexity of Lines 6 to 10 is in $\mathcal{O}(n(nm + n^2 + n \log m)) = \mathcal{O}(n^2 m + n^3)$.

Computing f in Line 11 may clearly be done with at most m tests. Lines 13 and 14 are all trivial computations. Lines 15 and Line 16 are in $\mathcal{O}(n^2 + n \log m)$. The complexity of Lines 11 to 16 is therefore in $\mathcal{O}(m + n^2 + n \log m)$.

Finally, each iteration of the while loop costs at most $\mathcal{O}(n^2 m + n^3 + m + n^2 + n \log m) = \mathcal{O}(n^2 m + n^3)$ time. We bound the overall time complexity of the algorithm by multiplying this by the number of iterations of the while loop, which is the number of elements in the configuration space. It is therefore in $\mathcal{O}(2^n m^2 (n^2 m + n^3)) = \mathcal{O}(2^n n^2 m^3 + 2^n n^3 m^2)$.

From this analysis, we obtain the following result:

Theorem 2. *Let $L = (T, V, E)$ be a link stream with $|V| = n$ and $|E| = m$, and let Δ be a duration, then Algorithm 1 computes the set of all maximal Δ-cliques of L in $\mathcal{O}(2^n nm^2)$ space and $\mathcal{O}(2^n n^2 m^3 + 2^n n^3 m^2)$ time.*

Notice that enumerating the maximal cliques in a graph $G = (V, E)$ is equivalent to enumerating the maximal Δ-cliques in $L = ([0, 0], V, E')$ where $(0, u, v) \in E'$ if and only if $(u, v) \in E$. The problem of enumerating maximal Δ-cliques in a link stream is therefore at least as difficult as enumerating maximal cliques in a graph, which has an exponential time complexity (in particular, there can be an exponential number of maximal cliques). Therefore any algorithm for enumerating maximal Δ-cliques in a link stream is at least exponential in the number of nodes.

Notice also that several optimizations may speed up our algorithm (without changing its worst-case complexity). In particular, f and l, computed in Lines 11 and 17, are necessarily in $[b, \min(e, b + \Delta)]$ and $[\max(b, e - \Delta), e]$, respectively. One may therefore focus the search on these intervals rather than $[b, e]$. Likewise, if $V(C)$ is the set of nodes satisfying condition of Line 7, then the set $V(C')$ of nodes satisfying this condition for the Δ-cliques C' added to S at Lines 10, 16 and 22 is included in $V(C)$. One may therefore associate to each element of S a set of candidate nodes to be considered at Line 6 in place of $V \setminus X$, thus drastically reducing the number of iterations of this loop.

3. Experiments

We implemented Algorithm 1 with the optimizations discussed above in Python (2.7) and provide the source code at [13]. We illustrate here its practical relevance by computing maximal Δ-cliques of the link stream from the Thiers-Highschool dataset, which is a trace of real-world contacts between individuals, captured with sensors. It was collected at a French high school in 2012, see [5] for full details. It induces a link stream of 181 nodes and $45,047$ links, connecting $2,220$ distinct pairs of nodes over a period of $729,500$ seconds (approximately 8 days). Each link (t, u, v) means that the sensor carried by individual u or v detected the sensor carried by the other individual at time t, which means in turn that these two individuals were close enough from each other at time t for the detection to happen. We call this a contact between individuals u and v. We also have the information of the class to which students belong.

We computed all maximal Δ-cliques for $\Delta = 60$ seconds, $\Delta = 900$ seconds (15 minutes), $\Delta = 3,600$ seconds (1 hour), and $\Delta = 10,800$ seconds (3 hours). We handpicked these values because of the rhythm of school day: on a typical day, courses usually last roughly two hours, with two 15 minutes breaks during the day, and a longer 1 hour lunch break. Our Python implementation took an hour on a standard server [1] to obtain the results. Although many discovered Δ-cliques are very small, we also found rather large and long ones. See Table 1 for a summary of these computations.

Δ (s)	$\lvert R\rvert$	Max $\lvert X\rvert$	Max $e-b$ (s)	Running time (s)	Memory (MB)
60	14 664	5	6 820	150	537
900	8 214	7	17 420	555	4 755
3 600	7 170	7	36 340	1 080	23 186
10 800	7 416	7	59 560	3 100	30 453

Table 1: Experimental results for computing all maximal Δ-cliques on the Thiers-Highschool dataset. $|R|$ is the size of the set returned by our algorithm, (i.e., the number of Δ-cliques found). For information, storing the dataset in RAM requires 51 MB.

We present in Figure 4, for each value of Δ, the complementary cumulative distributions for the size $|X|$ and duration $e - b$ of all maximal Δ-cliques $(X, [b, e])$. By definition, larger values of Δ trivially induce larger and longer Δ-cliques. Indeed, if $\Delta' > \Delta$ then every (maximal) Δ-clique also is a Δ'-clique (not maximal in general). More intuitively, small values of Δ detect local bursts, but are unable to find periodic behaviors if the period is larger than Δ. Notice

[1] A Debian machine with a 2.9 GHz CPU and 64 GB of RAM.

that when Δ grows the number of maximal Δ-cliques generally decreases, but this is not always true, as seen in Table 1. For an example of how the impact of Δ on the number of maximal Δ-cliques is not trivial, consider the stream presented in Figure 1: it contains four maximal 1-cliques, six maximal 2-cliques, and four maximal 3-cliques.

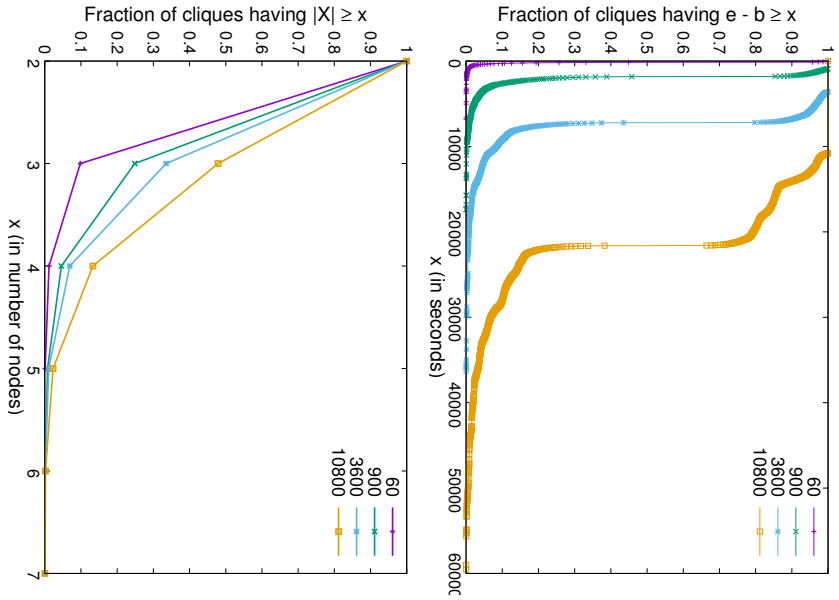

Figure 4: **Left:** complementary cumulative distribution of Δ-clique sizes for different values of Δ. **Right:** complementary cumulative distribution of Δ-clique durations for different values of Δ. The sharp drop at $2 \cdot \Delta$ is due to Δ-cliques involving only one link.

Notice now that Algorithm 1 makes no assumption on the order in which elements of S are processed, which corresponds to the way we explore the configuration space. In particular, if S is a first-in-first-out structure (a queue), the algorithm performs a BFS of the configuration space; if it is a last-in-first-out structure (a stack) then it performs a DFS. The execution time is essentially the same in all cases. The size of S may vary, but the space complexity of the algorithm is dominated by the size of M, that does not change. Still, the data structure impacts the order in which Δ-cliques are found.

We illustrate this in the practical case where $\Delta = 3600$ seconds (1 hour), see Figure 5. It shows that DFS rapidly discovers many cliques, and that those cliques are non-trivial cliques (cliques involving more than 2 nodes or lasting a substantial amount of time). In this case, using a DFS is therefore more interesting than a BFS, as it outputs results and exhibits non-trivial Δ-cliques faster. However, this behavior is dependent on the dataset, and deciding on the most appropriate exploration strategy in a given case remains an open question.

Consider now $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the graph induced by link stream $L = (T, V, E)$:

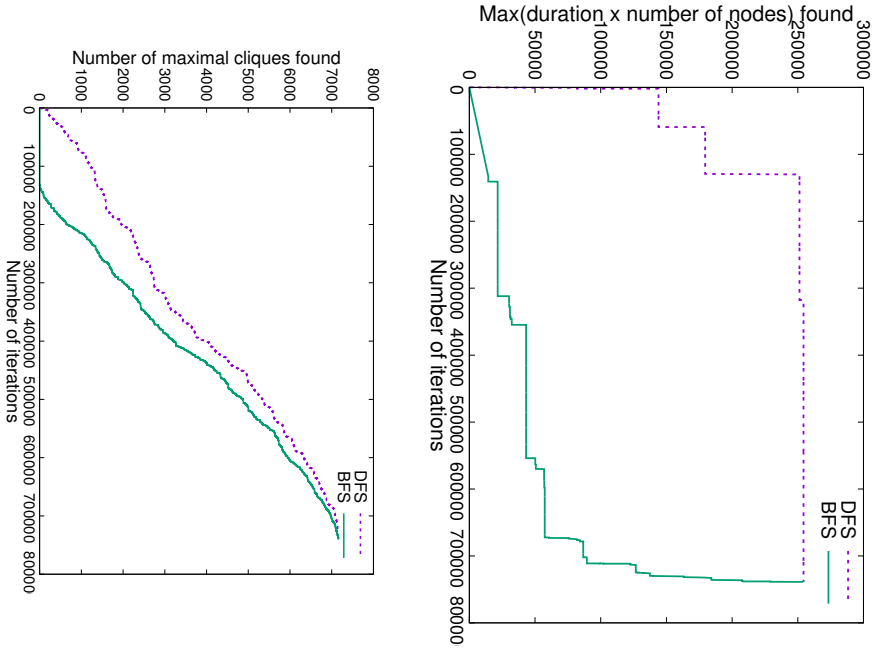

Figure 5: Behavior of our algorithm depending on the way it explores its configuration space (DFS or BFS). **Left**: number of maximal cliques discovered as a function of the number of iterations of the main loop of the algorithm. **Right**: maximal size of discovered cliques as a function of the number of iterations of the main loop of the algorithm. A clique size is estimated here by its number of nodes times its duration (in seconds).

$\mathcal{V} = V$ and $\mathcal{E} = \{(u, v) : \exists (t, u, v) \in E\}$. In other words, this is the graph where a link exists between two nodes u and v if and only if there is at least one contact between u and v in the link stream. The graph $\mathcal{G}$ contains 1742 cliques, the largest one involving 14 nodes. Approximately 70% of them involve students in the same class.

If $(X, [b, e])$ is a (maximal) Δ-clique of L, then by definition X is a clique of $\mathcal{G}$ (in general not maximal). However, as Δ-cliques capture time information they shed light on different patterns. For instance, L contains a 60-clique involving 4 students of different classrooms during roughly 5 minutes, which is likely to be the signature of a coffee break. Such Δ-cliques are non-trivial outputs of our algorithm, but they are invisible when considering graph cliques.

4. Conclusion

We introduced the notion of Δ-cliques in link streams, and proposed the first algorithm to compute the maximal such Δ-cliques. We implemented this algorithm and detected interesting Δ-cliques in real-world data.

Clearly, our algorithm may be improved further. Trying to adapt the Bron-Kerbosch algorithm [1] and some of its variants [11, 8, 3, 4], the most widely

used algorithms for computing cliques in graphs, is particularly appealing. Indeed, the configuration spaces built by these algorithms are trees, which avoids redundant computations. This is achieved by maintaining a set of candidate nodes that may be added to previously discovered cliques, which does not directly translate to our situation because of time in link streams. Still, we believe that progress is possible in this direction.

We also consider the case of links with duration as a promising perspective: each link (b, e, u, v) means that u and v interact from time b to e. In this case there is no need for a Δ anymore, as density in this context is nothing but the probability that two randomly chosen nodes are linked together at a randomly chosen time. The definition of cliques in link streams with durations follows directly, and our algorithm may be extended to compute maximal such cliques.

Acknowledgments.

We warmly thank the anonymous reviewers, who helped us improve this paper much. This work is supported in part by the French *Direction Générale de l'Armement* (DGA), by the Thales company, by the CODDDE ANR-13-CORD-0017-01 grant from the *Agence Nationale de la Recherche*, and by grant O18062-44430 of the French program *PIA – Usages, services et contenus innovants.*

References

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

[2] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. In *Ad-hoc, Mobile, and Wireless Networks*, volume 6811 of *Lecture Notes in Computer Science*, pages 346–359. 2011.

[3] F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1–3):564 – 568, 2008.

[4] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics*, 18:3.1:3.1–3.1:3.21, 2013.

[5] J. Fournet and A. Barrat. Contact patterns among high school students. *PLoS ONE*, 9:e107878, 2014.

[6] P. Holme and J. Saramäki. Temporal networks. *Physics Reports*, 519:97–125, 2012.

[7] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 3 1988.

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

[9] R. Rowe, G. Creamer, S. Hershkop, and S. J. Stolfo. Automated social hierarchy detection through email network analysis. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, WebKDD/SNA-KDD '07, pages 109–117, New York, NY, USA, 2007. ACM.

[10] R. Samudrala and J. Moult. A graph-theoretic algorithm for comparative modeling of protein structure. *Journal of Molecular Biology*, 279(1):287 – 302, 1998.

[11] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363:28–42, 2006.

[12] T. Viard and M. Latapy. Identifying roles in an IP network with temporal and structural density. In *Computer Communications Workshops (INFOCOM WKSHPS)*, pages 801–806, 2014.

[13] T. Viard and M. Latapy. Source code in python for computing cliques in link streams: `https://github.com/TiphaineV/delta-cliques`, 2014.

[14] K. Wehmuth, A. Ziviani, and E. Fleury. A Unifying Model for Representing Time-Varying Graphs. Research Report RR-8466, ENS Lyon, 2014.