Computing maximal cliques in link streams

Tiphaine Viard, Matthieu Latapy, Clémence Magnien

Computing maximal cliques in link streams

Jordan Viard, Matthieu Latapy, Clémence Magnien

Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
CNRS, UMR 7606, LIP6, F-75005, Paris, France

Abstract

A link stream is a sequence 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 every Δ during this time interval. We propose an algorithm to enumerate all maximal 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.

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 [3, 11, 7, 9].

For a given 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 for all $u \in X$, $v \in X$, and $\tau \in [b, e - \Delta]$, there is a link (t, u, v) in E with $t \in [\tau, \tau + \Delta]$.

More intuitively, all nodes in X interact at least once with each other at least every Δ from time b to time e. Clique C is maximal if it is included in no other clique, *i.e.* there exists no clique $C' = (X', [b', e']))$ such that $X' \subset X$ or $[b', e'] \subset [b, e]$. 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

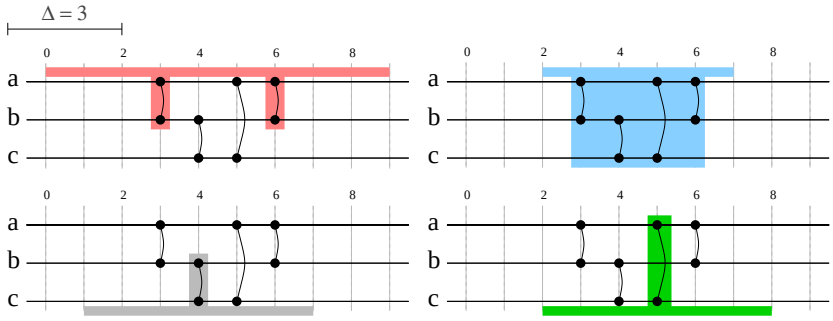

Figure 1: Examples of Δ-cliques. We consider the link stream $L = ([0, 9], \{a, b, c\}, E)$ with $E = ((3, a, b), (4, b, c), (5, a, c), (6, a, b))$ and $\Delta = 3$. There are four maximal 3-cliques in L: $(\{a, b\}, [0, 9])$ (top left), $(\{a, b, c\}, [2, 7])$ (top right), $(\{b, c\}, [1, 7])$ (bottom 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 [9]. Therefore, Δ-cliques in link streams are natural generalizations of cliques in graphs.

In this paper, we propose a 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 $(\forall v \in V,\ \exists (t, u, v) \in E)$. Finally, we define $\perp$ (resp. $\top$) as the occurrence time of the first (resp last) link in E: $\perp = \min\{t,\ \exists (t, u, v) \in E\}$ and $\top = \max\{t,\ \exists (t, u, v) \in E\}$. If $T = [\alpha, \omega]$, then we assume without loss of generality that $\perp > \alpha + \Delta$ and $\top < \omega - \Delta$.

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

2. Algorithm

One may trivially enumerate all maximal cliques in a graph as follows. One maintains a set S of previously found cliques (maximal or not). Then for each C in S, one removes C from S and searches for nodes outside C connected to all nodes in C, thus obtaining new cliques (one for each such node) larger than C. If one finds no such node, then C is maximal and it is part of the output. Otherwise, one adds the newly found cliques to S. The set S is initialized with the trivial cliques containing only one node, and all maximal cliques have been found when S is empty.

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 found Δ-cliques with the trivial Δ-cliques $(\{a, b\}, [t, t])$ for all (t, a, b) in L (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 8). We also look for values $b' < b$ such that $(X, [b', e])$ is a Δ-clique (lines 9 to 15), and likewise values $e' > e$ such that $(X, [b, e'])$ is a Δ-clique (lines 16 to 22). If we find such a node, such a b' or such an e', then C is not maximal and we add to S the new cliques larger than C we just found (lines 8, 15 and 22). Otherwise, C is maximal and it 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 ← ∅
2:  for (t, u, v) ∈ E: add ({u, v}, [t, t]) to S
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:              add (X ∪ {v}, [b, e]) to S and set isMax to False
9:      f ← max_{u,v∈X} f_{buv}          ▷ latest first occurrence time of a link in (X, [b, e])
10:     if b ≠ f − Δ then
11:         if ∃(t, u, v) ∈ E, f − Δ ≤ t < b and {u, v} ∩ X ≠ ∅ then
12:             let b' be the maximal such t
13:         else
14:             let b' be f − Δ
15:         add (X, [b', e]) to S and set isMax to False
16:     l ← min_{u,v∈X} l_{euv}          ▷ earliest last occurrence time of a link in (X, [b, e])
17:     if e ≠ l + Δ then
18:         if ∃(t, u, v) ∈ E, e < t ≤ l + Δ and {u, v} ∩ X ≠ ∅ then
19:             let e' be the minimal such t
20:         else
21:             let e' be l + Δ
22:         add (X, [b, e']) to S and set isMax to False
23:     if isMax then
24:         add (X, [b, e]) to R
25: return R
```

As time is a continuous quantity, finding appropriate values for b' and e' above is non-trivial. Let us explain the choice of b' (lines 10 to 15 in details, the choice of e' (lines 17 to 22) being symmetrical. Intuitively, we choose b' as small as possible, provided we do not miss any maximal Δ-clique. Therefore, for a given Δ-clique $(X, [b, e])$, we set b' to the latest time a link involving a node in X occurred before b: this link may make it possible to add a node to the

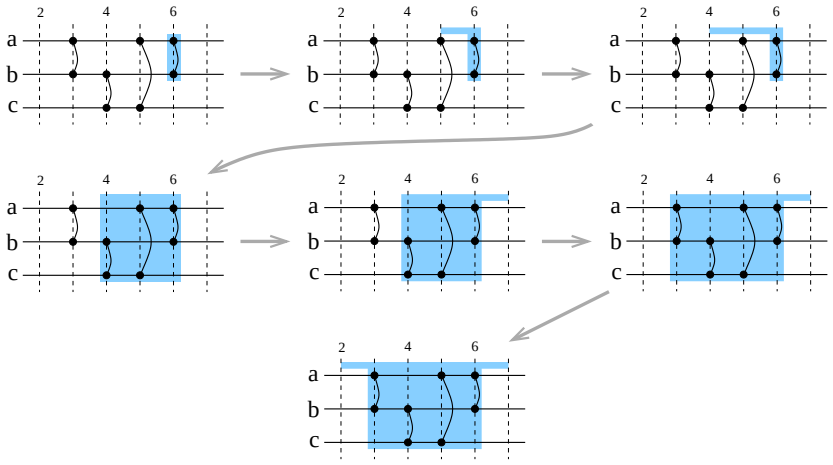

Figure 2: A sequence of Δ-cliques built by our algorithm to find a maximal Δ-clique (bottom row) from an initial trivial Δ-clique (top-left) in the link stream of Figure 1 when $\Delta = 3$. From left to right and top to bottom: the algorithm starts with $(\{a, b\}, [6, 6])$, and finds $(\{a, b\}, [5, 6])$ thanks to lines 9 to 15 of the algorithm. It then finds $(\{a, b\}, [4, 6])$ with the same lines, and $(\{a, b, c\}, [4, 6])$ thanks to lines 6 to 8. It finds $(\{a, b, c\}, [4, 7])$ from lines 16 to 22, $(\{a, b, c\}, [3, 7])$ from lines 9 to 15, and finally $(\{a, b, c\}, [2, 7])$ from the same lines.

Δ-clique. We also have to ensure that the obtained object remains a Δ-clique after the transformation. This leads to the two constraints of lines 10 and 17: f is the latest of the first occurrence times of all links in the clique. If it is equal to $b + \Delta$ (line 10) then there is no $b' < b$ such that $(X, [b', e])$ is a Δ-clique. If it is different, then such a b' exists, and we choose it in lines 11–15: we search for the latest link before b that involves a node in X; if it occurs after $f - \Delta$ then we choose b' as the time of this link; otherwise we set b' to $f - \Delta$, which is the smallest possible value such that $(X, [b', e])$ is a clique.

We display in Figure 2 an example of a sequence of such operations from an initial trivial clique to a maximal 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.

To prove the validity of Algorithm 1, we must show that all the elements it outputs are cliques, that they are maximal, and that all maximal 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 S only contains Δ-cliques at the i-th iteration of the loop (induction hypothesis). The loop may add new elements to S at lines 15, 22 and 8.

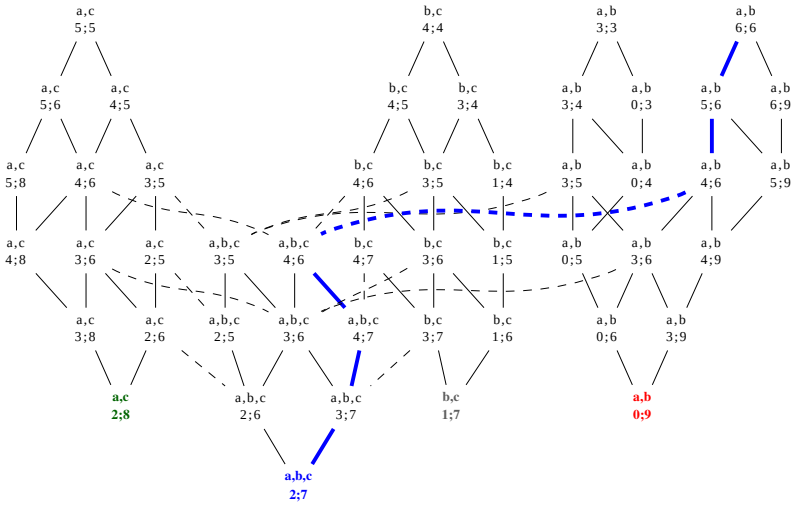

Figure 3: The configuration space built by our algorithm from the link stream of Figures 1 and 2 when $\Delta = 3$. Each element is a Δ-clique and it is linked to the Δ-cliques the algorithm builds from it (links are implicitely directed from top to bottom). Plain links indicate Δ-cliques discovered by lines 9 to 15 or lines 16 to 22 of the algorithm, which change the time span of the clique. Dotted links indicate Δ-cliques discovered by lines 6 to 8, 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.

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.

Let us show that $(X, [b, l + \Delta])$, where l is computed in line 16, 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 time of a link in C, for all u' and v' in X there is necessarily a link (t, u', v') in L 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 and since $e < e' \leq l + \Delta$, the element $(X, [b, e'])$ added to S at line 22, is a Δ-clique.

The same arguments hold for line 15, and it is trivial (from the test at line 7) that line 8 only adds Δ-cliques.

Finally, at the end of the $(i + 1)$-th iteration of the loop, S only contains Δ-cliques, which ends the induction. □

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 it is a Δ-clique. Assume it is not maximal; then we are in one of the three following situations.

There exists v in $V \setminus X$ 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 16, 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 17 of the algorithm is satisfied, and line 22 sets the boolean *isMax* to *false*. Like above, we reach a contradiction.

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

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

Lemma 3. *All maximal Δ-cliques of L are in the set returned by Algorithm 1.*

Proof. It is easy to see that if S contains a maximal Δ-clique then it is added to the set R returned by the algorithm, and so we show that all maximal Δ-cliques are in S at some stage. To do so, first notice that each Δ-clique A added to S in the *while* loop of our algorithm (lines 3 to 24) is built from a Δ-clique B taken from the set S at line 4. We denote this by $B \to A$.

Now, let us consider any maximal Δ-clique $C = (Y, [x, y])$, and let us denote by $x' \geq x$ (resp. $y' \leq y$) the first (resp. last) occurrence time of a link involving an element of Y after x (resp. before y). Notice that this link is not necessarily in C, as the other involved node may be outside Y. We define $C' = (Y, [x', y])$ and $C'' = (Y, [x', y'])$.

If $x' = x$ then trivially $C' = C$. If $x \neq x'$ then we show that $C' \to C$. Assume C' is the element taken from S at line 4: $X = Y$, $b = x' > x$ and $e = y$. As C is maximal, there exist u and v in Y such that the first occurrence of (u, v) in C is $(x+\Delta, u, v)$ (otherwise, there would be an ϵ such that $(Y, [x-\epsilon, y])$ would be a Δ-clique, and so C would not be maximal). In addition, for all u and v in Y, $x \leq f_{xuv} \leq x + \Delta$ (otherwise C would not be a Δ-clique). Since there is no link involving nodes in $X = Y$ from x to $x' = b$, the value of f computed in line 9 is $f = x + \Delta$. Since $x' \neq x$, we have $f = x + \Delta \neq x' + \Delta = b + \Delta$ and condition of line 10 holds. Now, by definition of x', there is no value of t, $x = f - \Delta \leq t < b = x'$, such that a there is a link (t, u, v) with u or v in $X = Y$. The condition in line 11 is therefore not satisfied, and line 14 sets b' to $f - \Delta = x$; the Δ-clique added to S at line 22 is nothing but $(X, [b', e]) = (Y, [x, y]) = C$, hence $C' \to C$.

Similarily, $C'' = C'$ or $C'' \to C'$. Therefore, if C'' is in S at some stage, then C' and C also will.

In order to show that C'' is in S at some stage, we build a sequence of Δ-cliques C_n, C_{n-1}, …, C_0 such that $C_n = C''$, and $C_0 = (\{u, v\}, [t, t])$ with $(t, u, v) \in E$. We build this sequence in order to ensure that for all i, $C_i \to C_{i+1}$. As C_0 is in S from line 2, this ensures that $C'' = C_n$ is finally added to S, and so C also is, thus ending the proof.

For the sake of the proof, we will in addition show that the following property P_i holds for all i: $C_i = (X_i, [b_i, e_i])$ is a Δ-clique and there is a link involving an element of X_i at time b_i and at time e_i. Since $C_n = C''$, property P_n is true from the construction above. We will show that, for all i, if it is true for C_i then it is also true for C_{i-1}.

Let us consider $C_i = (X_i, [b_i, e_i])$. We distinguish two main cases.

If $b_i = e_i$ and $|X_i| = 2$ then C_i is of the form $(\{x, y\}, [t, t])$, and so $C_i = C_0$, which ends the proof. If $b_i = e_i$ and $|X_i| \neq 2$ then we define C_{i-1} as $(X_i \setminus \{x\}, [b_i, e_i])$ for any x in X_i; it is trivial to check that $C_{i-1} \to C_i$ from lines 6 to 8, and that P_{i-1} is true.

If $b_i \neq e_i$, then we distinguish the following three sub-cases.

- If there exists x and y in X_i such that $(b_i, x, y) \in E$ and if $|X_i| > 2$, then we define C_{i-1} as $(X_i \setminus \{x\}, [b_i, e_i])$ if there is no x' such that $(e_i, x, x') \in E$, and as $(X_i \setminus \{y\}, [b_i, e_i])$ otherwise.

- If there exists x and y in X_i such that $(b_i, x, y) \in E$ as above, but now $|X_i| = 2$, we define C_{i-1} as $(X_i, [b_i, z])$ where z is the largest $z < e_i$ such that there is a link $(z, u, v) \in E$ with u or v in X_i.

- Otherwise (there is no link (b_i, x, y) with x and y in X_i), then we define C_{i-1} as $(X_i, [a, e_i])$ where a is the smallest $a > b_i$ such that there is a link $(a, x, y) \in E$ with x or y in X_i.

We will show for each case that $C_{i-1} \to C_i$ and that P_{i-1} is true. To do so, let us assume that C_{i-1} is the element taken from S at line 4 of the algorithm: $X = X_{i-1}$, $b = b_{i-1}$ and $e = e_{i-1}$.

In the first case, it is trivial to check that $C_{i-1} \to C_i$ from lines 6 to 8. Let us show that P_{i-1} is true. Clearly, C_{i-1} is a Δ-clique. Since only one of the vertices x or y is removed, the link $(b_i = b_{i-1}, x, y)$ is such that x or y is in X_{i-1}. If there is no x' such that $(e_i, x, x') \in E$ then because of P_i there exists some link (e_i, u, v) such that u or v is in $X_i \setminus \{x\} = X_{i-1}$, and therefore P_{i-1} holds. If such a x' exists, then the link $(e_i = e_{i-1}, x, x')$ involved a node of X_{i-1} and P_{i-1} also holds.

In the second case, P_{i-1} trivially holds and we denote by x and y the two elements of X_i. We have $X = X_{i-1} = X_i = \{x, y\}$, $b = b_{i-1} = b_i$, and $e = e_{i-1} = z$. The value of l computed in line 16 is the last occurrence time of (x, y) before $e = e_{i-1} = z$.

We first show that the condition in line 17 is satisfied, *i.e.* $e \neq l + \Delta$. Let us show that there exists a t, $e_i - \Delta \leq t < e_i$, such that $(t, x, y) \in E$: if $e_i - b_i \leq \Delta$ then $t = b_i$ satisfies this property; if $e_i - b_i > \Delta$, then if the property is not satisfied then there is an ϵ such that the interval $[e_i - \epsilon, e_i - \Delta - \epsilon]$ contains no occurrence of a link between x and y, which contradicts the fact that C_i is a Δ-clique. Moreover, by definition of z there is no link involving x or y from $z = e_{i-1} = e$ to e_i, and so there exists a t, $e_i - \Delta \leq t \leq e_{i-1}$, such that $(t, x, y) \in E$. Finally, l is the last occurrence time of (x, y) before e_{i-1} and

therefore we have $e_i - \Delta \leq l \leq e_{i-1}$. As $z < e_i$, we have $l > z - \Delta$ and therefore $l \neq e - \Delta$.

We now show that the condition of line 18 also is satisfied. From P_i, there exists a link (e_i, u, v) such that u or v is in X. Let us take $t = e_i$. We then have $e = e_{i-1} = z < e_i = t$ and we have shown above that $l \geq e_i - \Delta = t - \Delta$, therefore condition of line 18 is satisfied.

In addition, by definition of z there is no occurrence of a link involving x or y strictly before e_i and after $e = e_{i-1} = z$. Therefore, line 19 sets $e' = t = e_i$, and the Δ-clique added at line 22 is $(X, [b, e']) = (X_i, [b_i, e_i]) = C_i$, hence $C_{i-1} \to C_i$.

In the third case, it is also trivial that P_{i-1} holds (just notice that C_{i-1} necessarily is a Δ-clique as there is no link between elements of $X_i = X_{i-1}$ at b_i and between b_i and b_{i-1}). We have $X = X_{i-1} = X_i$, $b = b_{i-1} = a > b_i$, and $e = e_{i-1} = e_i$.

We first show that the condition in line 10 is satisfied. As C_i is a Δ-clique, and as there is no link between two nodes in X_i at b_i, then there must exist a link for all pairs of nodes in X_i at a time t in $]b_i, \min(b_i + \Delta, e_i)]$. There is no link involving nodes of X_i before a, by definition of a, and so t necessarily is in $[a, \min(b_i + \Delta, e_i)]$. The value of f computed in line 9 is the latest first occurrence time after $b = b_{i-1} = a$ of a link between two nodes of $X = X_{i-1} = X_i$. Therefore, f also is in $[a, \min(b_i + \Delta, e_i)]$. Therefore, $f \leq b_i + \Delta < a + \Delta = b + \Delta$ and so $f - \Delta < b$.

In addition, similarly to condition of line 18 in the previous case, condition of line 11 is satisfied when $t = b_i$. This ultimately leads to $C_{i-1} \to C_i$, which ends the proof.

□

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.*

Enumerating maximal cliques in graph $G = (V, E)$ is equivalent to enumerating maximal Δ-cliques in $L = ([0, 0], V, E')$ where $(0, u, v) \in E'$ if and only if $(u, v) \in E$. Therefore, enumerating Δ-cliques in a link stream is exponential (in particular the number of Δ-cliques may be exponential). To this regard, our algorithm is optimal: the number of elements of the configuration space built by the algorithm is bounded by the number of subsets of the set of links times the number of subsets of the set of link arrival times, and the number of operations performed for each of them is polynomial.

Still, several optimisations may speed up our algorithm (without changing its worst-case complexity), and we discuss some now.

First, Algorithm 1 may build and add to S the same Δ-clique many times. To avoid redundancy of computations, one may store the Δ-cliques seen so far. Then, a Δ-clique is added to S only the first time it is seen. This ensures that the number of runnings of the main loop is the number of links in the configuration space; in the naive version presented above, this number is the

sum of the lengths of all paths from any initial trivial Δ-clique to the maximal ones it can reach.

Notice also that f and l, computed in lines 9 and 16, 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 $[b, e]$.

Going further, let us notice that 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 cliques C' added to S at lines 8, 15 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.

Finally, notice that Algorithm 1 makes no assumption on the order in which elements of S are processed. This order corresponds to the way we explore the configuration space. In particular, if S is a first-in-first-out structure (like a queue), the algorithm performs a BFS of the configuration space; if it is a last-in-first-out structure (like a stack) then it performs a DFS. The execution time is the same in all cases. The size of S may vary, but the space complexity of the algorithm is dominated by the size of the set of already seen cliques just discussed, that does not change. Still, different exploration methods have different advantages and drawbacks. A BFS rapidly discovers all maximal cliques of small sizes and durations, which makes it suitable for computing the clique size distribution, or if one is interested in discovering as many maximal cliques as possible as rapidly as possible, even if they are all small ones. Conversely, a DFS first discovers large cliques and so it is appealing if one searches for non-trivial cliques.

3. Experimentation

We implemented Algorithm 1 with all optimisations discussed above and provide the Python source code at [10]. We illustrate here its practical relevance by computing maximal Δ-cliques of a link stream representing real-world contacts between individuals, captured with sensors. This trace was collected at a french high school in 2012, see [6] for full details. It induces a link stream of 181 nodes and 45047 links, connecting 2220 distinct pairs of nodes over a period of 729500 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 used here $\Delta = 3600$ seconds $= 1$ hour. Therefore, a Δ-clique $C = (X, [b, e])$ means here that the individuals in set X were pairwise in contact at least once every hour from time b to time e.

Our Python implementation succeeds in finding all the 8212 maximal Δ-cliques in this link stream in a few hours on a standard computer. We display in Figure 4 the impact of the configuration space exploration strategy on the program behaviour, as discussed above. It shows clearly that, as expected, BFS

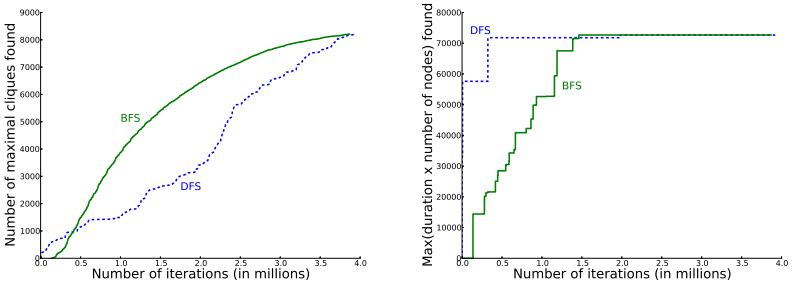

Figure 4: Behaviour of our algorithm depending on the way it explores its configuration space (DFS or BFS). **Left**: number of maximal cliques discovered as a function of the number of iterations of the main loop of the algorithm. **Right**: maximal size of discovered cliques as a function of the number 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).

rapidly discovers many small cliques, while DFS rapidly discovers non-trivial cliques. Although many discovered cliques are very small, we also find rather large and long cliques: they contain up to 7 individuals, and last up to 10 hours. Such structures are non-trivial, and worth studying in a dedicated work.

4. Conclusion

We introduced the notion of Δ-cliques in link streams, and proposed a first algorithm to compute the maximal such 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 [2] and some of its variants [8, 4, 1, 5], the most widely used algorithms for computing cliques in graphs, is particularly appealing. Indeed, the configuration spaces built by these algorithms are trees, and they are much smaller than our own configuration spaces. 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 are in contact 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.

This work is supported in part by the french *Direction Générale de l'Armement* (DGA), 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*.

[1] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996.

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

[3] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *CoRR*, abs/1012.0009, 2010.

[4] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal of Computing*, 14(1):210–223, 1985.

[5] David Eppstein and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *Experimental Algorithms*, pages 364–375, 2011.

[6] Julie Fournet and Alain Barrat. Contact patterns among high school students. *PLoS ONE*, 9:e107878, 2014.

[7] Petter Holme and Jari Saramäki. Temporal networks. *Physics Reports*, 519:97–125, 2012.

[8] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363:28–42, 2006.

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

[10] Jordan Viard and Matthieu Latapy. Source code in python for computing cliques in link streams: `https://github.com/jordanV/delta-cliques`, 2014.

[11] Klaus Wehmuth, Artur Ziviani, and Eric Fleury. A Unifying Model for Representing Time-Varying Graphs. Research Report RR-8466, ENS Lyon, 2014.