Enumerating Maximal Cliques in Temporal Graphs

Bachelorarbeit im Studiengang Informatik

an der Technischen Universität Berlin

vorgelegt von

Anne-Sophie Himmel

Matrikelnummer: 320296

Berlin, den 19. Januar 2016

Erstgutachter: Prof. Dr. Rolf Niedermeier

Zweitgutachterin: Prof. Dr. Sabine Glesner

Betreuer: Manuel Sorge, Hendrik Molter, Prof. Dr. Rolf Niedermeier

Zusammenfassung

In vielen komplexen Systemen spielt die Dynamik der Interaktionen zwischen den einzelnen Einheiten des Systems eine wichtige Rolle. Eine Modellierungsumgebung, die diese Dynamik erfasst, sind temporelle Graphen. Der Fokus wird auf der Erweiterung des Konzepts von Cliquen auf temporelle Graphen liegen: für eine gegebene Zeitperiode Δ, ist eine Δ-Clique eine Menge aus Knoten und ein Zeitintervall für die gilt, dass die Knoten spätestens nach allen Δ Zeiteinheiten innerhalb des gegebenen Zeitintervalls paarweise in Kontakt zueinander stehen. Eine Aufzählung aller maximaler Cliquen in temporellen Graphen wird bereits durch einen Greedyalgorithmus abgehandelt. Im Kontrast zu diesem Ansatz wird in dieser Arbeit eine Anpassung des Bron-Kerbosch Algorithmus—ein effizienter, rekursiver Backtracking-Algorithmus zur Aufzählung aller Cliquen in einfachen Graphen—auf das Problem angewandt. In Experimenten mit realen Datensätzen wird gezeigt, dass im Vergleich zu dem ersten Ansatz eine signifikante Verbesserung der Laufzeit erreicht wird.

Abstract

In many complex systems, interactive dynamics play a very important role in the analysis of such systems. A modeling framework to capture these dynamics are temporal graphs. Our main focus lies on the extension of the concept of cliques to temporal graphs: for a given time period Δ, a Δ-clique in temporal graphs is a set of vertices and a time interval, such that all vertices interact with each other at least after every Δ time steps within the time interval. A greedy algorithm which enumerates all maximal Δ-cliques in a temporal graph was already introduced. In contrast to this approach, this thesis applies the Bron-Kerbosch algorithm—an efficient, recursive backtracking algorithm which enumerates all maximal cliques in simple graphs—to this problem. Experiments on real-world datasets show that there is a significant improvement in the running time in comparison to the greedy algorithm approach.

Contents

1 Introduction

Many complex, real-world systems can be modeled as simple graphs. These consist of vertices representing the units in the system and edges representing the interactions between these units. This abstraction helps to reduce the complexity of the system and makes it more accessible for analysis. However, such a crude modeling framework is prone to lose important information, making the model less meaningful. One possibility to make these representations more expressive is by including additional levels of detail, a widely used extension to simple graph models is the use of edge weights [HS12]. In this work, however, we want to concentrate on another parameter worth studying: the time dimension. In a plethora of systems, dynamics play a very important role: social networks, communication networks, transportation networks, and epidemic networks are some of the fields worth mentioning. In these systems, the interactions among units are rarely persistent over time and the non-temporal interpretation is an "oversimplifying approximation" [Nic+13]. Consequently, whenever we deal with a networked system that evolves over time, the concept of interaction needs to be redefined appropriately. One approach of dealing with these dynamics is the use of so-called temporal graphs.

1.1 Temporal Graphs

A temporal graph can be informally described as a graph that changes over time. Temporal graphs—also referred to as time-varying graphs [Nic+13], temporal networks [HS12], or link streams [VLM15]—not only consider the edges between vertices but also the points in time when these edges occur during the observation time of the dynamic system. Literature commonly defines temporal graphs as a sequence of static graphs over a fixed set of vertices [EHK15; Mic15; MS14]. The definition that we want to use is essentially the same: instead of having a static graph for every point in time, the edges of the temporal graph are not only a set of two vertices indicating an interaction between these vertices but also consist of a time stamp that indicates the time of the interaction.

Definition 1.1. A *temporal graph* $\mathbb{G} = (T, V, E)$ is defined as a triple consisting of a time interval $T = [\alpha, \omega]$, where $\alpha, \omega \in \mathbb{N}$, $T \subseteq \mathbb{N}$ and $\omega - \alpha$ is the *lifetime* of the temporal graph $\mathbb{G}$, a set of vertices V, and a set of *time-edges* $E \subseteq \binom{V}{2} \times T$.

The notation $\binom{V}{2}$ describes the set of all possible undirected edges $\{v_1, v_2\}$ with $v_1 \neq v_2$ and $v_1, v_2 \in V$. A time-edge $e = (\{v_1, v_2\}, t) \in E$ can be interpreted as an interaction between v_1 and v_2 at time t. Note that we will restrict our attention to discretized time, implying that changes only occur at discrete points in time. This seems close to a natural abstraction of real-world dynamic systems and "gives the problems a purely combinatorial flavor" [MS14].

1.2 Δ-Cliques

Many concepts and problem definitions of static graphs can be easily adapted to temporal graphs. We want to have a look at a concept that has not received much attention in studies even though it seems to be a very conclusive concept in temporal graphs, especially in dynamic communication and social networks: the clique. A clique in a static graph is a subset of vertices in which all vertices are pairwise connected by an edge. We want to adapt this definition to temporal graphs. First off, we have to address the question of what a clique in temporal graphs is meant to express.

The result of simply examining consecutive points in time and determining which vertices form a clique at these points would hardly be meaningful: if the subject matter of examination is e-mail traffic and the dataset includes e-mails with time stamps accurate to the second, we are not interested in people who sent e-mails to each other every second over a certain time interval, but would like to know which groups of people were in contact with each other at least after every seven days over months. One possible approach would be to generalize the time stamps, taking into account only the week an e-mail was sent, resulting in a loss of accuracy in the dataset. In order to achieve our goal without forfeiting data accuracy, we have to set a period of time—let us call it Δ—in which we are looking for cliques in the original dataset, whilst entirely maintaining its accuracy. We can define a so-called Δ-clique as follows:

Definition 1.2. For a given time period $\Delta \in \mathbb{N}$, a Δ-*clique* in a temporal graph $\mathbb{G} = (T, V, E)$ is a tuple $C = (X, I = [a, b])$ with $X \subseteq V$ and $I \subseteq T$ such that for all $\tau \in [a, b - \Delta]$ and for all $v, w \in X$ with $v \neq w$ there exists $(\{v, w\}, t) \in E$ with $t \in [\tau, \tau + \Delta]$.

In other words, for a Δ-clique $C = (X, I)$ all pairs of vertices in X interact with each other at least after every Δ time units during the time interval I. We implicitly exclude Δ-cliques with time intervals smaller than Δ.

It is evident that the parameter Δ is a measurement of the intensity of interactions in Δ-cliques. Small Δ-values imply that the interaction between vertices in a Δ-clique has to be more frequent than in the case of large Δ-values. The choice of Δ depends on the dataset and the purpose of the analysis.

We can also consider Δ-cliques from another point of view. For a given temporal graph $\mathbb{G} = (T, V, E)$ and a $\Delta \in \mathbb{N}$, the static graph $G_\tau^\Delta = (V_\tau, E_\tau)$ describes all contacts that appear within the Δ-sized time window $[\tau, \tau + \Delta]$ with $\tau \in [\alpha, \omega - \Delta]$ in the temporal graph $\mathbb{G}$—that is $V_\tau = V$ and for every $\{v_1, v_2\} \in E_\tau$ it holds that a $t \in [\tau, \tau + \Delta]$ with $(\{v_1, v_2\}, t) \in E$ exists. The existence of a Δ-clique $C = (X, I = [a, b])$ indicates that all vertices in X form a clique in all static graphs G_τ^Δ with $\tau \in [a, b - \Delta]$. This implies that all vertices in X are pairwise connected to each other in the static graphs of all sliding, Δ-sized time windows from time a until $b - \Delta$.

A closer look at these definitions reveals that a Δ-clique always exists at least over a time interval of size Δ. Furthermore, by setting Δ to the length of the whole lifetime of the temporal graph, every Δ-clique corresponds to a normal clique in the underlying static graph that results from ignoring the time stamps of the time-edges. This implies that for every Δ-clique it holds that there is at least one interaction between the vertices during the whole lifetime of the graph.

We are not interested in all possible Δ-cliques of a temporal graph, but in the maximal ones: A Δ-clique X is maximal if it is not contained in any other Δ-clique. In other words, the time interval can not be increased nor can a vertex be added to the set without losing the properties of a Δ-clique. In this thesis, when we talk about Δ, we will always consider a fixed value $\Delta \in \mathbb{N}$.

1.3 Our Contribution

Viard, Latapy, and Magnien [VLM15] have already published an algorithm for finding maximal Δ-cliques in temporal graphs. In contrast to their approach, we adapt the idea of the Bron-Kerbosch algorithm [BK73]—an efficient, recursive algorithm for finding maximal cliques in static graphs [ELS13]—to the problem. We show that there is a notable improvement in the running time of the Bron-Kerbosch adaption in comparison to the first algorithm proposed for enumerating maximal Δ-cliques in temporal graphs on real-world datasets.

After introducing the main definitions and notations used in this thesis, we present the main idea of the original Bron-Kerbosch algorithm in Section 2. We explain the idea of pivoting: a procedure to reduce the recursive calls of the Bron-Kerbosch algorithm and shortly introduce degeneracy—a measure of sparsity in non-temporal graphs—and how this parameter can be exploited to upper-bound the running time of the Bron-Kerbosch algorithm. In Section 3, we propose an adaption of the Bron-Kerbosch algorithm to enumerate all maximal Δ-cliques in a temporal graph and prove the correctness of the algorithm. Furthermore, we adapt the idea of pivoting to the algorithm to reduce the number of recursive calls. In Section 4, we present the most important details of the implementation design and show the main results of the test runs on real-world datasets. We observe the efficiency of the algorithm and compare the running time to the algorithm introduced by Viard, Latapy, and Magnien [VLM15]. Additionally, we study the behavior of the algorithm for different Δ-values.

1.4 Preliminaries

In this subsection we sum up the most important notations and definitions used. We already mentioned the definition of temporal graphs:

Temporal Graph A *temporal graph* $\mathbb{G} = (T, V, E)$ is defined as a triple consisting of a time interval $T = [\alpha, \omega]$, where $\alpha, \omega \in \mathbb{N}$ and $\omega - \alpha$ is the *lifetime* of a temporal graph $\mathbb{G}$, a set of vertices V, and a set of *time-edges* $E \subseteq \binom{V}{2} \times T$.

Δ-Neighborhood We define a tuple (v, I) with $v \in V$ and $I \subseteq T$ as a *vertex-interval pair* of a temporal graph. We need this definition to adapt the idea of a neighborhood to temporal graphs with respect to a time period Δ: For a vertex $v \in V$ and a time interval $I \subseteq T$ in a temporal graph, we define the Δ-neighborhood $N^{\Delta}(v, I)$ as the set of all vertex-interval pairs $(w, I' = [a, b])$ with the property that for every $\tau \in [a, b - \Delta]$ at least one edge $(\{v, w\}, t) \in E$ with $t \in [\tau, \tau + \Delta]$ exists. Furthermore, it holds $b - a \geq \Delta$, $I' \subseteq I$ and I' is maximal—that is, a time interval $I'' \subseteq I$ with $I' \subseteq I''$ satisfying the properties above does not exist. The Δ-neighborhood of a vertex $v \in V$ in the time interval $I \subseteq T$ of a temporal graph contains every vertex w and every related time window $I' \subseteq I$ in which there is a contact between v and w at least after every Δ time units—we call w a Δ-*neighbor* of v during the time I'. In Figure 1, we visualize the concept of Δ-neighborhood in a temporal graph.

Δ-Cut In this thesis, we want to work with this neighborhood to find all maximal Δ-cliques in a temporal graph. Therefore, we need to be able to find the intersection of two neighborhoods. However, by definition, two vertices can only be neighbors during a time interval of at least the size Δ. Therefore, we introduce the following function: Let X and Y be sets of vertex-interval pairs. The Δ-*cut* $X \sqcap Y$ includes all $(v, I = [a, b])$ so that $b - a \geq \Delta$ and I is maximal with respect to inclusion among all intervals J that fulfill the existence of I' and I'' with $(v, I') \in X$ and $(v, I'') \in Y$ and $J \subseteq I'$ and $J \subseteq I''$.

We will need two more relations concerning sets of vertex-time pairs in the later stages of the thesis. Let X and Y be sets of vertex-interval pairs. The relation $(v, I) \sqsubseteq X$ expresses that a vertex-interval pair $(v, I') \in X$ with $I \subseteq I'$ exists. The relation $X \sqsubseteq Y$ expresses that for every vertex-time pair $(v, I) \in X$ it holds that $(v, I) \sqsubseteq Y$.

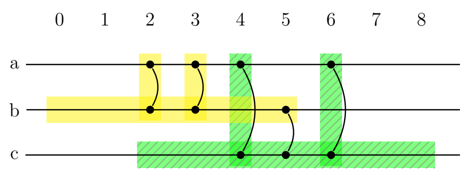
Δ-Clique Finally, let us shortly recap the definition of Δ-cliques and formalize the definition of maximal Δ-cliques: A Δ-clique in a temporal graph $\mathbb{G} = (T, V, E)$ is a tuple $C = (X, I = [a, b])$ with $X \subseteq V$ and $I \subseteq T$ such that for all $\tau \in [a, b - \Delta]$ and for all vertices $v, w \in X$ with $v \neq w$ there is at least one edge $(\{v, w\}, t) \in E$ with $t \in [\tau, \tau + \Delta]$. In other words, for all vertices $v, w \in X$ it holds that v is a Δ-neighbor of w during I.

A Δ-clique C is maximal if it is not *contained* in any other Δ-clique. This implies that there is neither a vertex $v \in V \setminus X$ so that $(X \cup \{v\}, I)$ is a Δ-clique nor an interval $I' \subseteq T$ with $I \subset I'$ so that (X, I') is a Δ-clique—in short, the Δ-clique is *vertex-maximal* and *time-maximal*.

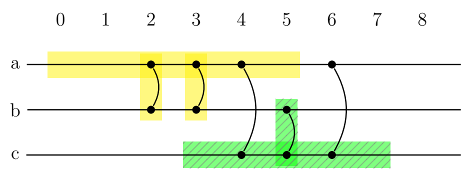
Example 1. In Figure 1 we visualize a temporal graph and the concept of Δ-neighborhood and Δ-clique. We consider a temporal graph $\mathbb{G} = (T, V, E)$ with $T = [0, 8]$, $V = \{a, b, c\}$ and $E = \{(\{a, b\}, 2), (\{a, b\}, 3), (\{a, c\}, 4), (\{b, c\}, 5), (\{a, c\}, 6)\}$ and $\Delta = 2$. The edges between two vertices at a specific time represent the time-edges of the temporal graph.

We visualize the Δ-neighborhood of each vertex of the temporal graph over the whole time interval T in Figure 1(a)-1(c):

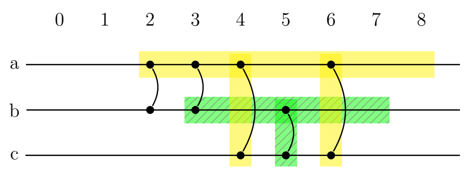
- In Figure 1(a), we consider the Δ-neighborhood of vertex a during the whole time interval T—$N^{\Delta}(a, T)$. The yellow bar marks the vertex-interval pair $(b, [0, 5]) \in N^{\Delta}(a, T)$. The vertex b is a Δ-neighbor of a during $[0, 5]$ because for every $\tau \in [0, 5 - \Delta = 3]$ at least one time-edge $(\{a, b\}, t) \in E$ with $t \in [\tau, \tau + \Delta]$ exists since $(\{a, b\}, 2), (\{a, b\}, 3) \in E$. The same holds for the vertex-interval pair $(c, [2, 8]) \in N^{\Delta}(a, T)$ which is marked in green (hatched).



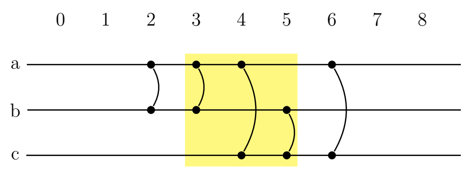
(a) $N^{\Delta}(a, T)$



(b) $N^{\Delta}(b, T)$



(c) $N^{\Delta}(c, T)$



(d) maximal Δ-clique $(\{a, b, c\}, [3, 5])$

Figure 1: Temporal graph, Δ-neighborhood, and Δ-clique (Example 1)

- In Figure 1(b), we visualize the Δ-neighborhood of b over the whole lifetime T of the temporal graph—$N^{\Delta}(b, T)$. The vertex-interval pair $(c, [3, 7]) \in N^{\Delta}(b, T)$ is marked in green (hatched). The vertex-interval pair $(a, [0, 5]) \in N^{\Delta}(b, T)$ is marked in yellow. It becomes evident that being a Δ-neighbor of another vertex is a symmetric relation—if a is a Δ-neighbor of b during $[0, 5]$, then b is also a Δ-neighbor of a during $[0, 5]$.

- In Figure 1(c), we visualize the Δ-neighborhood of c over the whole lifetime T of the temporal graph—$N^{\Delta}(c, T)$. The vertex-interval pair $(b, [3, 7]) \in N^{\Delta}(c, T)$ is marked in green (hatched). The vertex-interval pair $(a, [2, 8]) \in N^{\Delta}(c, T)$ is marked in yellow.

Figure 1(d) shows the maximal Δ-clique $(\{a, b, c\}, [3, 5])$. During the time interval $[3, 5]$, a and b are Δ-neighbors, b and c are Δ-neighbors and a and c are Δ-neighbors which can be observed in Figure 1(a)-1(c). We cannot increase the time interval because at time step 2 the vertices b and c are not yet Δ-neighbors and at time step 6 the vertices a and b are no longer Δ-neighbors respectively. Further maximal Δ-cliques in this temporal graph are: $(\{a, b\}, [0, 5]), (\{a, c\}, [2, 8]), (\{b, c\}, [3, 7])$ as well as the trivial Δ-cliques $(\{a\}, [0, 8]), (\{b\}, [0, 8]), (\{c\}, [0, 8])$.

In the next section, we present the main idea of the Bron-Kerbosch algorithm and the main improvements of the algorithm. After that, we generalize these ideas and apply them to temporal graphs.

Algorithm 1 Enumerating all maximal cliques in a graph

```
function BronKerbosch($P, R, X$)
    if $P \cup X = \emptyset$ then
        add $R$ as maximal clique to the solution
    end if
    for $v \in P$ do
        BronKerbosch($P \cap N(v), R \cup \{v\}, X \cap N(v)$)
        $P \leftarrow P \setminus \{v\}$
        $X \leftarrow X \cup \{v\}$
    end for
end function
```

2 Bron-Kerbosch Algorithm

The Bron-Kerbosch algorithm [BK73] is a widely used algorithm for finding all maximal cliques in undirected, static graphs. It is a recursive backtracking algorithm, which is easy to implement and has often shown to be more efficient in practice than alternative algorithms [ES11]. In this section, we explain the basic idea of the Bron-Kerbosch algorithm. Furthermore, we present two techniques to improve the running time of the algorithm.

The Bron-Kerbosch algorithm——see Algorithm 1——receives three disjoint vertex-sets as input parameters: P, R, and X. The set R is a clique and $P \cup X$ is the set of all vertices which are adjacent to every vertex in R. Each vertex in $P \cup X$ is a witness that the clique R is not maximal yet. The set P represents the vertices that have not been considered yet whereas the set X includes all vertices that have already been taken into account in earlier steps. This set is used to avoid enumerating maximal cliques more than once. In each call, the algorithm checks whether the given clique R is maximal or not. If $P \cup X = \emptyset$, then there are no vertices that can be added to the clique. Therefore, the clique is maximal and can be added to the solution. Otherwise, the clique is not maximal because a vertex exists that is adjacent to all vertices in R and consequently would form a clique with R. For each $v \in P$ the algorithm makes a recursive call for the clique $R \cup \{v\}$ and restricts P and X to the neighborhood of v. After the recursive call, the vertex v is removed from P and added to X. This guarantees that the same maximal cliques are not detected multiple times. For a graph $G = (V, E)$ the algorithm will be initially called with $P = V$ and $R = X = \emptyset$.

2.1 Pivoting

Bron and Kerbosch [BK73] also introduced an optimization for their basic algorithm by choosing a pivot element to decrease the number of recursive calls and therefore enable faster backtracking.

The idea is based on the observation that for any vertex $u \in P \cup X$—the pivot element—either u itself or one of its non-neighbors must be contained in any maximal clique containing R. If neither u nor one of the non-neighbors of u are included in a

Algorithm 2 Enumerating all maximal cliques in a graph with pivoting

```
function BronKerboschPivot($P, R, X$)
    if $P \cup X = \emptyset$ then
        add $R$ as maximal clique to the solution
    end if
    choose pivot vertex $u \in P \cup X$ with $|P \cap N(u)| = \max_{v \in P \cup X} | P \cap N(v) |$
    for $v \in P \setminus N(u)$ do
        BronKerboschPivot($P \cap N(v), R \cup \{v\}, X \cap N(v)$)
        $P \leftarrow P \setminus \{v\}$
        $X \leftarrow X \cup \{v\}$
    end for
end function
```

Algorithm 3 Enumerating all maximal cliques in a graph with degeneracy ordering

```
function BronKerboschDeg($P, R, X$)
    for $v_i$ in a degeneracy odering $v_0, v_1, \ldots, v_n$ of $G = (V, E)$ do
        $P \leftarrow N(v_i) \cap \{v_{i+1}, \ldots, v_{n-1}\}$
        $X \leftarrow N(v_i) \cap \{v_0, \ldots, v_{i-1}\}$
        BronKerboschPivot($P, \{v_i\}, X$)
    end for
end function
```

maximal clique containing R, this clique cannot be maximal because u can be added to this clique due to the fact that only neighbors of u were added to the original clique R.

Therefore, choosing an arbitrary pivot element $u \in P \cup X$ and iterating just over u and all its non-neighbors decreases the number of recursive calls in the for-loop that lead to non-maximal cliques. Tomita, Tanaka, and Takahashi [TTT06] have shown that in the case that u is chosen from $P \cup X$ in a way that u has the most neighbors in P—see Algorithm 2—then the running time for a graph $G = (V, E)$ is in $O(3^{|V|/3})$.

2.2 Degeneracy Ordering

A possibility to influence the running time and provide a fixed-parameter tractable modification of the basic Bron-Kerbosch algorithm is by exploiting the degeneracy of the graph, which is a measure of sparsity:

A graph G is called *k-degenerated* if every non-empty subgraph G' of G contains a vertex v with $\deg(v) \leq k$. The *degeneracy* of a graph G is defined as the smallest value k such that G is k-degenerated. If the degeneracy of a graph is d, then the maximal clique size of the graph is at most $d + 1$. In case there is a clique of the size of at least $d + 2$, the vertices of this clique would form a subgraph in which every vertex v of the clique has $deg(v) \geq d + 1$. For each k-degenerated graph there is a *degeneracy ordering*, such that for every vertex v there are at most k of its neighbors later on in the ordering. The degeneracy d and a corresponding degeneracy ordering for a graph $G = (V, E)$ can be computed efficiently in $O(|V| + |E|)$ time [ELS10]: For graph G, the vertex with the smallest degree is taken in each step and removed from the graph until no vertex is left. The degeneracy of the graph is the highest degree of a vertex at the time the vertex has been removed from the graph—a corresponding degeneracy ordering is the order in which the vertices were removed from the graph. For a graph $G = (V, E)$ with degeneracy d, using the degeneracy ordering of G in the outer-most recursive call and afterwards using pivoting—see Algorithm 3—the algorithm runs in $O(d \cdot |V| \cdot 3^{d/3})$ time according to Eppstein, Löffler, and Strash [ELS10].

In the preceding section, the basic idea of the Bron-Kerbosch algorithm to find maximal cliques in an undirected graph was introduced. In the following sections, an attempt is made to adapt the idea of the Bron-Kerbosch algorithm in order to find maximal Δ-cliques in temporal graphs. In addition, the idea of pivoting is adapted to the proposed algorithm.

Algorithm 4 Enumerating all maximal Δ-cliques in a temporal graph

```
1:  function BronKerboschDelta($P, P_{\max}, R = (C, I), X, X_{\max}$)
    ▷ $R = (C, I)$ : time-maximal Δ-clique
    ▷ $P \cup X$ : set of all $(v, I')$ that fulfill that $I' \subseteq I$ and $(C \cup \{v\}, I')$ is a time-maximal Δ-clique
    ▷ $P_{\max} \cup X_{\max}$ : set of all $(v, I)$ that fulfill that $(C \cup \{v\}, I)$ is a time-maximal Δ-clique
    ▷ if $P_{\max} = \emptyset$ and $X_{\max} = \emptyset$ then there exists no vertex that can be added to $C$ without downsizing the interval $I \rightsquigarrow R$ is vertex-maximal $\rightsquigarrow R$ is a maximal Δ-clique
2:      if $P_{\max} \cup X_{\max} = \emptyset$ then
3:          add $R$ as maximal clique to solution
4:      end if
5:      for $(v, I') \in P$ do
    ▷ create new time-maximal Δ-clique by adding $v$ to $C$ and downsizing the interval to $I' \subseteq I$
6:          $R' \leftarrow (C \cup \{v\}, I')$
    ▷ find all intersections between the sets $P, X$ and the Δ-neighborhood $N^{\Delta}(v, I')$
7:          $P' \leftarrow P \sqcap N^{\Delta}(v, I')$
8:          $X' \leftarrow X \sqcap N^{\Delta}(v, I')$
    ▷ find all $(w, I')$ in $P', X'$ such that $w$ can be added to $R'$ without downsizing the interval $I'$
9:          $P'_{\max} \leftarrow \{(w, I') \mid (w, I') \in P'\}$
10:         $X'_{\max} \leftarrow \{(w, I') \mid (w, I') \in X'\}$
    ▷ check new clique $R'$ with the adapted sets $P', P'_{\max}, X'$, and $X'_{\max}$
11:         BronKerboschDelta($P', P'_{\max}, R', X', X'_{\max}$)
    ▷ remove the vertex-time pair that created the new clique $R'$ from set $P$ and add it to set $X$ to avoid multiple enumerations of cliques generated from $R$.
12:         $P \leftarrow P \setminus \{(v, I')\}$
13:         $X \leftarrow X \cup \{(v, I')\}$
14:     end for
15: end function
```

3 Bron-Kerbosch Algorithm in Temporal Graphs

The goal is to design an algorithm that enumerates all maximal Δ-cliques in temporal graphs. Due to the procedural similarities of finding maximal cliques in static graphs and finding Δ-cliques in temporal graphs, it seems natural to try to adapt the Bron-Kerbosch algorithm—see Algorithm 1. The algorithm should not only enumerate all maximal Δ-cliques but also maintain the main characteristic of the basic Bron-Kerbosch algorithm: it should enumerate each maximal Δ-clique exactly once. The following theorem for our adaption of the Bron-Kerbosch algorithm—Algorithm 4, which will be introduced subsequently—will be proven later on in this thesis.

Theorem 3.1 (Correctness of Algorithm 4)**.** *Given a temporal graph $\mathbb{G}$ and a time period Δ, then Algorithm 4 enumerates only maximal Δ-cliques of $\mathbb{G}$ and each maximal Δ-clique exactly once.*

The adaption of the Bron-Kerbosch algorithm is based on the following observation: Assume a Δ-clique (C, I) with a set of vertices C and a maximal time-interval I is given. By adding a vertex to the set C, the length of the time interval I can only remain the same or decrease to produce a new Δ-clique. Due to this observation, the algorithm starts with an empty vertex set and the entire time interval of the temporal graph.

In each recursion step, the number of vertices increases while the length of the time intervals monotonically decreases.

3.1 Description of the Algorithm

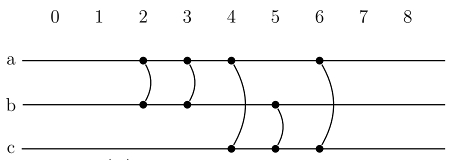
As shown in Algorithm 4, the function receives five parameters $P, P_{\max}, X, X_{\max}$, and R. The tuple $R = (C, I)$ contains a set of vertices C and a time interval I. As we will prove below, this tuple forms a Δ-clique with a maximal time interval I with respect to C. The sets $P, P_{\max}, X$, and $X_{\max}$ contain vertex-interval pairs and it holds $P_{\max} \subseteq P$ and $X_{\max} \subseteq X$ by definition of these sets. We will also prove below that $P \cup X$ includes all vertex-interval pairs (v, I') for which $(C \cup \{v\}, I')$ is a time-maximal Δ-clique because v is a Δ-neighbor of every vertex $w \in C$ during the time $I' \subseteq I$. For each $(v, I) \in P_{\max} \cup X_{\max}$ the vertices $C \cup \{v\}$ form a Δ-clique over the whole time interval I of the original Δ-clique $R = (C, I)$. While each vertex-interval pair in P still has to be combined with R to ensure that every maximal Δ-clique will be found, for every vertex-interval pair $(v, I') \in X$ every possible Δ-clique (C', I'') with $C \cup \{v\} \subseteq C'$ and $I'' \subseteq I'$ has already been detected in earlier steps. The same holds for $P_{\max}$ and $X_{\max}$.

If $P_{\max} \cup X_{\max} = \emptyset$, there is no vertex v that forms a Δ-clique together with C over the whole time interval I of the original Δ-clique $R = (C, I)$. Due to the maximality of the time interval I of R, we therefore know that R is a maximal Δ-clique and has to be added to the solution.

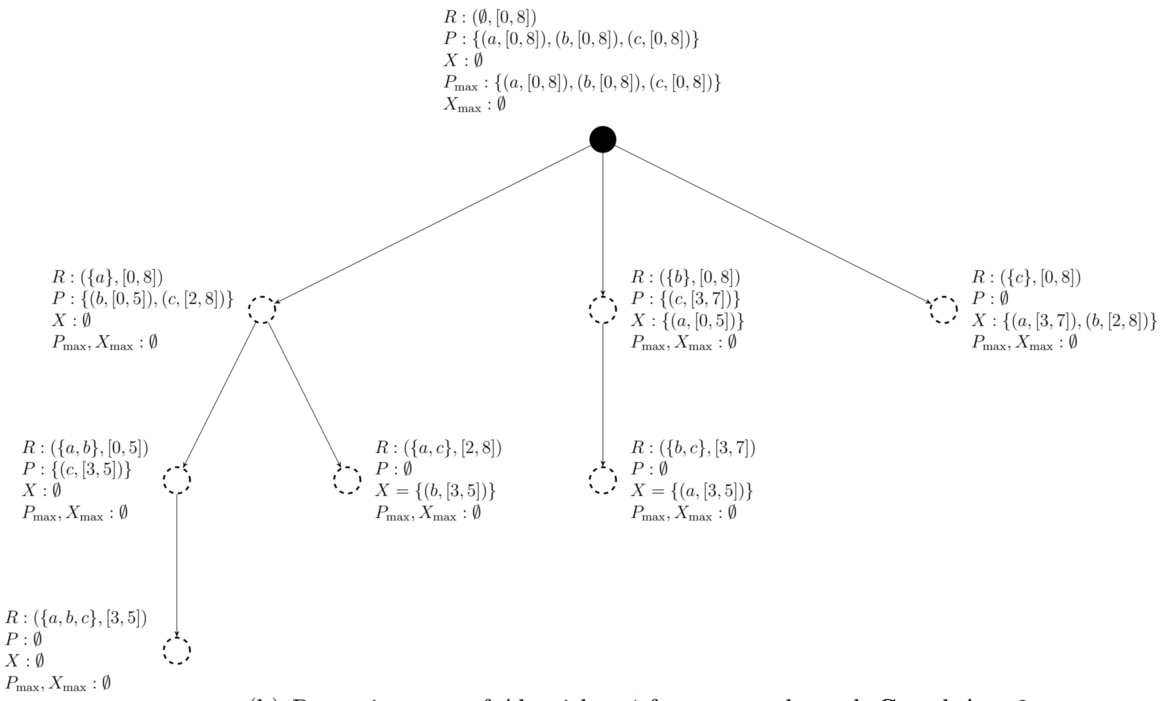
In the next step, for every vertex-interval pair $(v, I') \in P$ a recursive call is initiated for the Δ-clique $R' = (C \cup \{v\}, I')$ with all parameters restricted to the Δ-neighborhood of v in the time interval I'—$P \sqcap N^{\Delta}(v, I')$ and $X \sqcap N^{\Delta}(v, I')$. For the set P' for example, we get a set of all time-maximal vertex-interval pairs (w, I'') for which it holds that $(w, I'') \sqsubseteq N^{\Delta}(v, I')$ and $(w, I'') \sqsubseteq P$. This restriction is made so that for all $(w, I'') \in P'$ of the recursive call the vertex w is not only a Δ-neighbor of all $x \in C$ but also of the vertex v during the time $I'' \subseteq I'$.

After the recursive call for Δ-clique $(C \cup \{v\}, I')$, the tuple (v, I') is removed from the set P and added to the set X to avoid that the same cliques are found multiple times. For a temporal graph $\mathbb{G} = (T, V, E)$ and a given time period Δ, the *initial call* for Algorithm 4 to enumerate all maximal Δ-cliques in graph $\mathbb{G}$ is made with $P = \{(v, T) \mid v \in V\}, P_{\max} = \{(v, T) \mid v \in V\}, R = (\emptyset, T), X = \emptyset$, and $X_{\max} = \emptyset$.

Before we try to improve the running time of the algorithm by adapting the idea of pivoting—see Subsection 3.4—we prove the correctness of Algorithm 4: it must be shown that all elements in the output are Δ-cliques, that all elements are maximal Δ-cliques, that all maximal Δ-cliques are in the output, and that no maximal Δ-clique is found more than once. To present these proofs, we have to formally define the corresponding recursion tree that is built by the recursive calls of the initial call of Algorithm 4.



(a) Temporal graph $\mathbb{G}$


$R : (\emptyset, [0, 8])$
$P : \{(a, [0, 8]), (b, [0, 8]), (c, [0, 8])\}$
$X : \emptyset$
$P_{\max} : \{(a, [0, 8]), (b, [0, 8]), (c, [0, 8])\}$
$X_{\max} : \emptyset$
$R : (\{a\}, [0, 8])$
$P : \{(b, [0, 5]), (c, [2, 8])\}$
$X : \emptyset$
$P_{\max}, X_{\max} : \emptyset$
$R : (\{b\}, [0, 8])$
$P : \{(c, [3, 7])\}$
$X : \{(a, [0, 5])\}$
$P_{\max}, X_{\max} : \emptyset$
$R : (\{c\}, [0, 8])$
$P : \emptyset$
$X : \{(a, [3, 7]), (b, [2, 8])\}$
$P_{\max}, X_{\max} : \emptyset$
$R : (\{a, b\}, [0, 5])$
$P : \{(c, [3, 5])\}$
$X : \emptyset$
$P_{\max}, X_{\max} : \emptyset$
$R : (\{a, c\}, [2, 8])$
$P : \emptyset$
$X = \{(b, [3, 5])\}$
$P_{\max}, X_{\max} : \emptyset$
$R : (\{b, c\}, [3, 7])$
$P : \emptyset$
$X = \{(a, [3, 5])\}$
$P_{\max}, X_{\max} : \emptyset$
$R : (\{a, b, c\}, [3, 5])$
$P : \emptyset$
$X : \emptyset$
$P_{\max}, X_{\max} : \emptyset$

(b) Recursion tree of Algorithm 4 for temporal graph $\mathbb{G}$ and $\Delta = 2$

Figure 2: A recursion tree of the temporal graph of Figure 2(a) for Δ=2 generated by the initial call of Algorithm 4. All nodes—except the root—signal maximal Δ-cliques.

3.2 Properties of the Recursion Tree

In the recursion tree, each tree node corresponds to a call of Algorithm 4, each edge between two tree nodes indicates that the lower call is induced by the upper one. Each tree node is associated with the sets $P, P_{\max}, X, X_{\max}$, and $R = (C, I)$—the parameters of the function call. To clarify the syntax, we denote these parameters for a node p by $P^p, P^p_{\max}, X^p, X^p_{\max}$, and R^p. For each tuple $(v, I') \in P^p$ of a tree node p there exists a child node c such that $R^c = (C^p \cup \{v\}, I')$. Thus, if $P = \emptyset$, then the tree node p is a leaf. Moreover, we know that $P^c \sqsubseteq P^p$ and $X^c \sqsubseteq X^p$. The child nodes are ordered from left to right according to the recursive calls. For each child c of a parent p with $C^c = C^p \cup \{v\}$ and $I^c = I'$ of a $(v, I') \in P$, every child d of p left of child c with $C^d = C^p \cup \{w\}$ and $I^d = I''$ of a $(w, I'') \in P$, there is no tuple $(w, I''') \in P^c$ with $I''' \subseteq I''$. The root of the tree is defined by the parameters of the initial call for a temporal graph. An example of the recursion tree is given in Figure 2 based on the temporal graph introduced in Figure 1.

With the help of the structure of the recursion tree created by the initial call of Algorithm 4 for a temporal graph, we prove the correctness of the algorithm in the following section.

3.3 Proof of Theorem 3.1

In all proofs we talk about the recursion tree induced by the initial call of Algorithm 4 of a temporal graph $\mathbb{G} = (T, V, E)$ and a fixed $\Delta \in \mathbb{N}$. Some proofs are by induction on the depth of the tree nodes. The *depth* $d \in \mathbb{N}$ of a tree node is one plus the depth of the parent node starting at the root with depth $d = 0$.

At first we show that the set $P \cup X$ in each node of the recursion tree contains all vertex-interval pairs (v, I') that meet the condition that v is a Δ-neighbor of all $c \in C$ during time $I' \subseteq I$. We rely on this observation in nearly all subsequent proofs to determine the correctness of the algorithm.

Lemma 3.2. *Given a node in the recursion tree with $R = (C, I)$, then $P \cup X = \{(v, T) \mid v \in V\} \sqcap \prod_{v \in C} N^\Delta(v, I)$.*

Proof. We prove this lemma by induction on the depth of the tree nodes. For the root with $d = 0$, the set $C = \emptyset$ and consequently $P \cup X = P = \{(v, T) \mid v \in V\}$.

Now we assume that for each tree node with depth $d \leq k$ with $k \in \mathbb{N}$ the set $P \cup X = \prod_{c \in C} N^\Delta(c, I)$. We have to show that the the assumption also holds for tree nodes with $d = k + 1$. Let c be an arbitrary node with $d = k + 1$ and p be the parent node of c. Due to the induction hypothesis, we know that $P^p \cup X^p = \prod_{v \in C^p} N^\Delta(v, I^p)$. We also know that there exists a $(w, I') \in P^p$ with $C^c = C^p \cup \{w\}$ and $I^c = I'$. Due to the construction of

Algorithm 4, the following holds:

$$\begin{aligned}
P^c \cup X^c &= \bigsqcap_{v \in C^p} N^\Delta(v, I) \sqcap N^\Delta(w, I^c) \\
&= \bigsqcap_{v \in C^p} N^\Delta(v, I^c) \sqcap N^\Delta(w, I^c) \\
&= \bigsqcap_{v \in C^p \cup \{w\}} N^\Delta(v, I^c) \\
&= \bigsqcap_{v \in C^c} N^\Delta(v, I^c)
\end{aligned}$$

Hence, for every node of the recursion tree with $R = (C, I)$ it holds that $P \cup X = \bigsqcap_{v \in C} N^\Delta(v, I)$.

□

The proof of Lemma 3.2 does not only show us that for a given node in the recursion tree with a Δ-clique $R = (C, I)$ for all $(v, I') \in P \cup X$ and for all $c \in C$ it holds that $(v, I') \sqsubseteq N^\Delta(c, I)$, but also shows that all elements (v, I') that can expand the Δ-clique R are in these sets. Furthermore, it gives us the certainty that the time interval I' of every $(v, I') \in P \cup X$ is maximal. This observation relies on the fact that the Δ-cut ('$\sqcap$') and the Δ-neighborhood ('N^Δ') always return sets with vertex-interval pairs that consist of maximal time intervals by definition.

We continue by showing that the tuple $R = (C, I)$ is a Δ-clique in each node of the recursion tree and that this Δ-clique is time-maximal. We will need these proofs to show that all elements that will be added to the solution are in fact maximal Δ-cliques.

Lemma 3.3. *The tuple $R = (C, I)$ of a node in the recursion tree is a Δ-clique.*

Proof. We prove the lemma by induction on the depth of the tree nodes. For the root with $d = 0$, the parameter $R = (C = \emptyset, I = T)$ of this initial call is a trivial Δ-clique. Furthermore, due to the set P of this initial call, for each $v \in V$ there exists a child node of the root with $R = (C = \{v\}, I = T)$—a trivial Δ-clique. These nodes are the only child nodes of the root and consequently the only tree nodes with $d = 1$. Thus, every tree node with $d = 1$ is also a Δ-clique.

Now we assume that for each tree node with depth $d \leq k$ with $k \in \mathbb{N}$ the tuple $R = (C, I)$ is a Δ-clique. We have to show that for a tree node with depth $d = k + 1$ the tuple R is also a Δ-clique. Let c be any node with $d = k + 1$ and let p be the parent of this node. Due to the induction hypothesis, we know that $R^p = (C^p, I^p)$ is a Δ-clique. Let $(v, I') \in P^p$ be the vertex-interval pair with $C^c = C^p \cup \{v\}$ and $I^c = I'$. By Lemma 3.2 we know that the vertex v is a Δ-neighbor of all vertices in the set C^p during time I'. Consequently, $R^c = (C^c, I^c)$ is a Δ-clique. Ergo, for every tree node in the recursion tree with $d = k + 1$ the tuple $R = (C, I)$ is a Δ-clique. □

We have shown that $R = (C, I)$ is a Δ-clique in all nodes of the recursion tree. We now have to prove that these Δ-cliques are also time-maximal. This means that we cannot increase the time interval I of R without losing the properties of a Δ-clique.

Lemma 3.4. *Each Δ-clique $R = (C, I)$ of a node in the recursion tree is time-maximal.*

Proof. We will, again, prove the lemma by induction on the depth of the tree nodes. For the root with $d = 0$, the Δ-clique $R = (C = \emptyset, I = T)$ is time-maximal because there is no $I' \subseteq T$ with $I = T \subset I'$. The same holds for every tree node with $d = 1$ where $R = (C = \{v\}, I = T)$ for a $v \in V$.

Now we assume that for each tree node with depth $d \leq k$ with $k \in \mathbb{N}$ the Δ-clique $R = (C, I)$ is time-maximal. We have to show that for every tree node with $d = k + 1$ the Δ-clique $R = (C, I)$ is time-maximal. Let c be any node with $d = k+1$ and let p be the parent node of c. Let $(v, I') \in P^p$ be the vertex-interval pair with $C^c = C^p \cup \{v\}$ and $I^c = I'$. We know that the Δ-clique $R^p = (C^p, I^p)$ is time-maximal due to the induction hypothesis. Furthermore, we have proven in Lemma 3.2 that the time intervals of the vertex-interval pairs in P^p and X^p are maximal. Hence, the Δ-clique R^c is time-maximal as well. □

We can now easily prove, that all Δ-cliques that are added to the solution by Algorithm 4 are maximal Δ-cliques.

Lemma 3.5. *Each Δ-clique of a node in the recursion tree that is added to the solution is a maximal Δ-clique.*

Proof. Let $R = (C, I)$ be a Δ-clique that is added to the solution and $P, P_{\max}$ and $X, X_{\max}$ be the corresponding sets of vertex-interval pairs in the tree node of this Δ-clique. We already know by Lemma 3.3 that the Δ-clique R is time-maximal. Furthermore, according to the Algorithm 4, $P_{\max} = \emptyset$ and $X_{\max} = \emptyset$ so that R is added to the solution. In Lemma 3.2 we have proven that the set $P \cup X$ contains all vertex-interval pairs (v, I') for which v is a Δ-neighbor of all vertices in C during time I'. If $P_{\max} \cup X_{\max} = \emptyset$, then there is no vertex $v \in V \setminus C$ that will form a Δ-clique with C over the whole time interval I. Thus, $R = (C, I)$ is vertex-maximal as well as time-maximal and consequently not contained in any other Δ-clique. Hence, R is a maximal Δ-clique. □

We now prove that all maximal cliques are found by Algorithm 4. The proof is conducted by showing that every maximal Δ-clique is represented in a node of the recursion tree. In order to conclude from this proof that all maximal Δ-cliques are added to the solution, we have to show first that every maximal Δ-clique that appears in a node of the recursion tree is added to the solution.

Lemma 3.6. *Each Δ-clique of a node in the recursion tree that is a maximal Δ-clique is added to the solution.*

Proof. Assume there is a maximal Δ-clique $R = (C, I)$ in a node of the recursion tree. Let $P, P_{\max}$ and $X, X_{\max}$ be the corresponding sets of vertex-interval pairs in this tree node. This Δ-clique R is time-maximal and, more importantly, vertex-maximal. Thus, no vertex $v \in V \setminus C$ exists such that $(C \cup \{v\}, I)$ is a Δ-clique. Ergo, there is no vertex-interval pair $(*, I) \in P \cup X$. Consequently, $P_{\max} \cup X_{\max} = \emptyset$ and R has been added to the solution. □

After proving that all maximal Δ-cliques that are represented in a node of the recursion tree are added to the solution, it remains to show that every maximal Δ-clique is represented in a tree node. This proof is made by contradiction, showing that for all maximal Δ-cliques a path to a node exists representing the Δ-clique.

Lemma 3.7. *Each maximal Δ-clique of the temporal graph has at least one node within the recursion tree.*

Proof. The proof is conducted by contradiction. Assume there is a Δ-clique $R = (C, I)$ that does not appear in any node of the recursion tree. Let s be the tree node with the greatest depth d of the recursion tree which the clique R can still be generated from, that is, $C^s \subseteq C, I \subseteq I^s$, and for every $v \in C \setminus C^s$ it holds that $(v, I) \sqsubseteq P^s$. We consider hereby the set P^s at the beginning of the function call. These conditions are definitely given in the root of the recursion tree with $d = 0$. Let t be the leftmost tree node that is generated by a vertex-interval pair $(v, I') \in P^s$ with $v \in C \setminus C^s$ and $I \subseteq I' \subseteq T$. Thus, all vertex-interval pairs (w, I'') with $w \in C \setminus C^s$ and $I \subseteq I'' \subseteq T$ have not been removed from P^s yet. It holds $C^s \subset C^t \subseteq C$. Due to the fact that $R = (C, I)$ is a Δ-clique, we know that every $w \in C \setminus C^t$ is a Δ-neighbor of vertex v at least during I. Consequently, for every $w \in C \setminus C^t$ there exists a $(w, I) \sqsubseteq P^s \sqcap N^\Delta(v, I') = P^t$. Thus, the Δ-clique can also be generated from node t which is a child of the node s and thus has a greater depth d than its parent s. This is a contradiction to the assumption that s is the tree node with the greatest depth from which the Δ-clique R can be generated. □

Additionally, Algorithm 4 also ensures that every maximal Δ-clique is added to the solution only once. This proof is conducted again by contradiction, proving that every Δ-clique appears not at least but exactly once in a node of the recursion tree.

Lemma 3.8. *Each maximal Δ-clique of the temporal graph has exactly one node within the recursion tree.*

Proof. Assume there are two nodes in the recursion tree with the same maximal Δ-clique $R = (C, I)$. Let p be the last node that the paths from the root to these nodes have in common. It holds $C^p \subset C$, $I \subseteq I^p$, and for every $v \in C \setminus C^p$ it holds that $(v, I) \sqsubseteq P^p$ at the beginning of the function call. Let s be the child of p and the next node in the left path, let t also be a child of p and the next node in the right path. The node s is the leftmost node that is generated from a vertex-interval pair $(v, I') \in P^p$ with $v \in C \setminus C^p$ and $I \subseteq I' \subseteq T$. Recall that the nodes are ordered from left to right according to the recursive calls. This means that the recursive call for node s is induced by node p before the recursive call for node t is induced. The vertex-interval pair (v, I') is removed from P^p and added to X^p after the recursive call for node s. Hence, at the time when the recursive call for node t is made, the Δ-clique can not be generated from set P^p any longer because $(v, I') \notin P^p$ and consequently $(v, I) \not\sqsubseteq P^t$. Ergo, the right path does not lead to a node that represents $R = (C, I)$ and consequently there is no second node that represents R. This is a contradiction to our assumption and therefore there is exactly one node for each maximal Δ-clique in the recursion tree. □

Algorithm 5 Enumerating all maximal Δ-cliques in a temporal graph with pivoting

```
1: function BronKerboschDeltaPivot($P, P_{\max}, R = (C, I), X, X_{\max}$)
2:     if $P_{\max} \cup X_{\max} = \emptyset$ then
3:         add $R$ as maximal clique to solution
4:     end if
5:     choose pivot element $(v_p, I_p) \in P \cup X$
6:     for $(v, I') \in P \setminus \{(w, I'') \mid (w, I'') \in P \wedge (w_2, I'') \sqsubseteq N^{\Delta}(v_p, I_p)\}$ do
7:         $R' \leftarrow (C \cup \{v\}, I')$
8:         $P' \leftarrow P \sqcap N^{\Delta}(v, I')$
9:         $X' \leftarrow X \sqcap N^{\Delta}(v, I')$
10:        $P'_{\max} \leftarrow \{(w, I') \mid (w, I') \in P'\}$
11:        $X'_{\max} \leftarrow \{(w, I') \mid (w, I') \in X'\}$
12:        BronKerboschDeltaPivot($P', P'_{\max}, R', X', X'_{\max}$)
13:        $P \leftarrow P \setminus (v, I')$
14:        $X \leftarrow X \cup (v, I')$
15:    end for
16: end function
```

By proving that each maximal Δ-clique has exactly one node in the recursion tree, it follows that each maximal Δ-clique is added to the solution only once. We have now proven that our algorithm finds all maximal Δ-cliques of a temporal graph exactly once and adds them to the solution. Hence, we can finally conclude Theorem 3.1 from Lemmata 3.2-3.8.

We have proven the correctness of Algorithm 4. Now we examine a possibility to improve the running time of the algorithm. We adapt the idea of pivoting in the basic Bron-Kerbosch Algorithm—see subsection 2.1—to decrease the number of recursive calls in each function call.

3.4 Pivoting

We try to reduce the number of recursive calls in each step. Therefore, we recall the idea of pivoting of the Bron-Kerbosch algorithm for non-temporal graphs and adapt this idea to the Bron-Kerbosch algorithm for temporal graphs which was introduced in the previous section.

The idea of pivoting in the Bron-Kerbosch algorithm for non-temporal graphs—as we have already discussed in Section 2.1—is based on the observation that for any vertex $u \in P \cup X$—the pivot element—either u itself or one of its non-neighbors must be contained in any maximal clique containing R. This observation also holds for maximal Δ-cliques in temporal graphs: for any $(v_p, I_p) \in P \cup X$—the pivot element—in any maximal Δ-clique $R_{\max} = (C_{\max}, I_{\max})$ with $C \subset C_{\max}$ and $I_{\max} \subseteq I_p \subseteq I$ either the vertex v_p or one vertex w which is not a Δ-neighbor of v_p during the time $I_{\max}$—$(w, I_{\max}) \not\sqsubseteq N^{\Delta}(v_p, I_p)$—must be contained in $C_{\max}$.

In order to make this a bit more obvious, let us consider the following example: Let $R = (C, I)$, $(v, I') \in P$, and $(v_p, I_p) \in P \cup X$. It holds that $(v, I') \sqsubseteq N^{\Delta}(v_p, I_p)$, implying v is a Δ-neighbor of v_p during $I' \subseteq I_p$. We choose the pivot element $(v_p, I_p) \in P \cup X$. We now know for every maximal Δ-clique $R_{\max} = (C_{\max}, I_{\max})$ that is constructed by expanding R by (v, I') it either holds that $v_p \in C_{\max}$ or $w \in C_{\max}$ with $(w, I'') \in P \cup X$, $I_{\max} \subseteq I''$, and $(w, I_{\max}) \not\sqsubseteq N^{\Delta}(v_p, I_p)$. Consequently $(w, I'') \not\sqsubseteq N^{\Delta}(v_p, I_p)$ because if w is not a Δ-neighbor of v_p during $I_{\max}$, then it can not be a Δ-neighbor of v_p during I'' which contains $I_{\max}$. In both cases, we would also find the maximal Δ-clique $R_{\max}$ by the recursive calls for (v_p, I_p) expanding R or (w, I'') expanding R.

The previous paragraph provided an example for pivoting in the Bron-Kerbosch algorithm for enumerating maximal Δ-cliques in temporal graphs. We now prove that our intuition is correct in the following lemma:

Lemma 3.9. *For each Δ-clique $R = (C, I)$ in a node of the recursion tree and a pivot element $(v_p, I_p) \in P \cup X$ the following holds: for every $R_{\max} = (C_{\max}, I_{\max})$ with $C \subset C_{\max}$ and $I_{\max} \subseteq I_p \subseteq I$ it either holds that $v_p \in C_{\max}$ or there is a vertex $w \in C_{\max}$ that satisfies $(w, I') \in P \cup X$, $I_{\max} \subseteq I'$, and $(w, I_{\max}) \not\sqsubseteq N^{\Delta}(v_p, I_p)$ and consequently $(w, I') \not\sqsubseteq N^{\Delta}(v_p, I_p)$.*

Proof. Let $R_{\max} = (C_{\max}, I_{\max})$ be a maximal Δ-clique with $C \subset C_{\max}$ and $I_{\max} \subseteq I_p \subseteq I$. Assume all $w \in C_{\max}$ hold $(w, I_{\max}) \sqsubseteq N^{\Delta}(v_p, I_p)$. Consequently, for all $w \in C_{\max} \setminus C$ a $(w, I') \in P \cup X$ with $I_{\max} \subseteq I'$ must exist. Because v is a Δ-neighbor of all vertices in $C_{\max} \setminus C$ at least during $I_{\max}$ and a Δ-neighbor of all vertices in C during I_p, the vertex v_p can be added to the Δ-clique $R_{\max}$. This shows that $R_{\max}$ is not vertex-maximal, and consequently not maximal—this is a contradiction to the assumption that $R_{\max}$ is maximal. □

Conversely, by choosing a pivot element $(v_p, I_p) \in X \cup P$ we only have to iterate over all elements in P which are not in the Δ-neighborhood of the pivot element. In other words, we do not have to make a recursive call for any $(w, I') \in P$ which holds $(w, I') \sqsubseteq N^{\Delta}(v_p, I_p)$. Due to Lemma 3.9, we know that Algorithm 5—the implementation of pivoting—also enumerates all maximal Δ cliques in a temporal graph.

The pivot element must be chosen in such a way that minimizes the number of recursive calls. The optimal pivot element is the element in the set $P \cup X$ of which the most elements in P are in its Δ-neighborhood. We have seen that the whole procedure is quite similar to pivoting in the basic Bron-Kerbosch algorithm but with one difference: we are able to choose more than one pivot element. The only condition that has to be satisfied is that the time intervals of the pivot elements cannot overlap. This might not be obvious at first glance but a closer look at Lemma 3.9 sheds some light:

For each Δ-clique $R = (C, I)$ in a node of the recursion tree, choosing a pivot element $(v_p, I_p) \in P \cup X$ only affects maximal Δ-cliques $R_{\max} = (C_{\max}, I_{\max})$ fulfilling $I_{\max} \subseteq I_p$. Furthermore, for all elements $(w, I') \in P$ satisfying $(w, I') \sqsubseteq N^{\Delta}(v_p, I_p)$ it holds $I' \subseteq I_p$. Consequently, a further pivot element $(v'_p, I'_p) \in P \cup X$ fulfilling that I'_p does not overlap with I_p, neither interferes with the considered maximal Δ-cliques nor with the vertex-interval pairs in P that are in the Δ-neighborhood of the pivot element (v_p, I_p).

Now we face the problem that choosing the optimal set of pivot elements is no longer a simple task. We have to find the optimal subset of vertex-interval pairs in $P \cup X$ that fulfill the condition that the time intervals do not intersect and that maximizes the number of elements in P for which we do not have to make a recursive call due to the previous argumentation in Lemma 3.9. The problem of finding the optimal set of pivot elements in $P \cup X$ can be formulated as a weighted interval scheduling maximization problem:

WEIGHTED INTERVAL SCHEDULING MAXIMIZATION PROBLEM
Input: A set J of jobs j with a time interval I_j and a weight w_j
Task: Find a subset of jobs $J' \subseteq J$ that maximizes $\sum_{j \in J'} w_j$ while fulfilling that for all $i, j \in J'$ with $i \neq j$, the time intervals I_i and I_j do not overlap!

In our problem, the jobs are the elements of $P \cup X$ and the weight of an element is thereby the number of all elements that are in P and lie in the Δ-neighborhood of this element. Formally, the jobs are the elements $(v, I') \in P \cup X$, the corresponding time interval is I' of the element (v, I') and the corresponding weight $w_{(v,I')} = \mid \{(v, I) \mid (v, I) \in P \wedge (v, I) \sqsubseteq N^{\Delta}(v, I')\} \mid$. This problem can be solved efficiently in $O(n \log n)$ time by using dynamic programming under the assumption that the weights of the potential pivot elements are known.

We have shown that by choosing a set of pivot elements, we can reduce the number of recursive calls in our algorithm and therefore enable faster backtracking. The selection of an optimal set of pivot elements can thereby be done efficiently in $O(n \log n)$ time. To complete this thesis, we test our algorithm on real-world datasets to examine the running time of our adaption of the Bron-Kerbosch algorithm to temporal graphs. We concentrate on the implementation of Algorithm 4 to explore the efficiency of the basic algorithm idea.

4 Implementation and Experiments

We implemented Algorithm 4 to explore the efficiency of our algorithm on different datasets and to compare our algorithm to the first approach for computing maximal Δ-cliques in temporal graphs. This approach was presented by Viard, Latapy, and Magnien [VLM16]. The source code in Python was provided by the authors on GitHub [VL14]. To ensure comparability, we decided to implement our algorithm in the same programming language. The algorithm itself is very short and straightforward to program. For this reason, we decided to limit our elaborations to choice design decisions concerning the representation of the dataset as Δ-neighborhoods and data structures. Furthermore, we look into the implementation of the Δ-cut-function ('$\sqcap$'), the only non-obvious and time-consuming function in the algorithm.

4.1 Implementation Design

After importing the dataset of the temporal graph, we compute the whole Δ-neighborhood of every vertex throughout its entire lifetime. The algorithm only works on these Δ-neighborhoods. The dataset itself is no longer needed. As a result of the Δ-neighborhood representation of the graph we lose some information in respect to the detailed time-edges between the vertices. Thus, if the value of Δ changes, then the new Δ-neighborhood has to be computed again with the help of the original dataset. This implies a trade-off between storing the original dataset and its repeated re-importation in order to compute the Δ-neighborhoods for changing values of Δ.

Δ-Neighborhood Let us shortly recap the Δ-neighborhood in a temporal graph $\mathbb{G} = (T, V, E)$: For a vertex $v \in V$ and a time interval $I \subseteq T$ the Δ-neighborhood $N^{\Delta}(v, I)$ is the set of all vertex-interval pairs $(w, I' = [a, b])$ with the property that for every $\tau \in [a, b - \Delta]$ at least one edge $(\{v, w\}, t) \in E$ with $t \in [\tau, \tau + \Delta]$ exists. Furthermore, it holds that $b - a \geq \Delta$, $I' \subseteq I$ and I' is maximal.

Computing the Δ-neighborhood of all vertices v over the whole lifetime of the graph—$N^{\Delta}(v, T)$—is doable in polynomial time. After importing the dataset, we have a set of all vertices, the beginning and end of the lifetime and a hash map representing the time-edges of the temporal graph. Thereby, every pair of vertices that has at least one contact represents a key in this hash map. The value of the key is a set of all points in time where a contact between these two vertices appeared. To compute the Δ-neighborhoods of a given temporal graph, we have to iterate over all keys in the hash map and compute the time intervals during which the two vertices of the key are Δ-neighbors. Therefore, we simply have to sort the set of time stamps, iterate over the resulting ascending list, and compute all maximal time intervals during which the two vertices have a contact at least after every Δ time steps, implying they are Δ-neighbors during these time intervals.

Data Structures Due to the symmetry of the Δ-neighbor relation, we can also use a hash map data structure to store the Δ-neighborhoods. The set of two vertices—let us call them v and w—represents a key in the hash map and an ascending list L of time intervals represents the corresponding value of the key. For all time intervals $I \in L$ it holds that the vertices v and w are Δ-neighbors during I.

The sets P and X are also represented as hash maps. Let us consider the set P: A vertex v is the key and an ascending list L of time intervals is the corresponding value. For all $I \in L$ it holds $(v, I) \in P$. The same holds for the set X.

Two important properties have to be highlighted concerning the lists of intervals in both, the Δ-neighborhoods and the sets P and X: These lists of intervals are always in ascending order relative to the starting points of the intervals. Furthermore, due to the maximality of the time intervals in the Δ-neighborhoods as well as in the sets P and X, the intervals in the list can only overlap in less than Δ time steps. The reader should keep this in mind to understand the implementation of the Δ-cut-function. The choice of this data structure will be justified when we discuss the Δ-cut function.

Δ-Cut We shortly recap the definition of this function before we discuss the implementation: Let X and Y be sets of vertex-interval pairs. The Δ-*cut* $X \sqcap Y$ includes all $(v, I = [a, b])$ so that $b - a \geq \Delta$ and I is maximal with respect to inclusion among all intervals J that fulfill the existence of I' and I'' with $(v, I') \in X$ and $(v, I'') \in Y$ and $J \subseteq I'$ and $J \subseteq I''$.

The Δ-cut function appears twice in the code of Algorithm 4: When the Δ-clique is expanded by adding the vertex v to the clique and when reducing the time interval to I' for $(v, I') \in P$, the sets X and P also have to be adapted to the new vertex v and the new time interval I'. Let us consider the set P: the Δ-cut of the set P and the neighborhood $N^{\Delta}(v, I')$ has to be found. We now present an efficient way of computing this Δ-cut—$P \sqcap N^{\Delta}(v, I')$:

To find the Δ-cut of P and $N^{\Delta}(v, I')$, we only have to examine the vertices that exist in a vertex-time pair of P since a vertex that does not already exist in P cannot exist in the resulting set of this Δ-cut. Consequently, we proceed to go through each vertex w that exists in a vertex-time pair of P—the keys of the hash map of P:

- get list L_w of the vertex w in the hash map of the set P.
  The list L_w contains all time intervals I such that $(w, I) \in P$ holds in ascending order relative to the starting points.

- get list $L_{\{v,w\}}$ of the set $\{v, w\}$ in the hash map of the Δ-neighborhoods.
  The list $L_{\{v,w\}}$ contains all time intervals during which v and w are Δ-neighbors in ascending order relative to the starting points.

- find all intersections of the time intervals in the lists $L_{\{v,w\}}$ and L_w during the time I' with at least size Δ.
  The time intervals within these lists only overlap in less than Δ time steps. Thus, we can go through these lists concurrently without regression to find all overlaps during time I' with at least size Δ. The new list L'_w containing these overlaps is automatically sorted in ascending order relative to the starting points of the time intervals.

- replace L_w with the new list L'_w.
  If list L'_w is empty, then we delete the vertex/key w from the hash map of the set P.

We not only get an ascending list L'_w of time intervals such that for all $I'' \in L'_w$ it holds that $I'' \subseteq I'$ and it holds that the vertex w is a Δ-neighbor of all vertices in the new, expanded Δ-clique during time I'', but can also compute this list in linear time. Thanks to the data structure of the hash maps and the lists of time intervals presorted in ascending order, the whole operation works efficiently in polynomial time.

We have shortly described the main design decisions that were made to guarantee an efficient implementation of Algorithm 4. In the following subsection, we begin to test our algorithm using different datasets. We shortly introduce the three datasets and present the main results of our test runs.

4.2 Experiments

We decided to conduct the test runs of our algorithm on three different real world datasets. We ran the experiments on a computer running a 64-bit version of Ubuntu 15.10 with a 2.9 GHz Intel Core i5 4300Ua processor and 8 GB RAM.

4.2.1 Datasets

The first two datasets are high school dynamic contact networks. Both temporal networks contain contact data between high school students in Marseilles, France. In 2011, three classes participated in the experiment which lasted for four consecutive school days. In 2012, five classes participated over a period of seven days from Monday until Tuesday of the following week. The contact time stamps are accurate to the second. The dataset of 2012 was also used by Viard, Latapy, and Magnien [VLM15] to illustrate the relevance of Δ-cliques in temporal graphs. Both datasets are available on the website 'SocioPatterns' [BF14b] and analyzed by Barrat and Fournet [BF14a]. Another dataset is the e-mail network of KIT Informatics released on the KIT website [Goe11]. It contains the e-mail traffic of 1 890 persons between September 2006 and August 2010. The time stamps of the e-mails are also accurate to the second. In Table 1 the reader can find detailed information concerning the number of nodes and the number of time-edges in the temporal graphs as well as the number of Δ-cliques and the maximum number of vertices in a maximal Δ-clique for a selected Δ. For the high school contact network, the value $\Delta = 3\,600$ (1 hour) seems like a suitable value to extract study groups. For the e-mail network of KIT Informatics, the value $\Delta = 2\,592\,000$ (1 month) seems suitable to extract research groups. The running time of our algorithm and the running time of the algorithm by Viard, Latapy, and Magnien [VLM15] for a given Δ is listed in Table 2. For the e-mail network of KIT Informatics, we had to stop the test run of the algorithm by Viard, Latapy, and Magnien [VLM15] after roughly 37 hours due to the lack of computing resources.

Dataset	# nodes	# time edges	Δ	# maximal Δ-cliques	max $\|C\|$
High school dynamic contact network (2011)	126	28 561	3 600	7 692	10
High school dynamic contact network (2012)	180	45 047	3 600	7 349	7
E-mail network of KIT Informatics	1 890	550 000	2 592 000	199 021	14

Table 1: Information on the introduced datasets. The column '$\max |C|$' contains the maximum number of vertices contained in a maximal Δ-clique $R = (C, I)$ of the temporal graphs.

4.2.2 Viard, Latapy, and Magnien Algorithm

We want to analyze the differences in the running time of the two algorithms for enumerating all maximal Δ-cliques for the temporal networks we have introduced. We briefly present the main idea of the greedy algorithm given by Viard, Latapy, and Magnien [VLM15], which we will refer to as the VLM Algorithm in the next subsections. The algorithm works as follows:

Given a temporal graph $\mathbb{G} = (T, V, E)$, the algorithm starts with a set S of all trivial Δ-cliques $(\{a, b\}, [t, t])$ for all $(\{a, b\}, t) \in E$. While the set S is not empty, the algorithm takes a Δ-clique C out of the set S and checks whether a $v \in V$ that can be added to the Δ-clique C exists or the time interval of the Δ-clique C can be enlarged. All the newly found Δ-cliques will be added to the set S. Otherwise, if no such expansions of the Δ-clique C are possible, then this Δ-clique is maximal and will be added to the solution. If the set S is empty, then all maximal Δ-cliques have been found [VLM15; VLM16].

4.2.3 Comparison of the Algorithms

There is a clear improvement in the running time of our algorithm against the VLM algorithm. A possible reason for the notable difference in the running time may lie in the fact that the VLM algorithm computes maximal Δ-cliques multiple times. More precisely, every Δ-clique in a temporal graph is detected as often as the amount of time-edges comprising the Δ-clique. As an improvement, the authors store the Δ-cliques seen so far. This method reduces, but does not avoid this redundancy of computation. Nevertheless, the query to check whether a Δ-clique has already appeared comes at the cost of computing time and storage space, as well as a rapid expansion of the usage of these resources as the algorithm progresses. Furthermore, during the process of finding a maximal Δ-clique, the algorithm detects almost all possible Δ-cliques. Among other factors, these behavior patterns cause the high running time of the algorithm. This assumption is supported by the running time of the VLM algorithm for the two high school dynamic contact networks of 2011 and 2012. Even though the high school dynamic

Dataset	Δ	Running time of Algorithm 4	Running time of VLM algorithm
High school dynamic contact network (2011)	3 600	3.43 sec	3 214.93 sec
	10 800	3.64 sec	6 515.14 sec
High school dynamic contact network (2012)	3 600	3.59 sec	721.11 sec
	10 800	2.95 sec	2 398.29 sec
E-mail network of KIT Informatics	2 592 000	224.14 sec	> 37 h

Table 2: Running times of Algorithm 4 and the algorithm by Viard, Latapy, and Magnien [VLM15] enumerating all maximal Δ-cliques of different datasets. The running times of Algorithm 4 consist of the computation of the Δ-neighborhood and the enumeration of all maximal Δ-cliques. Both algorithms report all maximal Δ-cliques.

contact network of 2011 has less vertices, less time-edges, and a smaller lifetime than the network of 2012, Table 2 surprisingly shows a significantly higher running time. This can be explained as follows: The number of Δ-cliques is higher in 2011, so is the maximum number of vertices in a Δ-clique. For a constant time interval, the more vertices there are in a Δ-clique, the more time-edges comprise the clique. Consequently, more non-maximal Δ-cliques are detected in the process of finding all maximal Δ-cliques in the network of 2011. These non-maximal Δ-cliques have to be stored and queried in every loop of the VLM algorithm.

Our algorithm, in comparison, works only with the Δ-neighborhoods. These neighborhoods can be computed in polynomial time with respect to the input size. Therefore, the algorithm does not have to deal with the entire amount of time-edges. Additionally, our algorithm only deals with time-maximal Δ-cliques as proven in Lemma 3.3. It also avoids enumerating maximal Δ-cliques multiple times naturally as proven in Lemma 3.8.

4.2.4 Behavior for Different Δ-Values

In order to show that the good performance of our algorithm does not depend on the chosen Δ-values, we ran our algorithm on the e-mail network of KIT Informatics for various Δ-values. We ran the test for the value of $\Delta = 1$ day, 1 month, 1 week, and 1 year. Apart from the running time of the algorithm, we are also interested in the number of Δ-cliques and the maximum number of vertices in a maximal Δ-clique that will be found.

Intuitively, we expect that, with smaller Δ-values, more Δ-cliques exist in the temporal graph. With smaller Δ-values, there will most likely be more, but smaller maximal time intervals during which two vertices are Δ-neighbors. Thus, there will be more but smaller time intervals in the Δ-neighborhoods. Consequently, there are more but smaller sized maximal Δ-cliques with respect to the size of the time interval.

We also expect that the size of maximal Δ-cliques with respect to the number of vertices will monotonically decrease with shrinking Δ-values: The size of the time inter-

Δ	Running time of Algorithm 4	# maximal Δ-cliques	max $\lvert C \rvert$
86 400 (1 day)	1 418 sec	247 371	8
604 800 (1 week)	421 sec	207 988	12
2 592 000 (1 month)	224 sec	199 021	14
31 536 000 (1 year)	19 681 sec	327 321	21

Table 3: Running time of Algorithm 4 in relation to different Δ-values in the e-mail network of KIT Informatics.

vals in the Δ-neighborhoods increases and, with it, the probability of two time intervals overlapping. We expect that the higher number of maximal time intervals in the Δ-neighborhoods will form more Δ-cliques than Δ-cliques with a higher number of vertices will get lost. Consequently, the overall number of maximal Δ-cliques in temporal graphs will increase with smaller Δ-values.

In Table 3, we see that our assumption is partly correct. With smaller Δ-values, the maximum number of vertices in maximal Δ-cliques decreases. Furthermore, the number of maximal Δ-cliques increases with smaller Δ-values. But there is one surprising outlier. For $\Delta = 31\,536\,000 = 1$ year, there are 327 321 maximal Δ-cliques with a maximal clique size of 21 with respect to the number of vertices. We would have expected to get only a few large maximal Δ-cliques with respect to the number of vertices and the time intervals for $\Delta = 31\,536\,000 = 1$ year. The high number of maximal Δ-cliques can be explained by the very detailed resolution of the time-stamped e-mails.

All subsets of vertices in a maximal Δ-clique probably form an own maximal Δ-clique over a slightly larger time interval. Consider the following example: we have a Δ-clique $(\{a, b\}, [0, 126\,144\,000])$ indicating that person a has contact with person b at least after every year over four years (= 126 144 000 sec) during the observation time of the e-mail traffic. A person c is in contact with both at least after every year for four years minus 1000 seconds—in the time interval $[0, 126\,143\,000]$. This results in two maximal Δ-cliques $(\{a, b\}, [0, 126\,144\,000])$ and $(\{a, b, c\}, [0, 126\,143\,000])$. The first maximal Δ-clique seems redundant in this case but will be detected due to the resolution of the e-mail time stamps accurate to the second. In Figure 3, we see that a change in the resolution of the time stamps results in fewer small maximal Δ-cliques with respect to the number of vertices while the number of big maximal Δ-cliques remains the same.

This is an overall problem of the resolution of the time stamps: A resolution, which is too fine-grained and differs too much from the chosen Δ-value, results in a multitude of insignificant maximal Δ-cliques. The resolution should be carefully tailored to the purpose of the analysis as well as the Δ-value. Note that the running time of the algorithm for a given Δ-value does not change with different resolutions of time stamps. While the changes in the resolution do not notably influence the expansion of the recursion tree of Algorithm 4, they will still alter the number of Δ-cliques which are declared maximal.

For $\Delta = 31\,536\,000 = 1$ year, the running time of the algorithm is significantly higher than for the other Δ-values. This is due to the algorithm's high recursion depth. In this case especially, it would be interesting to explore the implications of the pivoting

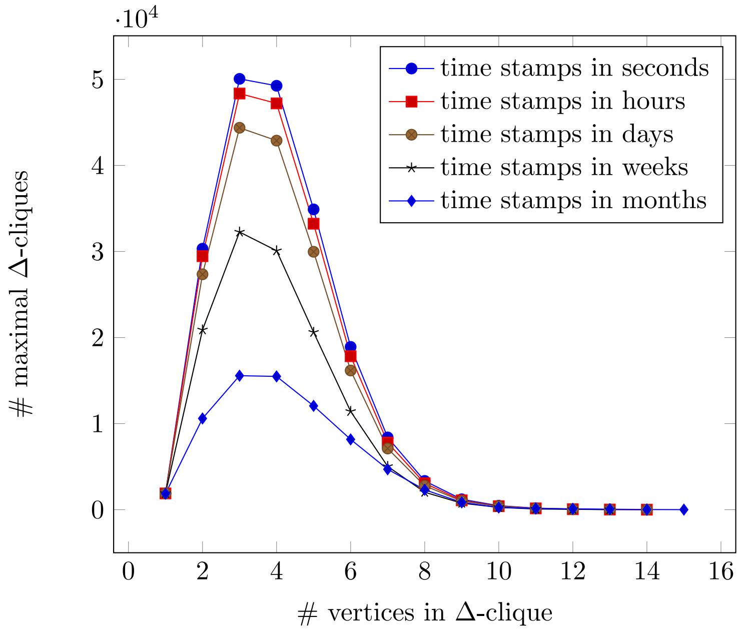


Figure 3: Different resolutions for the time stamps of the e-mail network of KIT Informatics with $\Delta = 2\,592\,000$ (1 month)

procedure introduced in Subsection 3.4, which was not implemented in this thesis due to constraints of time. Note that the general performance of our algorithm is acceptable considering the input size of the temporal graph irrespective of the chosen Δ-value.

5 Conclusion

In this thesis, we studied the concept of cliques in temporal graphs: so-called Δ-cliques. In order to enumerate all maximal Δ-cliques in temporal graphs, we adapted the idea of the Bron-Kerbosch algorithm, including the procedure of pivoting to reduce the number of recursion calls. We were able to maintain the main characteristic of the original Bron-Kerbosch algorithm: all maximal Δ-cliques are naturally enumerated exactly once. In experiments, we could show that our algorithm is notably faster than the first approach for enumerating all maximal Δ-cliques in temporal graphs introduced by Viard, Latapy, and Magnien [VLM16] in real-world datasets.

We introduced the idea of pivoting for Δ-cliques and how this idea can be exploited to reduce the number of recursive calls and therefore enable faster backtracking in our algorithm. Nevertheless, we did not consider this procedure in our test runs. The effects of pivoting are interesting to examine, especially in large datasets such as the e-mail traffic of KIT Informatics. Additionally, different heuristics for choosing a set of pivot elements can be tested to avoid solving a weighted interval scheduling maximization problem in each call of our algorithm.

We mentioned degeneracy—a measure of sparsity in non-temporal graphs—and how this parameter can be exploited to upper-bound the running time of the original Bron-Kerbosch algorithm. A promising perspective could be using degeneracy as a role model to define a measurement of sparsity in temporal graphs. This parameter can be exploited to upper-bound the number of maximal Δ-cliques in temporal graphs and to design a fixed-parameter tractable algorithm for enumerating all maximal Δ-cliques in temporal graphs based on our approach.

Literature

[BF14a] A. Barrat and J. Fournet. “Contact Patterns among High School Students”. In: *PLoS ONE* 9.9 (Sept. 2014), e107878 (cit. on p. 25).

[BF14b] A. Barrat and J. Fournet. *DATASET: High school dynamic contact networks.* http://www.sociopatterns.org/datasets/high-school-dynamic-contact-networks/. 2014 (cit. on p. 25).

[BK73] C. Bron and J. Kerbosch. “Algorithm 457: finding all cliques of an undirected graph”. In: *Communications of the ACM* 16.9 (1973), pp. 575–577 (cit. on pp. 7, 10).

[EHK15] T. Erlebach, M. Hoffmann, and F. Kammer. “On Temporal Graph Exploration”. English. In: *Automata, Languages, and Programming.* Ed. by M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann. Vol. 9134. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2015, pp. 444–455 (cit. on p. 5).

[ELS10] D. Eppstein, M. Löffler, and D. Strash. “Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time”. English. In: *Algorithms and Computation.* Ed. by O. Cheong, K.-Y. Chwa, and K. Park. Vol. 6506. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 403–414 (cit. on pp. 11, 12).

[ELS13] D. Eppstein, M. Löffler, and D. Strash. “Listing All Maximal Cliques in Large Sparse Real-World Graphs in Near-Optimal Time”. In: *ACM Journal of Experimental Algorithmics* 18.3 (2013), 3.1:1–3.1:21 (cit. on p. 7).

[ES11] D. Eppstein and D. Strash. “Listing All Maximal Cliques in Large Sparse Real-World Graphs”. English. In: *Experimental Algorithms.* Ed. by P. Pardalos and S. Rebennack. Vol. 6630. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 364–375 (cit. on p. 10).

[Goe11] R. Goerke. *Email Network of KIT Informatics.* http://i11www.iti.uni-karlsruhe.de/en/projects/spp1307/emaildata. 2011 (cit. on p. 25).

[HS12] P. Holme and J. Saramäki. “Temporal networks”. In: *Physics Reports* 519.3 (2012), pp. 97–125 (cit. on p. 5).

[Mic15] O. Michail. “An Introduction to Temporal Graphs: An Algorithmic Perspective”. English. In: *Algorithms, Probability, Networks, and Games.* Ed. by C. Zaroliagis, G. Pantziou, and S. Kontogiannis. Vol. 9295. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 308–343 (cit. on p. 5).

[MS14] O. Michail and P. Spirakis. “Traveling Salesman Problems in Temporal Graphs”. English. In: *Mathematical Foundations of Computer Science 2014.* Ed. by E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik. Vol. 8635. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 553–564 (cit. on p. 5).

[Nic+13] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora. "Graph Metrics for Temporal Networks". English. In: *Temporal Networks*. Ed. by P. Holme and J. Saramäki. Understanding Complex Systems. Springer Berlin Heidelberg, 2013, pp. 15–40 (cit. on p. 5).

[TTT06] E. Tomita, A. Tanaka, and H. Takahashi. "The worst-case time complexity for generating all maximal cliques and computational experiments". In: *Theoretical Computer Science* 363.1 (2006), pp. 28–42 (cit. on p. 11).

[VL14] J. Viard and M. Latapy. *Source code in Python for computing cliques in link streams*. https://github.com/JordanV/delta-cliques. 2014 (cit. on p. 23).

[VLM15] J. Viard, M. Latapy, and C. Magnien. "Revealing Contact Patterns Among High-school Students Using Maximal Cliques in Link Streams". In: *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*. ASONAM '15. Paris, France: ACM, 2015, pp. 1517–1522 (cit. on pp. 5, 7, 25–27).

[VLM16] J. Viard, M. Latapy, and C. Magnien. "Computing maximal cliques in link streams". In: *Theoretical Computer Science* 609, Part 1 (2016), pp. 245 –252 (cit. on pp. 23, 26, 30).