Adapting the Bron-Kerbosch Algorithm for Enumerating Maximal Cliques in Temporal Graphs[*]

Anne-Sophie Himmel[1], Hendrik Molter[1], Rolf Niedermeier[1], and
Manuel Sorge[2]

[1]Institut für Softwaretechnik und Theoretische Informatik, TU Berlin,
Germany, anne-sophie.himmel@campus.tu-berlin.de,
{h.molter, rolf.niedermeier}@tu-berlin.de
[2]Ben Gurion University of the Negev, Department of Industrial
Engineering and Management, Be'er Sheva, Israel,
sorge@post.bgu.ac.il

August 9, 2018

Abstract

Dynamics of interactions play an increasingly important role in the analysis of complex networks. A modeling framework to capture this are temporal graphs which consist of a set of vertices (entities in the network) and a set of time-stamped binary interactions between the vertices. We focus on enumerating Δ-cliques, an extension of the concept of cliques to temporal graphs: for a given time period Δ, a Δ-clique in a temporal graph 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. Viard, Latapy, and Magnien [ASONAM 2015, TCS 2016] proposed a greedy algorithm for enumerating all maximal Δ-cliques in temporal graphs. In contrast to this approach, we adapt the Bron-Kerbosch algorithm—an efficient, recursive backtracking algorithm which enumerates all maximal cliques in static graphs—to the temporal setting. We obtain encouraging results both in theory (concerning worst-case running time analysis based on the parameter "Δ-slice degeneracy" of the underlying graph) as well as in practice[1] with experiments on real-world data. The latter culminates in an improvement for most interesting Δ-values concerning running time in comparison with the algorithm of Viard, Latapy, and Magnien.

---

1 Introduction

Network analysis is one of the main pillars of data science. Focusing on networks that are modeled by undirected graphs, a fundamental primitive is the identification of complete subgraphs, that is, cliques. This is particularly true in the context of detecting communities in social networks. Finding a maximum-cardinality clique in a graph is a classical NP-hard problem, so super-polynomial worst-case running time seems unavoidable. Moreover, often one wants to solve the more general task of not only finding one maximum-cardinality clique but to list *all maximal* cliques. Their number can be exponential in the graph size. The famous Bron-Kerbosch algorithm ("Algorithm 457" in *Communications of the ACM 1973*, [3]) addresses this task and still today forms the basis for the best (practical) algorithms to enumerate all maximal cliques in static graphs [4]. However, to realistically model many real-world phenomena in social and other network structures, one has to take into account the dynamics of the modeled system of interactions between entities, leading to so-called temporal networks. In a nutshell, compared to the standard static networks, the interactions in temporal networks (edges) appear sporadically over time (while the vertex set remains static). Indeed, as Nicosia et al. [22] pointed out, in many real-world systems the interactions among entities are rarely persistent over time and the non-temporal interpretation is an "oversimplifying approximation". In this work, we use the standard model of temporal graphs. A temporal graph consists of a vertex set and a set of edges, each with an integer time-stamp. The generalization of a clique to the temporal setting that we study is called Δ-clique and was introduced by Viard et al. [28, 29]. Intuitively, being in a Δ-clique means to be regularly in contact with all other entities in this Δ-clique. In slightly more formal terms, each pair of vertices in the Δ-clique has to be in contact in at least every Δ time steps. A fully formal definition is given in Section 2. We present an adaption of the framework of Bron and Kerbosch to temporal graphs. To this end, we overcome several conceptual hurdles and propose a temporal version of the Bron-Kerbosch algorithm as a new standard for efficient enumeration of maximal Δ-cliques in temporal graphs.

1.1 Related Work

Our work relates to two main lines of research. First, enumerating Δ-cliques in temporal graphs generalizes the enumeration of maximal cliques in static graphs, this being subject of many different algorithmic approaches (sometimes also exploiting specific properties such as the "degree of isolation" of the cliques searched for) [3, 4, 12, 14, 16, 25]. Indeed, clique finding is a special case of dense subgraph detection. Second, more recently, mining dynamic or temporal networks for periodic interactions [17] or preserving

structures [26] (in particular, this may include cliques as a very fundamental pattern) has gained increased attention. Our work is directly motivated by the study of Viard et al. [28, 29] who introduced the concept of Δ-cliques and provided a corresponding enumeration algorithm for Δ-cliques. In fact, following one of their concluding remarks on future research possibilities, we adapt the Bron-Kerbosch algorithm to the temporal setting, thereby outperforming their greedy-based approach in most cases.

1.2   Results and Organization

Our main contribution is to adapt the Bron-Kerbosch recursive backtracking algorithm for clique enumeration in static graphs to temporal graphs. In this way, we achieve a significant speedup for most interesting time period values Δ (typically two orders of magnitude of speedup) when compared to a previous algorithm due to Viard et al. [28, 29] which is based on a greedy approach. We also provide a theoretical running time analysis of our Bron-Kerbosch adaption employing the framework of parameterized complexity analysis. The analysis is based on the parameter "Δ-slice degeneracy" which we introduce, an adaption of the degeneracy parameter that is frequently used in static graphs as a measure for sparsity. This extends results concerning the static Bron-Kerbosch algorithm [4]. A particular feature to achieve high efficiency of the standard Bron-Kerbosch algorithm is the use of pivoting, a procedure to reduce the number of recursive calls of the Bron-Kerbosch algorithm. We show how to define this and make it work in the temporal setting, where it becomes a significantly more delicate issue than in the static case. In summary, we propose our temporal version of the Bron-Kerbosch approach as a current standard for enumerating maximal cliques in temporal graphs.

The paper is organized as follows. In Section 2, we introduce all main definitions and notations. In addition, we give a description of the original Bron-Kerbosch algorithm as well as two extensions: pivoting and degeneracy ordering. In Section 3, we propose an adaption of the Bron-Kerbosch algorithm to enumerate all maximal Δ-cliques in a temporal graph, prove the correctness of the algorithm and give a running time upper bound. Furthermore, we adapt the idea of pivoting to the temporal setting. In Section 4 we adapt the concept of degeneracy to the temporal setting and give an improved running time bound for enumerating all maximal Δ-cliques. In Section 5, we present the main results of the experiments on real-world data sets. We measure the Δ-slice degeneracy of real-world temporal graphs, we study the efficiency of our algorithm, and compare its running time to the algorithm of Viard et al. [28], showing a significant performance increase due to our Bron-Kerbosch approach. We conclude in Section 6, also presenting directions for future research.

2   Preliminaries

In this section we introduce the most important notations and definitions used throughout this article.

2.1   Graph-Theoretic Concepts

In the following we provide definitions of adaptations to the temporal setting for central graph-theoretic concepts.

2.1.1   Temporal Graphs

The motivation behind temporal graphs, which are also referred to as temporal networks [11], time-varying graphs [22], or link streams [28], is to capture changes in a graph that occur over time. In this work, we use the well-established model where each edge is given a time stamp [2, 11, 28]. Assuming discrete time steps, this is equivalent to a sequence of static graphs over a fixed set of vertices [5, 19]. Formally, the model is defined as follows.

Definition 1 (Temporal Graph). *A temporal graph $\mathbb{G} = (V, E, T)$ is defined as a triple consisting of a set of vertices V, a set of* time-edges $E \subseteq \binom{V}{2} \times T$, *and 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 $\mathbb{G}$.*

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" [20].

2.1.2   Δ-Cliques

A straightforward adaptation of a clique to the temporal setting is to additionally assign a lifetime $I = [a, b]$ to it, that is, the largest time interval such that the clique exists in each time step in said interval. However, this model is often too restrictive for real-world data. For example, if the subject matter of examination is e-mail traffic and the data set includes e-mails with time stamps including seconds, we are not interested in people who sent e-mails to each other every second over a certain time interval, but we would like to know which groups of people were in contact with each other, say, at least 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 data set. The constraint of each pair of vertices being connected in each time step can be relaxed by introducing an

additional parameter Δ, quantifying how many time steps may be skipped between two connections of any vertex pair. These so-called Δ-cliques were introduced by Viard et al. [28, 29] and are formally defined as follows.

Definition 2 (Δ-Clique). *Let $\Delta \in \mathbb{N}$. A Δ-clique in a temporal graph $\mathbb{G} = (V, E, T)$ is a tuple $C = (X, I = [a, b])$ with $X \subseteq V$, $b - a \geq \Delta$, 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 a $(\{v, w\}, t) \in E$ 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 steps 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 data set 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 $\{v_1, v_2\} \in E_\tau$ there is a time step $t \in [\tau, \tau + \Delta]$ such that $(\{v_1, v_2\}, t) \in E$. 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 $b - \Delta$.

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.

We are most interested in Δ-cliques that are not contained in any other Δ-clique. For this we also need to adapt the notion of maximality to the temporal setting [28, 29]. Let $\mathbb{G}$ be a temporal graph. We call a Δ-clique $C = (X, I)$ in $\mathbb{G}$ *vertex-maximal* if we cannot add any vertex to X without having to decrease the clique's lifetime I. That is, there is no Δ-clique $C' = (X', I')$ in $\mathbb{G}$ with $I \subseteq I'$ and $X \subsetneq X'$. We say that a Δ-clique is *time-maximal* if we cannot increase the lifetime I without having to remove vertices from X. That is, there is no Δ-clique $C' = (X', I')$ in $\mathbb{G}$ with $I \subsetneq I'$ and $X \subseteq X'$. We call a Δ-clique *maximal* if it is both vertex-maximal and time-maximal.

2.1.3 Δ-Neighborhood, Δ-Cut, and other Temporal Graph Concepts

In this section, we introduce and define further graph theoretical concepts that need to be adapted to the temporal setting.

We refer to a tuple $(v, I = [a, b])$ with $v \in V$ and $I \subseteq T$ as a *vertex-interval pair* of a temporal graph. We call a the *starting point* of interval I and b the *endpoint* of interval I. Let X be a set of vertex-interval pairs. The modified element relation $(v, I) \sqsubseteq X$ (*temporal membership*) expresses that there exists a vertex-interval pair $(v, I') \in X$ with $I \subseteq I'$.

Using these definitions, we can adapt the notion of a neighborhood of a vertex to temporal graphs. Intuitively, we want that two vertex-interval pairs are neighbors if they can be put into a Δ-clique together.

Definition 3 (Δ-Neighborhood)**.** *For a vertex $v \in V$ and a time interval $I \subseteq T$ in a temporal graph, 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, $b' - a' \geq \Delta$, $I' \subseteq I$, and I' is maximal, that is, there is no time interval $I'' \subseteq I$ with $I' \subset I''$ satisfying the properties above.*

Notice that being a Δ-neighbor of another vertex is a symmetric relation. If $(w, I') \sqsubseteq N^{\Delta}(v, I)$, then we say that w is a vΔ-*neighbor* of v during the time interval I'. In Figure 1, we visualize the concepts of Δ-neighborhood and Δ-clique in a temporal graph. See also Example 1 below.

We need to define a suitable way of intersecting of two sets of vertex-interval pairs, so that, as the intuition suggests, a Δ-clique is just the intersection of the "closed" Δ-neighborhoods[2] of its elements over the lifetime of the clique.

Definition 4 (Δ-Cut)**.** *Let X and Y be two sets of vertex-interval pairs. The Δ-cut $X \sqcap Y$ contains for each vertex, all intersections of intervals in X and Y that are of size at least Δ. More precisely,*

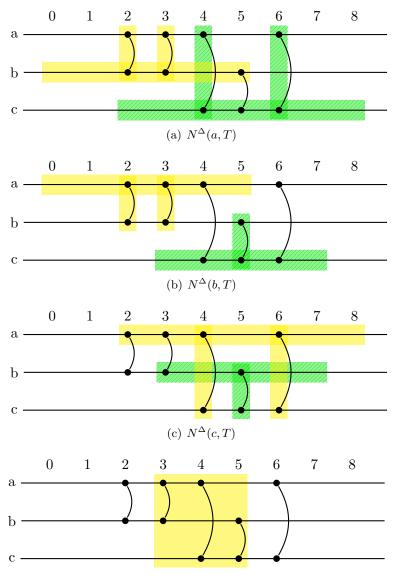$$X \sqcap Y = \{(v, I \cap I') \mid (v, I) \in X \wedge (v, I') \in Y \wedge |I \cap I'| \geq \Delta\}.$$

In other words, the Δ-cut $X \sqcap Y$ contains all vertex-interval pairs (v, I) such that $(v, I) \sqsubseteq X$ and $(v, I) \sqsubseteq Y$, as well as $|I| \geq \Delta$, and I is maximal under these properties. That is, there is no J with $I \subsetneq J$ and $J \subseteq I'$ and $J \subseteq I''$ such that $(v, I') \sqsubseteq X$ and $(v, I'') \sqsubseteq Y$ for some I' and I''.

Example 1. *In Figure 1 we visualize a temporal graph and the concepts of Δ-neighborhood and Δ-clique. We consider a temporal graph $\mathbb{G} = (T, V, E)$ with $T = [0, 8]$, $V = \{a, b, c\}$, $E = \{(\{a, b\}, 2), (\{a, b\}, 3), (\{a, c\}, 4), (\{b, c\}, 5), (\{a, c\}, 6)\}$, and $\Delta = 2$. The vertices are visualized as horizontal lines. The connections between two vertices at a specific time step 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 Figures 1(a)-1(c):

---

[2]In static graphs, the closed neighborhood of a vertex includes the vertex itself.

(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: Δ-Neighborhoods and a Δ-clique of a temporal graph with $\Delta = 2$. The lifetime of the graph is $T = [0, 8]$. The elements of the Δ-neighborhoods in (a), (b), and (c) are shaded in yellow and green (hatched), respectively. A maximal Δ-clique (d) is shaded in yellow.

- *In Figure 1(a), we consider the Δ-neighborhood $N^\Delta(a, T)$ of vertex a during the whole time interval T. The yellow shaded 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 hatched green.*

- *In Figure 1(b), we visualize the Δ-neighborhood $N^\Delta(b,T)$ of b over the whole lifetime T of the temporal graph. The vertex-interval pair $(c,[3,7]) \in N^\Delta(b,T)$ is marked in hatched green. The vertex-interval pair $(a,[0,5]) \in N^\Delta(b,T)$ is shaded 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 $N^\Delta(c,T)$ of c over the whole lifetime T of the temporal graph. The vertex-interval pair $(b,[3,7]) \in N^\Delta(c,T)$ is marked in hatched green. The vertex-interval pair $(a,[2,8]) \in N^\Delta(c,T)$ is shaded 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, see Figures 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. Further nontrivial 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 $(\{a\},[0,8])$, $(\{b\},[0,8])$, and $(\{c\},[0,8])$.

2.2 Bron-Kerbosch Algorithm

In this section, we explain the basic idea of the (static) Bron-Kerbosch algorithm. We also present two techniques known from the literature which improve the running time of the algorithm.

The Bron-Kerbosch algorithm [3] enumerates all maximal cliques in undirected, static graphs. It is a widely used recursive backtracking algorithm which is easy to implement and more efficient than alternative algorithms in many practical applications [4].

The Bron-Kerbosch algorithm, see Algorithm 1, receives three disjoint vertex-sets as an input: P, R, and X. The set R induces 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 contains the vertices that have not been considered yet whereas the set X includes all vertices that have already been considered in earlier steps. In each recursive 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 and therefore, the clique is maximal and can be added to the solution. Otherwise, the clique is not maximal because at least one vertex exists that

Algorithm 1 Enumerating all Maximal Cliques

1: **function** BronKerbosch(P, R, X)
    ▷ R : a clique
    ▷ $P \cup X$: set of all vertices v such that $R \cup \{v\}$ is a clique and where
        • vertices in P have not yet been considered as additions to R and
        • vertices in X already have been considered in earlier steps
2:    **if** $P \cup X = \emptyset$ **then**
3:        add R to the solution
4:    **end if**
5:    **for** $v \in P$ **do**
6:        BronKerbosch$(P \cap N(v), R \cup \{v\}, X \cap N(v)))$
7:        $P \leftarrow P \setminus \{v\}$
8:        $X \leftarrow X \cup \{v\}$
9:    **end for**
10: **end function**

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, 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 is initially called with $P = V$ and $R = X = \emptyset$.

2.2.1 Pivoting

Bron and Kerbosch [3] introduced a method to increase the efficiency of the basic algorithm by choosing a pivot element to decrease the number of recursive calls. It is based on the observation that for any vertex $u \in P \cup X$ either u itself or one of its non-neighbors must be contained in any maximal clique containing R. This is true since if neither u nor one of the non-neighbors of u are included in a clique containing R, then 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 R. Hence, if we modify the Bron-Kerbosch algorithm (Algorithm 1) so that we choose an arbitrary pivot element $u \in P \cup X$ and iterate only over u and all its non-neighbors, then we still enumerate all maximal cliques containing R but decrease the number of recursive calls in the for-loop of Algorithm 1. Tomita et al. [25] have shown that if u is chosen from $P \cup X$ such that u has the most neighbors in P, then all maximal cliques of a graph $G = (V, E)$ are enumerated in $O(3^{|V|/3})$ time, see Algorithm 2.

Algorithm 2 Enumerating all Maximal Cliques in a Graph with Pivoting
---
1: **function** BRONKERBOSCHPIVOT(P, R, X)

$\quad\triangleright R : a\ clique$

$\quad\triangleright P \cup X : set\ of\ all\ vertices\ v\ such\ that\ R \cup \{v\}\ is\ a\ clique\ and\ where$

$\qquad\bullet\ vertices\ in\ P\ have\ not\ yet\ been\ considered\ as\ additions\ to\ R\ and$

$\qquad\bullet\ vertices\ in\ X\ already\ have\ been\ considered\ in\ earlier\ steps$

2:     **if** $P \cup X = \emptyset$ **then**

3:         add R to the solution

4:     **end if**

5:     choose pivot vertex $u \in P \cup X$ with $|P \cap N(u)| = \max\limits_{v \in P \cup X} | P \cap N(v) |$

6:     **for** $v \in P \setminus N(u)$ **do**

7:         BRONKERBOSCHPIVOT($P \cap N(v), R \cup \{v\}, X \cap N(v)$))

8:         $P \leftarrow P \setminus \{v\}$

9:         $X \leftarrow X \cup \{v\}$

10:     **end for**

11: **end function**

2.2.2   Degeneracy of a Graph

Degeneracy is a measure of graph sparsity. Real-world instances of static graphs (especially social networks) are often sparse, resulting in a small degeneracy value [4]. This motivates a modification of the Bron-Kerbosch algorithm which we present in this section and the complexity analysis of this algorithm parameterized by the degeneracy of the input graph. The degeneracy of a graph is defined as follows.

Definition 5 (Degeneracy). *The* degeneracy *of a static graph G is defined as the smallest integer $d \in \mathbb{N}$ such that each subgraph G' of G contains a vertex v with degree at most d.*

If a graph has degeneracy d, we also call it *d-degenerated*. It is easy to see that the maximal clique size of a d-degenerated graph is at most $d + 1$: If there is a clique of size at least $d + 2$, then the vertices of this clique would form a subgraph in which every vertex v of the clique has a degree larger than d. For each d-degenerated graph there is a *degeneracy ordering*, which is a linear ordering of the vertices with the property that for every vertex v we have that at most d of its neighbors occur at a later position in the ordering. The degeneracy d and a corresponding degeneracy ordering for a graph $G = (V, E)$ can be computed in linear time [4]: For graph G, the vertex with the smallest degree is selected 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 and a corresponding degeneracy ordering is the order in which the vertices were removed from the graph.

Algorithm 3 Enumerating all Maximal Cliques in a Graph with Degeneracy Ordering

---

1: **function** BronKerboschDeg(P, R, X)
    ▷ *R : a clique*
    ▷ *$P \cup X$: set of all vertices v such that $R \cup \{v\}$ is a clique and where*
        • *vertices in P have not yet been considered as additions to R and*
        • *vertices in X already have been considered in earlier steps*
2:     **for** v_i in a degeneracy odering $v_0, v_1, \ldots, v_n$ of $G = (V, E)$ **do**
3:         $P \leftarrow N(v_i) \cap \{v_{i+1}, \ldots, v_{n-1}\}$
4:         $X \leftarrow N(v_i) \cap \{v_0, \ldots, v_{i-1}\}$
5:         BronKerboschPivot$(P, \{v_i\}, X)$
6:     **end for**
7: **end function**

For a graph $G = (V, E)$ with degeneracy d, Eppstein et al. [4] showed that using the degeneracy ordering of G in the outer-most recursive call and afterwards using pivoting, all maximal cliques can be enumerated in $O(d \cdot |V| \cdot 3^{d/3})$ time, see Algorithm 3. In other words, enumerating maximal cliques is fixed-parameter tractable with respect to the parameter degeneracy d of the input graph.

3   Bron-Kerbosch Algorithm for Temporal Graphs

We adapt the static Bron-Kerbosch algorithm to the temporal setting to enumerate all Δ-cliques, see Algorithm 4. The input of the algorithm consists of two sets P and X of vertex-interval pairs as well as a tuple $R = (C, I)$, where C is a set of vertices and I a time interval. The idea is that in every recursive call of the algorithm, R is a time-maximal Δ-clique, and the sets P and X contain vertex-interval pairs that are in the Δ-neighborhood of every vertex in C during an interval $I' \subseteq I$. In particular, $P \cup X$ includes all vertex-interval pairs (v, I) for which $(C \cup \{v\}, I)$ is a time-maximal Δ-clique. 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 maximal Δ-clique (C', I'') with $C \cup \{v\} \subseteq C'$ and $I'' \subseteq I'$ has already been detected in earlier steps.

We show below that if $\forall (w, I') \in P \cup X : I' \subsetneq I$, then there is no vertex v that forms a Δ-clique together with C over the whole time interval I. Consequently, $R = (C, I)$ is a maximal Δ-clique.

In one 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', that is, $P \sqcap N^{\Delta}(v, I')$ and $X \sqcap N^{\Delta}(v, I')$. For the set P' for example, we get a set of all time-maximal

11

Algorithm 4 Enumerating all Maximal Δ-Cliques

1: **function** BronKerboschDelta$(P, R = (C, I), X)$
   ▷ $R = (C, I)$: *time-maximal Δ-clique*
   ▷ $P \cup X$: *set of all vertex-interval pairs (v, I') such that $I' \subseteq I$ and $(C \cup \{v\}, I')$ is a time-maximal Δ-clique and where*

   - *vertex-interval pairs in P have not yet been considered as additions to R and*

   - *vertex-interval pairs in X already have been considered in earlier steps*

2:      **if** $\forall (w, I') \in P \cup X : I' \subsetneq I$ **then**
3:         add R to the solution
4:      **end if**
5:      **for** $(v, I') \in P$ **do**
6:         $R' \leftarrow (C \cup \{v\}, I')$
7:         $P' \leftarrow P \sqcap N^\Delta(v, I')$
8:         $X' \leftarrow X \sqcap N^\Delta(v, I')$
9:         BronKerboschDelta(P', R', X')
10:        $P \leftarrow P \setminus \{(v, I')\}$
11:        $X \leftarrow X \cup \{(v, I')\}$
12:      **end for**
13: **end function**
---

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 $\mathbb{G} = (V, E, T)$ and a given time period Δ, the *initial call* of Algorithm 4 to enumerate all maximal Δ-cliques in graph $\mathbb{G}$ is made with $P = \{(v, T) \mid v \in V\}$, $R = (\emptyset, T)$ and $X = \emptyset$. In the remainder of this document we will always assume that BronKerboschDelta is initially called with those inputs.

3.1 Analysis

In the following, we prove the correctness of the algorithm and analyze its running time. We start with arguing that the sets P and X behave as claimed.

Lemma 1. *For each recursive call of* BronKerboschDelta *with $R = (C, I)$ and $C \neq \emptyset$, it holds that $P \cup X = \bigsqcap_{v \in C} N^\Delta(v, I)$.*

Proof. We prove this by induction on the recursion depth, that is, the number $|C|$ of vertices in the clique in the current recursive call. In the initial call we have that $C = \emptyset$. In each iteration of the first call we have that $P \cup X = \{(v, T) \mid v \in V\}$ since, whenever a vertex-interval pair is removed from P, then it is added to X, and initially $P = \{(v, T) \mid v \in V\}$. For every recursive call of BronKerboschDelta with $R' = (C', I')$, P', and X', and $C' = \{v\}$ for some vertex v we have that $P' = P \sqcap N^\Delta(v, I')$ and $X' = X \sqcap N^\Delta(v, I')$. Hence, we get

$$P' \cup X' = \{(v, T) \mid v \in V\} \sqcap N^\Delta(v, I') = N^\Delta(v, I').$$

Now we assume that we are in a recursive call of BronKerboschDelta with $R = (C, I)$, P, and X, where $|C| > 1$. By the induction hypothesis we know that $P \cup X = \bigsqcap_{v \in C} N^\Delta(v, I)$. Let $(v, I') \in P$ be the vertex added to the Δ-clique, that is, in the next recursive call we have that $R' = (C', I')$, with $C' = C \cup \{v\}$, and $P' = P \sqcap N^\Delta(v, I')$ as well as $X' = X \sqcap N^\Delta(v, I')$. Then,

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

This proves the claim. $\qquad\square$

Next, we show that the set R behaves as claimed, that is, R is indeed a time-maximal Δ-clique in each recursive call of BronKerboschDelta.

Lemma 2. *In each recursive call of* BronKerboschDelta, $R = (C, I)$ *is a time-maximal Δ-clique.*

Proof. We show by induction on the recursion depth that $R = (C, I)$ is a time-maximal Δ-clique and that all vertex-interval pairs (v, I') in P are Δ-neighbors during I' to all vertices in the Δ-clique R and that I' is maximal under this property. The algorithm is initially called with $R = (\emptyset, T)$, which is a trivial time-maximal Δ-clique, and $P = \{(v, T) \mid v \in V\}$, which fulfills the desired property since the initial Δ-clique is empty and T is the maximum time interval. In each recursive call BronKerboschDelta is called with $(P \sqcap N^\Delta(v, I'), (C \cup \{v\}, I'), X \sqcap N^\Delta(v, I'))$ for some $(v, I') \in P$. By the induction hypothesis, v is a Δ-neighbor to all vertices in C during time interval I', and I' is maximal. Hence, $(C \cup \{v\}, I')$ is a time-maximal Δ-clique. Furthermore, each vertex-interval pair (v', I'') in $P \sqcap N^\Delta(v, I')$ is in the Δ-neighborhood of each vertex-interval pair (v'', I') with $v'' \in C \cup \{v\}$,

since it is both in P and hence in the Δ-neighborhood of each vertex in C and in $N^\Delta(v, I')$. The maximality of I' follows from the fact that the Δ-cut and Δ-neighborhood operations preserve maximality of intervals by definition. $\qquad\square$

Now we can prove the correctness of the algorithm.

Theorem 1 (Correctness of Algorithm 4). *Let $\mathbb{G} = (V, E, T)$ be a temporal graph. If algorithm* BRONKERBOSCHDELTA(P, R, X) *is run on input* $(V \times \{T\}, (\emptyset, T), \emptyset)$, *then it adds all maximal Δ-cliques of $\mathbb{G}$, and only these, to the solution.*

Proof. Let $R^* = (C^*, I^*)$ be a maximal Δ-clique with $|C^*| > 1$. For a recursive call of BRONKERBOSCHDELTA on (P, R, X), say that a vertex is a *candidate*, if there is an interval I with $I^* \subseteq I$ such that $(v, I) \in P$. We show by induction on $|C^*| - \ell$, that for each $\ell = 0, 1, \ldots, |C^*|$ there is a recursive call of BRONKERBOSCHDELTA on $(P, R = (C, I), X)$ with $C \subseteq C^*$ and $\ell = |C^* \setminus C|$ candidates.

Clearly, in the initial call, $C = \emptyset \subseteq C^*$ and each vertex in C^* is a candidate. Now assume that there is a recursive call with $(P, R = (C, I), X)$ and $C \subseteq C^*$, and with $\ell - 1 = |C^* \setminus C|$ candidates. Consider the for-loop in that recursive call and consider the first vertex-interval pair (v, I') in that loop in which v is a candidate and $I^* \subseteq I'$. BRONKERBOSCHDELTA proceeds with a recursive call on $(P \sqcap N^\Delta(v, I'), R' = (C \cup \{v\}, I'), X \sqcap N^\Delta(v, I'))$. Observe that each candidate except v remains a candidate also in this recursive call. Furthermore, $|C^* \setminus (C \cup \{v\})| = \ell$. Thus, by induction there is a recursive call with the sets (P, R, X) in which $R^* = R$. Since R^* is maximal by assumption, for each vertex-interval pair $(w, I'') \in \prod_{v \in C^*} N^\Delta(v, I^*)$ we have $I'' \subsetneq I^*$. By Lemma 1 we have $\forall(w, I') \in P \cup X : I' \subsetneq I^*$ and hence, R^* is added to the solution.

Now assume that $R = (C, I)$ is added to the solution. By Lemma 2, R is a time-maximal Δ-clique. By Lemma 1 and since $P \cup X = \emptyset$, there is no vertex that can be added to R. Hence, R is a maximal Δ-clique. $\qquad\square$

Next, we analyze the running time of BRONKERBOSCHDELTA. We start with the following observation.

Lemma 3. *For every time-maximal Δ-clique R of a temporal graph $\mathbb{G} = (V, E, T)$, there is at most one recursive call of* BRONKERBOSCHDELTA *with R as an input.*

Proof. Assume that there are two recursive calls A and B of BRONKERBOSCHDELTA with the same $R = (C, I)$. Let $R' = (C', I')$, with $C' \subset C$ and $I \subseteq I'$, occur in the recursive call corresponding to the closest common ancestor of the recursive calls A and B in the recursion tree. Hence,

there are two vertex-interval pairs $(v, J), (w, J') \in P$ that lead to the calls A and B, respectively, in the for loop.

Consider the case $v = w$. Then, J and J' must overlap in at least Δ time steps, because $I \subseteq J, J' \subseteq I'$. However, P is contained in the Δ-cut of the Δ-neighborhoods of C' over I' and thus, for each vertex no two vertex-interval pairs in P overlap in Δ time steps, a contradiction.

Now consider the case $v \neq w$. Without loss of generality due to symmetry assume that (v, J) is processed first in the for loop. Then, when processing (w, J'), pair (v, J) has been added to X. This is a contradiction to the fact that recursive call B outputs R, that is, it outputs a clique with time interval $I \subseteq J$.

Hence, we have that there cannot be two recursive calls of BRONKERBOSCHDELTA with $R = (C, I)$. $\square$

Now we upper-bound the running time for computing a Δ-cut.

Lemma 4. *Let X and Y be two sets of vertex-interval pairs with the following properties.*
- *For every $(v, I) \in X \cup Y$ we have that $|I| \geq \Delta$,*
- *for every $(v, I) \in X$ and $(v, I') \in X$ we have that $|I \cap I'| < \Delta$,*
- *for every $(v, I) \in Y$ and $(v, I') \in Y$ we have that $|I \cap I'| < \Delta$,*
- *X and Y are sorted lexicographically by first the vertex and then the starting point of the interval.*

Then the Δ-cut $X \sqcap Y$ can be computed in $O(|X| + |Y|)$ time such that it is also sorted lexicographically by first the vertex and then the starting point of the interval.

Proof. The Δ-cut $X \sqcap Y$ of two sets of vertex-interval pairs X and Y can be computed in the following way.

For every vertex v, we do the following:
1. Select the first vertex-interval pairs (v, I) and (v, I') from X and Y, respectively.
2. If $|I \cap I'| > \Delta$, then add $(v, I \cap I')$ to the output (the Δ-cut). If the endpoint of I' is smaller than the endpoint of I, then replace (v, I') with the next vertex-interval pair in Y, otherwise replace (v, I) with the next vertex-interval pair in X.
3. Repeat Step 2 until all vertex-interval pairs containing vertex v are processed.

Note that the intervals for each vertex v are added to the output in order of their starting point. Furthermore, by construction of the algorithm we have that for each (v, I) in the output, (v, I) is also in the Δ-cut $X \sqcap Y$. It remains to show that for all $(v, I) \in X$ and $(v, I') \in Y$ with $|I \cap I'| \geq \Delta$ we have that $(v, I \cap I')$ is included in the output. Let $I = [a, b]$ and $I' = [a', b']$. At some point, the procedure processes in Step 2 for the first time one of $(v, I) \in X$ or $(v, I') \in Y$. Without loss of generality, let $(v, I) \in X$ be

15

processed first. If at the same time also $(v, I') \in Y$ is processed, clearly, $(v, I \cap I')$ is added to the output, as required. Now assume that Step 2 processes some other vertex-interval pair $(v, I'' = [a'', b'']) \in Y$, $a'' < a'$, together with $(v, I) \in X$. Since $|I \cap I'| \geq \Delta$ and $|I' \cap I''| < \Delta$ we have that $b'' < b$ and hence, (v, I) is not replaced in this step. Consequently, the procedure eventually adds $(v, I \cap I')$ to the output.

In each step of the procedure at least one new vertex-interval pair is processed and each vertex-interval pair in X and Y is only processed once. Hence, the running time is in $O(|X| + |Y|)$. $\square$

Lemmata 3 and 4 allow us to upper-bound the running time of BRONKER-BOSCHDELTA depending on the number of different time-maximal Δ-cliques of the input graph.

Theorem 2. *Let* $\mathbb{G} = (V, E, T)$ *be a temporal graph with* x *distinct time-maximal* Δ-*cliques. Then* BRONKERBOSCHDELTA *enumerates all* maximal Δ-*cliques in* $O(x \cdot |E| + |E| \cdot |T|)$ *time.*

Proof. We assume that all edges of the temporal graph are sorted by their time stamp. Note that this can be done in a preprocessing step in $O(|E| \cdot |T|)$ time using Counting Sort. Furthermore, we assume that for each vertex v, the Δ-neighborhood $N^\Delta(v, T)$ is given. These neighborhoods can be pre-computed in $O(|E|)$ time, assuming that the edges are sorted by their time stamps.

By Lemma 3 we know that for each time-maximal Δ-clique there is at most one recursive call of BRONKERBOSCHDELTA. By charging the computation of P', R', and X' to the corresponding recursive call, for each recursive call we compute a constant number of Δ-neighborhoods and Δ-cuts. The size of the sets P, X, and any Δ-neighborhood is upper-bounded by $|E|$ and each of these sets has the property that for every (v, I) and (v, I') out of the same set we have that $|I \cap I'| < \Delta$. Given $N^\Delta(v, T)$, $N^\Delta(v, I)$ can be computed in $O(|E|)$ time for any I and by Lemma 4, a Δ-cut can be computed in $O(|E|)$ time. Hence, all maximal Δ-cliques can be enumerated in $O(x \cdot |E| + |E| \cdot |T|)$ time. $\square$

We now use a general upper bound for the number of time-maximal Δ-cliques in a temporal graph to bound the overall running time of BRONKER-BOSCHDELTA.

Corollary 1. *Let* $\mathbb{G} = (V, E, T)$ *be a temporal graph.* BRONKERBOSCHDELTA *enumerates all maximal* Δ-*cliques of* $\mathbb{G}$ *in* $O(2^{|V|} \cdot |T| \cdot |E|)$ *time.*

Proof. Note that the vertex set of each maximal Δ-clique induces a static clique in the static graph G underlying $\mathbb{G}$ that has an edge between two vertices if and only if there is a time-edge in $\mathbb{G}$ between these vertices at some time step. Furthermore, for each clique in G, there are at most $|T|$

maximal Δ-cliques because their time intervals are pairwise not contained in one-another. Hence, the number of time-maximal Δ-cliques of any temporal graph is upper-bounded by $2^{|V|} \cdot |T|$. By Theorem 2, we get an overall running time in $O(2^{|V|} \cdot |T| \cdot |E|)$. $\qquad\square$

3.2 Pivoting

In this section, we explain how we can decrease the number of recursive calls of BRONKERBOSCHDELTA by using pivoting. Recall that the idea of pivoting in the classic Bron-Kerbosch algorithm for static graphs is based on the observation that for any vertex $u \in P \cup X$ either u itself or one of its non-neighbors must be contained in any maximal clique containing R. Vertex u is also called *pivot*.

A similar observation holds for maximal Δ-cliques in temporal graphs. Instead of vertices, however, we now choose vertex-interval pairs as pivots: For any $(v_p, I_p) \in P \cup X$ and any maximal Δ-clique $R_{\max} = (C_{\max}, I_{\max})$ with $I_{\max} \subseteq I_p$, either vertex v_p or one vertex $w \neq v_p$ which is not a Δ-neighbor of v_p during the time $I_{\max}$, that is, $(w, I_{\max}) \notin N^{\Delta}(v_p, I_p)$, must be contained in $C_{\max}$.

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, see Algorithm 5. In other words, we do not have to make a recursive call for any $(w, I') \in P$ which holds $(w, I') \sqsubseteq N^{\Delta}(v_p, I_p)$.

In Figure 2 we give an illustrative example for pivoting. In this example, we assume that the algorithm runs on a temporal graph such that the set $P = \{(a, [0,8]), (b, [0,4]), (c, [1,3]), (c, [5,8])\}$ occurs within a recursive call of BRONKERBOSCHDELTAPIVOT. For simplicity, we show in Figure 2(a) only the subgraph containing the elements of P and the relation between these elements rather than displaying the whole graph. In Figure 2(b), we choose element $(a, [0,8])$ (hatched) as pivot. It can be seen that the elements $(b, [0,4])$ and $(c, [5,8])$ lie completely in the Δ-neighborhood (dotted) of the pivot, that is, $(b, [0,4]), (c, [5,8]) \sqsubseteq N^{\Delta}(a, [0,8])$. These two elements can therefore be left out in the iteration over the elements in P of the BRONKERBOSCHDELTA. We only have to iterate over the pivot $(a, [0,8])$ and the element $(c, [1,3])$ which is not completely in the Δ-neighborhood of our chosen pivot. In Figure 2(c), we can see that for every maximal Δ-clique (C, I) with respect to P either $a \in C$, $I \subseteq [0,8]$ or $c \in C$, $I \subseteq [1,3]$. The figure hence shows that iterating over the elements $(b, [0,4])$ and $(c, [5,8])$ will not find any maximal Δ-clique that we do not find via one of the elements $(a, [0,8])$ and $(c, [1,3])$.

Next, we formally prove the correctness of this procedure.

Proposition 1. *For each Δ-clique $R = (C, I)$ and a pivot element $(v_p, I_p) \in P \cup X$, the following holds: for every $R_{\max} = (C_{\max}, I_{\max})$ with $C \subset C_{\max}$*
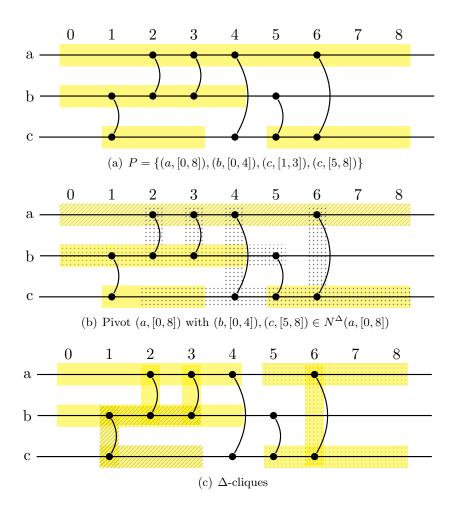
17

(a) $P = \{(a, [0, 8]), (b, [0, 4]), (c, [1, 3]), (c, [5, 8])\}$

(b) Pivot $(a, [0, 8])$ with $(b, [0, 4]), (c, [5, 8]) \in N^\Delta(a, [0, 8])$

(c) Δ-cliques

Figure 2: A exemplary set P of BRONKERBOSCHDELTAPIVOT, a possible pivot element (hatched) including its Δ-neighborhood (dotted), and all maximal Δ-cliques with respect to set P, $\Delta = 2$.

and $I_{\max} \subseteq I_p \subseteq I$ it either holds that $v_p \in C_{\max}$ or otherwise 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 that $v_p \notin C_{\max}$ and for each $w \in C_{\max}$ it holds that $(w, I_{\max}) \sqsubseteq N^\Delta(v_p, I_p)$. Consequently, for each $w \in C_{\max} \setminus C$ there exists a $(w, I') \in P \cup X$ with $I_{\max} \subseteq I'$. Because v_p 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 $R_{\max}$, yielding another Δ-clique with at least the same lifetime. This is a contradiction to the assumption that $R_{\max}$ is maximal. $\square$

Algorithm 5 Enumerating all Maximal Δ-Cliques in a Temporal Graph with Pivoting
---
1: **function** BRONKERBOSCHDELTAPIVOT$(P, R = (C, I), X)$
      ▷ $R = (C, I)$: *time-maximal Δ-clique*
      ▷ $P \cup X$: *set of all (v, I') s.t. $I' \subseteq I$ and $(C \cup \{v\}, I')$ is a time-maximal*
      Δ-clique
2:     **if** $\forall (w, I') \in P \cup X \colon I' \subsetneq I$ **then**
3:        add R to the solution
4:     **end if**
5:     choose pivot element $(v_p, I_p) \in P \cup X$
6:     **for** $(v, I') \in P \setminus \{(w, I'') \mid (w, I'') \in P \wedge (w, I'') \sqsubseteq N^\Delta(v_p, I_p)\}$ **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:      BRONKERBOSCHDELTAPIVOT(P', R', X')
11:      $P \leftarrow P \setminus (v, I')$
12:      $X \leftarrow X \cup (v, I')$
13:     **end for**
14: **end function**

An optimal pivot element is chosen in such a way that it minimizes the number of recursive calls. It is the element in the set $P \cup X$ having the largest number of elements in P 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:

For each Δ-clique $R = (C, I)$ in a recursive call of the algorithm, choosing a pivot element $(v_p, I_p) \in P \cup X$ only affects maximal Δ-cliques $R_{\max} = (C_{\max}, I_{\max})$ fulfilling $I_{\max} \subseteq I_p$. Moreover, 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).

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

Input: A set J of jobs j with a time interval I_j and a weight w_j each.

Task: Find a subset of jobs $J' \subseteq J$ that maximizes $\sum_{j \in J'} w_j$ such 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 $w_{(v,I')} = |\{(v, I) \mid (v, I) \in P \wedge (v, I) \sqsubseteq N^\Delta(v, I')\}|$. This problem can be solved efficiently in $O(\min(|E|, |V| \cdot |T|) \cdot \log(\min(|E|, |V| \cdot |T|)))$ time by using dynamic programming [15, Chapter 6.1] under the assumption that the weights of the potential pivot elements are known.

4 Degeneracy of Temporal Graphs

Recall from Section 2.2.2 that one can upper-bound the running time of the static Bron-Kerbosch algorithm using the degeneracy of the input graph. The degeneracy of a graph G is the smallest integer d such that every non-empty subgraph of G contains a vertex of degree at most d. We now give an analogue for the temporal setting, motivated by the fact that static graphs are often sparse in practice as measured by small degeneracy [4]. Intuitively, we want to capture the fact that a temporal graph keeps its degeneracy value during its whole lifetime.

Definition 6 (Δ-slice degeneracy). *A temporal graph $\mathbb{G} = (V, E, T)$ has Δ-slice degeneracy d if for all $t \in T$ we have that the graph $G_t = (V, E_t)$, where $E_t = \{\{v, w\} \mid (\{u, w\}, t') \in E$ for some $t' \in [t, t+\Delta]\}$, has degeneracy at most d.*

Using the parameter Δ-slice degeneracy, we can upper-bound the number of time-maximal Δ-cliques of a temporal graph.

Lemma 5. *Let $\mathbb{G} = (V, E, T)$ be a temporal graph with Δ-slice degeneracy d. Then, the number of time-maximal Δ-cliques in $\mathbb{G}$ is at most $3^{d/3} \cdot 2^{d+1} \cdot |V| \cdot |T|$.*

Proof. Let $\mathbb{G} = (V, E, T)$ be a temporal graph with Δ-slice degeneracy d. Then we call the graph $G_t = (V, E_t)$, where $E_t = \{\{v, w\} \mid (\{u, w\}, t') \in E$ for some $t' \in [t, t + \Delta]\}$ a Δ-*slice* of $\mathbb{G}$ at time t. The vertex set of each time-maximal Δ-clique which starts at time t is also a clique in G_t, otherwise there would be two vertices which are disconnected for more than Δ time-steps. Since G_t has degeneracy at most d, the number of maximal cliques of G_t is upper-bounded by $3^{d/3} \cdot |V|$ [4]. Furthermore, the maximum size of a clique is upper-bounded by $d + 1$. Hence, the total number of cliques is upper-bounded by $3^{d/3} \cdot 2^{d+1} \cdot |V|$. Note that for each of those cliques we have at most one time-maximal Δ-clique starting at time t. Hence, the total number of Δ-cliques is at most $3^{d/3} \cdot 2^{d+1} \cdot |V| \cdot |T|$. $\square$

Lemma 5 now allows us to bound the running time of Algorithm 4 using the Δ-slice degeneracy d of the input graph $\mathbb{G}$.

Theorem 3. *Let* $\mathbb{G} = (V, E, T)$ *be a temporal graph with Δ-slice degeneracy d. Then,* BronKerboschDelta *enumerates all Δ-cliques of $\mathbb{G}$ in* $O(3^{d/3} \cdot 2^d \cdot |V| \cdot |T| \cdot |E|)$ *time.*

Proof. By Lemma 5 we know that the number of time-maximal Δ-cliques in a temporal graph with Δ-slice degeneracy d is at most $3^{d/3} \cdot 2^{d+1} \cdot |V| \cdot |T|$. Hence, by Theorem 2, we get an overall running time in $O(3^{d/3} \cdot 2^d \cdot |V| \cdot |T| \cdot |E|)$. $\qquad\square$

Note that Theorem 3 implies that enumerating all maximal Δ-cliques is fixed-parameter tractable with respect to the parameter Δ-slice degeneracy. Hence, while NP-hard in general, the problem can be solved efficiently if the Δ-slice degeneracy of the input graph is small.

5 Experimental Results

In this section we present our experimental results. We give the Δ-slice degeneracy of several real-world temporal graphs for several values for Δ. Then we show the behavior of our implementation of BronKerboschDeltaPivot (Algorithm 5) applied to these real-world temporal graphs and compare it to the algorithm implemented by Viard et al. [29].

5.1 Setup and Statistics

We now give details of the implementation and the used reference algorithm, and introduce the data sets we used in the experiments. Furthermore, we explain how the values of Δ were chosen, give some statistics for the data set, and calculate the Δ-slice degeneracy of the data sets for the chosen values of Δ.

Implementation. We implemented[3] BronKerboschDeltaPivot with slight modifications that allow the algorithm to use multiple pivot elements (we refer to this version as BronKerboschDeltaPivot*). Furthermore, we implemented a simple algorithm to compute the Δ-slice degeneracy. Both implementations are in Python 2.7.12 and all experiments were carried out on an Intel Xeon E5-1620 computer with four cores clocked at 3.6 GHz and 64 GB RAM. We did not utilize the parallel-processing capabilities although it should be easy to achieve almost linear speed-up with growing number of cores due to the simple nature of BronKerboschDeltaPivot. The operating system was Ubuntu 16.04.4 with Linux kernel version 4.4.0-57. We compared BronKerboschDeltaPivot* with the algorithm by Viard et al. [29] which was also implemented in Python. We modified their

---

[3]Code freely available at `http://fpt.akt.tu-berlin.de/temporalcliques/` (GNU General Public License).

source code[4] by removing the text output in their implementation in order to avoid speed differences. We call their algorithm Algorithm VLM below.

Data Sets. We chose several freely available real-world temporal graphs aiming for an overview over the different kinds of contexts in which such graphs arise, that is, an overview over different modes of communication and different kinds of entities and environments in which this communication takes place. However, a focus is on temporal graphs based on physical proximity of individuals, since previous work on Δ-cliques also focused on these [28, 29]. The contexts and sources of our test set of temporal graphs are as follows:

- internet-router communication: as733 [18],

- email communication: karlsruhe [7],

- social-network communication: facebook-like [23], and

- physical-proximity[5] between

  - high school students: highschool-2011, highschool-2012, highschool-2013 [1, 6, 24],

  - patients and health-care workers: hospital-ward [27],

  - attendees of the ACM Hypertext 2009 conference: hypertext [13],

  - attendees of the Infectious SocioPatterns event: infectious [13], and

  - children and teachers in a primary school: primaryschool [24].

Table 1 contains the number of vertices, edges, temporal resolution, and lifetime of the corresponding temporal graphs. As a time step we fixed one second for each of the data sets. Viard et al. [29], as the first work on enumerating Δ-cliques, used the data set highschool-2012 in their experiments.

Chosen values of Δ. In order to limit the influence of time scales in the data and to make running times comparable between instances, as well as to be able to present the results in a unified way, we chose the Δ-values as follows. We decided on a reference point of the *edge appearance rate* that is, of the average number of edges per time step and we fixed a set of Δ-values for this reference point. For each considered instance we then scaled the reference Δ-values proportionally to the quotient of the reference edge appearance rate and the edge appearance rate in the instance.

---

[4]Code freely available at `https://github.com/TiphaineV/delta-cliques` .

[5]Available at `http://www.sociopatterns.org/datasets` .

Table 1: Statistics for the data sets used in our experiments.

Instance	Vertices	Edges	Resolution	Lifetime (s)
as733	7,716	11,410,810	1d	67,824,000
facebook-like	1,899	59,835	1s	16,736,181
highschool-2011	126	28,560	20s	272,330
highschool-2012	180	45,047	20s	729,500
highschool-2013	327	188,508	20s	363,560
hospital-ward	75	32,424	20s	347,500
hypertext	113	20,818	20s	212,340
infectious	10,972	415,912	20s	6,946,340
karlsruhe	1,870	461,661	1s	123,837,267
primaryschool	242	125,773	20s	116,900

As the reference point we chose the edge appearance rate of 1/5 edges per time step; this value was chosen for convenience within the interval of edge-appearance rates in the studied data sets (see Table 1). Since, intuitively, the Δ-values of interest in practice increase exponentially, we chose as reference Δ-values the numbers 0 and 5^i for $i = 1, 2, \ldots$. As mentioned, for each instance, these values are then multiplied by the quotient of edge appearance rates. That is, if the instance has m edges and lifetime ℓ, then we scaled the reference Δ-values by the factor $(1/5)/(m/\ell) = \ell/(5m)$. For example, for highschool-2012 we obtain the Δ-values $\{0, 80, 404, 2024, 10121, 50606, 253034, \ldots\}$. For reference, recall that each time step in highschool-2012 corresponds to one second (a day has 86,000 seconds and a week has 604,800 seconds). In figures, we refer to each scaled value of Δ by $\Delta \sim 5^i$ for some concrete i. Viard et al. [28] used Δ-values according to 60 seconds, 15 minutes, 1 hour and 3 hours.

Δ-Slice Degeneracy. The Δ-slice degeneracies for our set of instances are shown in Table 2 together with the static degeneracy of the underlying static graph which has an edge whenever there is an edge at some time step in the temporal graph. Clearly, as the value of Δ increases, the Δ-slice degeneracy approaches—and is upper-bounded by—the static degeneracy. The static degeneracy of our instances is small in comparison with the size of the graph. This falls in line with the analysis by Eppstein et al. [4] for many real-world graphs. Moreover, for many practically relevant values of Δ the Δ-slice degeneracy is still significantly smaller. For example, in the instance highschool-2012, the scaled value of Δ corresponding to 5^3 equals 2204 time steps (seconds) and the corresponding Δ-slice degeneracy is 5. This indicates that Δ-slice degeneracy can be a very promising (that is, also small) parameter when designing and analyzing algorithms for temporal graphs.

We computed the Δ-slice degeneracies using a straightforward approach.

Table 2: Static degeneracy and Δ-slice degeneracy. Empty cells indicate that the lifetime of the temporal graph is smaller than the scaled Δ-value.

Instance	Static	$\Delta = 0$	$\sim 5^3$	$\sim 5^5$	$\sim 5^7$	$\sim 5^9$
as733	24	13	13	14	15	24
facebook-like	20	1	3	6	19	
highschool-2011	21	4	7	11	19	
highschool-2012	18	4	5	6	12	
highschool-2013	24	4	5	9	14	
hospital-ward	22	4	6	11	18	
hypertext	28	6	7	8	22	
infectious	18	4	9	18	18	18
karlsruhe	33	2	6	9	17	32
primaryschool	47	4	4	10	31	

We iteratively computed for each Δ-long time interval the graph induced by the edges in that time interval. For each of these graphs we computed the static degeneracy using an implementation from the NetworkX python library [8]. This approach is rather inefficient. For example, it took about seven hours to compute the Δ-slice degeneracy for karlsruhe with $\Delta \sim 5^5$ (equating to a Δ value of about two hours).

5.2 Results and Running Times

We now study the efficiency of BronKerboschDeltaPivot*, evaluate pivoting strategies, and compare the result to Algorithm VLM.

Pivoting. Generally we observed that pivoting plays a negligible role when Δ is small compared to the overall lifetime of the graph, that is, when Δ is less than roughly one third of the lifetime. In this case, pivoting has almost no effect on the running time and the number of recursive calls. For larger values of Δ, however, pivoting can make a clear difference depending on the type of temporal graph.

We tested five strategies for selecting a set of pivots from P in BronKerboschDeltaPivot*. Call a set of pivots is *maximal* if the interval of each element from P overlaps with at least one pivot. We tested the following variants of pivot sets:

1A) a single arbitrary pivot,

1G) a single pivot maximizing the number of elements removed from P,

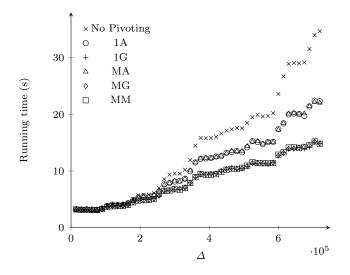MA) an arbitrary maximal set of pivots (pivots picked one-by-one arbitrarily),

Figure 3: Running time for different pivoting strategies on highschool-2012.

MG) a maximal set of pivots (pivots picked one by one according to the maximum number of further elements removed from P), and

MM) a set of pivots which maximizes the number of elements removed from P.

Clearly, each strategy has its own trade-off between the time needed to compute the pivots and the possible reduction in recursive calls.

Running times are given for highschool-2012 in Figure 3 with Δ between 15,000 and 725,000. We note that running times for some very small values of Δ below 15,000 are larger than 30 s and hence do not fit in the chart. We consider this phenomenon more closely below. For $\Delta \leq 15,000$ there is no appreciable difference between the pivoting strategies. In terms of relative difference between pivoting strategies, highschool-2012 seems to be a representative example. Strategies 1G and MG seem to be the best options: they do not incur much overhead compared to no pivoting for small Δ and yield strong running time improvements for larger Δ. In comparison to no pivoting, strategies 1G and MG achieve a 60 % reduction in recursive calls for Δ-values of around $7 \cdot 10^6$ in highschool-2012. Since the running times of strategy 1G and MG are so close to each other we conclude that in most cases there is only one important pivot that should be selected. We were surprised to see that maximizing the overall number of elements removed from P via the pivot set (strategy MM) results in slightly worse running times and slightly larger numbers of recursive calls. The number of elements that are removed by a pivot in one recursive call of the algorithm ranges between one and 14 while many of the calls remove two to four elements. Notice that occasional reduction by ten or more elements can
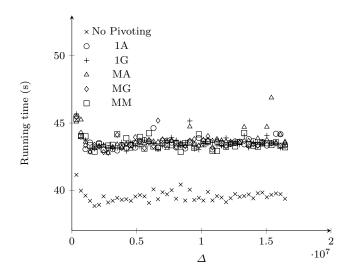
Figure 4: Running time for different pivoting strategies on facebook-like.

substantially decrease the search space, because in general its size depends exponentially on the size of P.

Figure 4 shows running times for facebook-like. On this graph, pivoting seldom removes more than one element from the candidate set P in one call of the recursive procedure. Hence, for this instance, pivoting mainly incurs overhead for computing the pivots, but do not substantially decrease the search space. We consequently observe about $10\,\%$ slower running times, regardless of the pivoting strategy.

In conclusion, strategy 1G offers the best trade-off between additional running time spent with computing the pivot(s) and running time saved due to decreased number of recursive calls. Overall, the the possible benefits seem to outweigh the overhead incurred by pivoting on some instances. All remaining experiments were thus carried out with strategy 1G.

Running Times and Comparison with Algorithm VLM. We experimented with BRONKERBOSCHDELTAPIVOT* (Algorithm 5) using pivoting strategy 1G and with Algorithm VLM for $\Delta = 0$ and $\Delta \sim 5^i$ with $i = 1, 3, 5, 7, 9$ (where the lifetime allowed such values of Δ). An excerpt of the results is given in Table 3. Clearly, larger instances with more vertices or edges demand a longer running time. However, even large instances like infectious can still be solved within one hour.

From our theoretical results in Section 3 we expected that the running time of BRONKERBOSCHDELTAPIVOT* increases exponentially with growing Δ-slice degeneracy. As the Δ-slice degeneracy grows very slowly with increasing Δ (see Table 2), we expected a corresponding moderate growth in running time with respect to Δ. For larger Δ, this is consistent with the ex-

Table 3: Δ-clique statistics and running times: $|\mathcal{C}|$ denotes the number of maximal Δ-cliques, s denotes the maximum Δ-clique size, ℓ the maximum Δ-clique lifetime divided by 10^5, t_{BKD} and t_{VLM} denote the running time in seconds of BRONKERBOSCHDELTA* and Algorithm VLM, respectively. Empty cells represent an exceeded running time limit of one hour.

	$\Delta = 0$				
Instance	$|\mathcal{C}|$	s	ℓ	t_{BKD}	t_{VLM}
facebook-like	61,648	2	1,674	169	12
highschool-2011	26,510	5	27	131	7
highschool-2012	42,285	5	73	248	12
highschool-2013	172,362	5	36	1,952	118
hospital-ward	27,910	5	35	370	14
hypertext	19,150	6	21	85	5
infectious	349,787	5	695	1,530	2,515
karlsruhe					1,494
primaryschool	107,121	5	12	995	147

	$\Delta \sim 5^3$				
Instance	$|\mathcal{C}|$	s	ℓ	t_{BKD}	t_{VLM}
facebook-like	33,876	4	1,675	70	1,141
highschool-2011	7,394	7	27	5	153
highschool-2012	9,501	6	73	8	236
highschool-2013	57,121	6	36	178	1,990
hospital-ward	8,694	7	35	15	226
hypertext	6,345	7	21	8	107
infectious	134,787	9	695	1,195	
karlsruhe					
primaryschool	83,314	9	12	83	

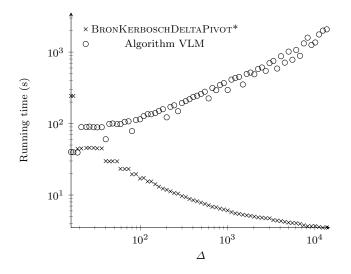|	$\Delta \sim 5^5$				
Instance	$|\mathcal{C}|$	s	ℓ	t_{BKD}	t_{VLM}
facebook-like	23,247	5	1,709	47	
highschool-2011	7,760	10	28	4	
highschool-2012	7,536	7	75	4	1,365
highschool-2013	29,752	8	37	23	
hospital-ward	10,869	12	36	7	
hypertext	7,459	7	23	4	2,193
infectious	163,162	16	697	1,277	
karlsruhe	235,684	9	12,417	1,235	
primaryschool	508,430	20	15	890	

Figure 5: Running time vs. Δ on highschool-2012.

perimental results, as shown in Figures 3, 4 and Table 3. However, for (very) small Δ we often observe an initial spike in the running time (and number of Δ-cliques) which then subsides. This is also shown in Figure 5. A possible explanation for this spike is that, for small Δ, the Δ-neighborhood of many vertices becomes very fragmented, leading to large candidate sets P in the algorithm (although the size of P is still linear in the input size for constant Δ-slice degeneracy). Furthermore, if Δ is small, then many singleton edges may form maximal Δ-cliques themselves. These Δ-cliques then get taken up into larger maximal Δ-cliques when Δ increases, which decreases the number of Δ-cliques and running times for BRONKERBOSCHDELTAPIVOT*.

On facebook-like our algorithm notably is comparably efficient given the relatively large size (see Figures 4 and 6). Furthermore, the number of Δ-cliques does not seem to vary strongly with changing values of Δ. These two facts may hint at some special structure that is present in temporal graphs based on online social networks, in addition to small Δ-slice degeneracy.

Algorithm VLM is usually faster than BRONKERBOSCHDELTAPIVOT* for small values of Δ below the $\Delta \sim 5^3$ threshold. Starting from there, however, BRONKERBOSCHDELTAPIVOT* outperforms Algorithm VLM with running times smaller by at least one order of magnitude and up to three orders of magnitude (see Table 3). In terms of main memory, $385\,\mathrm{MB}$ is the maximum used by BRONKERBOSCHDELTAPIVOT* over all solved instances, attained on infectious for $\Delta = 0$. On this instance, Algorithm VLM uses $494\,\mathrm{MB}$ and often more than $1\,\mathrm{GB}$.

Finally we mention that, when increasing the time limit to six hours, BRONKERBOSCHDELTAPIVOT* can solve all instances of karlsruhe for $\Delta = 0$
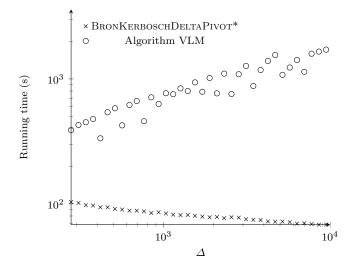
Figure 6: Running time vs. Δ on facebook-like.

and $\Delta \sim 5^i$ for $i = 1, 3, 5, 7, 9$ wherein the last value of Δ involves enumerating $43 \cdot 10^6$ maximal Δ-cliques.

6 Conclusion and Outlook

We studied the algorithmic complexity of enumerating Δ-cliques in temporal graphs. We adapted the Bron-Kerbosch algorithm ([3]), including the procedure of pivoting to reduce the number of recursion calls, to the temporal setting and provided a theoretical analysis. For the theoretical analysis, we formalized and employed the concept of Δ-slice degeneracy which may be a useful parameter when analyzing problems in sparse temporal graphs.

In experiments on real-world data sets, we showed that our algorithm is notably faster than the first approach for enumerating all maximal Δ-cliques in temporal graphs due to Viard et al. [28, 29]. Our experimental results further reveal that pivoting can notably decrease the running time for large values of Δ. Furthermore, we measured the Δ-slice degeneracy for different Δ-values and showed that it is reasonably small in many real-world data sets.

As to future research, an algorithmic challenge is to find a more efficient way to compute the Δ-slice degeneracy of a given temporal graph, perhaps via different characterizations as in the case of static graphs. See [4] for an account of several equivalent definitions of the degeneracy of a static graph. Regarding the adapted version of the Bron-Kerbosch algorithm, our theoretical analysis (based on the Δ-slice degeneracy parameter) of the running time still leaves room for improvement. In particular, we leave the impact of pivoting on the running time upper bound as an open question for future

research. It furthermore makes sense to try and implement further improved branching rules on top of pivoting. This was also successful for the static Bron-Kerbosch algorithm [21]. Another interesting question is whether an analogue to the degeneracy ordering can be defined in the temporal setting and, if so, whether it can be used to further improve the algorithm.

We are grateful to two anonymous SNAM reviewers whose feedback helped to significantly improve the presentation and to eliminate some bugs and inconsistencies.

References

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

[2] S. Boccaletti, G. Bianconi, R. Criado, C. I. Del Genio, J. Gómez-Gardeñes, M. Romance, I. Sendiña-Nadal, Z. Wang, and M. Zanin. The structure and dynamics of multilayer networks. *Physics Reports*, 544(1):1–122, 2014.

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

[4] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in large sparse real-world graphs in near-optimal time. *ACM Journal of Experimental Algorithmics*, 18(3):3.1:1–3.1:21, 2013.

[5] T. Erlebach, M. Hoffmann, and F. Kammer. On temporal graph exploration. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015)*, volume 9134 of *LNCS*, pages 444–455. Springer, 2015.

[6] V. Gemmetto, A. Barrat, and C. Cattuto. Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC Infectious Diseases*, 14(1):1, 2014.

[7] R. Goerke. Email network of KIT informatics. `http://i11www.iti.uni-karlsruhe.de/en/projects/spp1307/emaildata`, 2011.

[8] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, pages 11–15, 2008.

[9] A.-S. Himmel. Enumerating maximal cliques in temporal graphs. Bachelorthesis, TU Berlin, January 2016. URL `http://fpt.akt.tu-berlin.de/publications/theses/BA-anne-sophie-himmel.pdf`. Bachelor thesis.

[10] A.-S. Himmel, H. Molter, R. Niedermeier, and M. Sorge. Enumerating maximal cliques in temporal graphs. In *Proceedings of the 2016 IEEE/ACM International Conference On Advances in Social Networks Analysis and Mining (ASONAM 2016)*, pages 337–344. IEEE, 2016.

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

[12] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Isolation concepts for clique enumeration: Comparison and computational experiments. *Theoretical Computer Science*, 410(52):5384–5397, 2009.

[13] L. Isella, J. Stehlé, A. Barrat, C. Cattuto, J.-F. Pinton, and W. Van den Broeck. What's in a crowd? Analysis of face-to-face behavioral networks. *Journal of Theoretical Biology*, 271(1):166–180, 2011.

[14] H. Ito and K. Iwama. Enumeration of isolated cliques and pseudocliques. *ACM Transactions on Algorithms*, 5(4):40, 2009.

[15] J. Kleinberg and É. Tardos. *Algorithm Design*. Pearson Education, 2006.

[16] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38):3640–3654, 2009.

[17] M. Lahiri and T. Y. Berger-Wolf. Periodic subgraph mining in dynamic networks. *Knowledge and Information Systems*, 24(3):467–497, 2010.

[18] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 177–187. ACM, 2005.

[19] O. Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

[20] O. Michail and P. G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016.

[21] K. A. Naudé. Refined pivot selection for maximal clique enumeration in graphs. *Theoretical Computer Science*, 613:28–37, 2016.

[22] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora. Graph metrics for temporal networks. In P. Holme and J. Saramäki, editors, *Temporal Networks*, pages 15–40. Springer Berlin Heidelberg, 2013.

[23] T. Opsahl and P. Panzarasa. Clustering in weighted networks. *Social Networks*, 31(2):155–163, 2009.

[24] J. Stehlé, N. Voirin, A. Barrat, C. Cattuto, L. Isella, J.-F. Pinton, M. Quaggiotto, W. Van den Broeck, C. Régis, B. Lina, et al. High-resolution measurements of face-to-face contact patterns in a primary school. *PLoS ONE*, 6(8):e23176, 2011.

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

[26] T. Uno and Y. Uno. Mining preserving structures in a graph sequence. *Theoretical Computer Science*, 654:155–163, 2016.

[27] P. Vanhems, A. Barrat, C. Cattuto, J.-F. Pinton, N. Khanafer, C. Régis, B.-a. Kim, B. Comte, and N. Voirin. Estimating potential infection transmission routes in hospital wards using wearable proximity sensors. *PLoS ONE*, 8(9):e73970, 2013.

[28] 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*, pages 1517–1522. ACM, 2015.

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