Algorithms for analyzing evolving networks on the Dark Web & in science

Boekhout, H.D.

Citation
Boekhout, H. D. (2026, March 17). *Algorithms for analyzing evolving networks on the Dark Web & in science*. Retrieved from https://hdl.handle.net/1887/4297227

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4297227

Note: To cite this publication please use the final published version (if applicable).

Part II

Temporal weighted maximal clique enumeration

4

Fast maximal clique enumeration in weighted temporal networks

Cliques, groups of fully connected nodes in a network, are often used to study group-dynamics of complex systems. In real-world settings, group dynamics often have a temporal component. For example, conference attendees moving from one group conversation to another. Recently, maximal clique enumeration methods have been introduced that add temporal (and frequency) constraints, to account for such phenomena. These methods enumerate so-called (δ, γ)-maximal cliques.

In this chapter, we introduce an efficient (δ, γ)-maximal clique enumeration algorithm, that extends γ from a frequency constraint to a more versatile weighting constraint, while maintaining δ as the temporal constraint. Additionally, we introduce a definition of (δ, γ)-cliques, that resolves a problem of existing definitions in the temporal domain. Our approach, which was inspired by a state-of-the-art two-phase approach, introduces a more efficient initial (stretching) phase. Specifically, we reduce the time complexity of this phase to be linear with respect to the number of temporal edges. Furthermore, we introduce a new approach to the second (bulking) phase, which allows us to efficiently prune search tree branches. Consequently, in experiments we observe significant speed-ups, at times by several order of magnitude, on various (large) real-world datasets. Our algorithm vastly outperforms the existing state-of-the-art methods for temporal networks, while also extending applicability to weighted networks.

This chapter is based on:

- H. D. Boekhout and F. W. Takes. Fast maximal clique enumeration in weighted temporal networks. *Social Network Analysis and Mining*, 16(1):10, 2026

4.1 Introduction

To study real-world systems, the field of network science models these systems as networks of connected entities. For example, entities, also called nodes, can represent the attendees at conferences that are connected through their proximity [76], online social media users connected through their interactions [115, 142], or online communities connected through hyperlinks [85]. Finding groups of entities that are fully connected to one another, called *cliques*, in these networks, can provide valuable insight into (the group-dynamics of) the complex systems they model. Consequently, the enumeration of cliques has long been the subject of study in network science and graph theory [92].

Many real-world systems are more complex than can be modeled by a simple (undirected) graph. Connections may be short-lived, occurring only at specific times, such as the proximity of conference attendees. Different connections may also hold differing significance. For example, the hyperlinks between online communities may be accompanied by positive or negative sentiment. We can model such complexities through timestamps and weights on the connections to form so-called *(weighted) temporal networks*. Consequently, we can also add temporal and weighting constraints on the clique enumeration. For example, we may require cliques to be fully connected with a given minimum weight γ for every δ-time interval of their time span, e.g., teams of scientists where every pair of authors in the team co-author at least γ publications for every δ-year period of the team's lifespan. In the example of the social network modeling people in proximity of each other, we would be looking at people that interacted with each other a particular number of times γ during each length of time δ in a particular time span. Specifically, this chapter focuses on the enumeration of (δ, γ)-*maximal cliques*, which are (δ, γ)-cliques for which neither the node set nor the time span of the clique can be extended further.

In the literature, several methods have been proposed to enumerate temporal cliques (i.e., δ-cliques). In a recent publication, Banerjee & Pal [8] introduced a two-phase approach to enumerate (δ, γ)-maximal cliques, where γ refers to the frequency of links instead. Of course, for unweighted networks, when all edges have a weight of one, the frequency and cumulative weight are equal. While it is in theory possible to simulate larger, positive, integer based weights by including multiple edges with the same timestamp, this would come at the cost of increasing the network size and therefore increasing the computational costs of clique enumeration. Furthermore, this approach would not work when a network includes negative and decimal weights. Here, we define (δ, γ)-cliques to cover weighted networks, including those with negative and decimal weighting, while fixing a practical problem of existing definitions in the temporal domain. Additionally, we propose a different usage of δ in the definition of cliques to more naturally align with common-sense interpretation, to prevent

incorrect usage by end-users. Furthermore, we introduce a new (δ, γ)-maximal clique enumeration algorithm that was inspired by Banerjee & Pal's two-phase approach, but uses new approaches for both phases. Importantly, our algorithm improves the time complexity of the first phase (i.e., the stretching phase), such that it is independent of γ, and introduces pruning of search tree branches to speed up the second phase (i.e., the bulking phase). Consequently, our algorithm is able to process weighted networks at no added computational cost compared to their unweighted counterparts.

Real-world weighted temporal scenarios can range from identifying groups that have on balance exclusively positive (or negative) relationships with one another, to identifying groups that are sufficiently strongly connected. The former scenario might, for example, involve identifying groups of users on social media that express primarily positive sentiment in their interactions, where weights are determined through sentiment analysis of social media posts. The latter scenario could involve identifying scientific teams with strong collaborative ties from weighted temporal co-authorship networks. Here, the strength of collaborative ties may be determined by weighting edges by the number of co-authored publications in a year. Alternatively, we may weigh each of the co-authored publications by, for example, the size of the author team, presuming that fewer authors lead to stronger individual collaborative ties. In short, the inclusion of weights allows for the encoding of important contextual information into a network, potentially leading to the identification of cliques representing groups with new characteristics that could otherwise not be investigated.

Through experiments on a set of both small and larger real-world network datasets with up to 4.4 million nodes and 19 million edges, we show that our proposed algorithm outperforms all existing temporal clique enumeration algorithms, at times by several orders of magnitude, demonstrating better scalability. In almost all cases, we outperform the current state-of-the-art approach by Banerjee & Pal in both time and space complexity, and consequently, running time and memory usage.

The remainder of this paper is structured as follows. In Section 4.2 we discuss related work on clique enumeration. Then, in Section 4.3 we provide notation and definitions used throughout the paper, including our proposed definition of (δ, γ)-cliques. Next, we introduce our new faster (δ, γ)-clique enumeration algorithm in Section 4.4. In Section 4.5, we describe the datasets used in our experiments. Subsequently, the experiments and results are discussed in Section 4.6. Finally, we draw our conclusions in Section 4.7.

4.2 Related work

Cliques and their enumeration were already subject of research over 50 years ago. In 1973, Bron & Kerbosch [35] presented two backtracking algorithms for clique enumeration in simple static undirected networks, that use a branch-and-bound technique to prune search tree branches. These algorithms still form the basis of some state-of-the-art approaches. Tomita et al. [133] vastly improved the pruning of Bron & Kerbosch's backtracking algorithms through smart pivot selection. When expanding a clique, they prioritize vertices that have the largest neighborhood overlap with the set of candidate vertices. Consequently, Eppstein et al. [63] employed a degeneracy ordering in the outer level of recursion of Tomita's adaptation, to achieve a time complexity that is nearly worst-case optimal. These adaptations of the Bron & Kerbosch algorithm are still used in many network-science software packages today. The pruning techniques we apply in this paper for temporal cliques, were in part inspired by these methods.

Various other methods and problem extensions were introduced over the years. For example, some approaches explored the use of partitioning to deal with limited memory [44], others attempted to improve efficiency through parallelisation [126, 154, 73, 55], and yet others tried using iterative enumeration [84] or decomposition [102]. Furthermore, there were also works that, instead of maximal cliques, specifically considered maximum cliques [61] or the slightly different isolated, pseudo, and/or defective cliques [77, 134, 54] or even k-plexes [48]. Finally, some research focused on different network types, ranging from spatial networks [157], to uncertain networks [106, 148, 99], to signed networks [43, 147], to bi-partite networks [148, 147], to temporal networks [5, 6, 7, 8, 139, 140, 70, 141, 158, 120]. In this chapter, we focus specifically on the latter type of network, the temporal networks and specifically (δ, γ)-maximal cliques.

The δ-maximal cliques, which enforce temporal constraints upon the edges of a clique, were first introduced by Viard et al. [139, 140]. The methods introduced by Viard et al. to enumerate these cliques were greedy in nature and scaled poorly. Himmel et al. [70] then adapted the Bron & Kerbosch algorithm to account for the temporal dimension to enumerate δ-maximal cliques, which significantly improved on Viard's methods. Next, Banerjee & Pal [5] extended the δ-maximal clique definition with a minimum edge frequency γ requirement. Recently, the same authors introduced a two-phase approach to more efficiently enumerate such (δ, γ)-maximal cliques, which also improved upon Himmel et al.'s performance for δ-maximal cliques [6, 8]. In their approach, the second phase relies on iterative enumeration which prunes only duplicate branches. Here, we build upon the work of Banerjee & Pal [8] to accommodate also weighted temporal networks and to improve the algorithm's efficiency through further pruning methods.

4.3 Preliminaries

In this section we first introduce notation and definitions dealing with (weighted) temporal networks in Section 4.3.1. Next, in Section 4.3.2 we discuss existing maximal clique definitions (both static and temporal) and identify two practical problems existing definitions cause in the temporal domain when applied in a real-world setting. Additionally, we redefine maximal cliques in the temporal domain so that these issues no longer occur and such that it can be applied to weighted temporal networks.

4.3.1 Basic notation and definitions

A *temporal network* $\mathcal{G} = (V, E, \mathcal{T})$ is comprised of its static elements V and E, describing the node set and the edges connecting those nodes, respectively, and a mapping $\mathcal{T}$ of the edges to their temporal occurrence time stamps, i.e., $\mathcal{T}\colon E \to 2^T \setminus \emptyset$ with $T = \{t_{min}, \ldots, t_{max}\}$ as the set of all discrete time stamps. Consequently, the set of temporal edge occurrences $E^{\mathcal{T}}$ consists of all *temporal edge instances* (t, u, v) in $\mathcal{G}$, where $u, v \in V$, $(u, v) \in E$, and $t \in T$. We say that $m = |E^{\mathcal{T}}|$. Note that (u, v) does not imply a direction here, as we only consider undirected networks in this chapter.

Given the above, we can define a *weighted temporal network* $\mathcal{G} = (V, E, \mathcal{T}, \mathcal{W}^{\mathcal{T}})$, as a temporal network with the added mapping $\mathcal{W}^{\mathcal{T}}\colon E^{\mathcal{T}} \to \mathbb{R}$, which maps each temporal edge occurrence to a weight. The set of weighted temporal edge occurrences $E^{\mathcal{T}\mathcal{W}^{\mathcal{T}}}$, thus consists of all *weighted temporal edge instances* (t, u, v, w) in $\mathcal{G}$, where $u, v \in V$, $(u, v) \in E$, $t \in T$, and $w \in \mathbb{R}$. Note that a temporal network is equivalent to a weighted temporal network with all weights set to one.

Given an edge $(u, v) \in E$ and time interval $[t_b, t_e]$, the *frequency* $f((u, v), [t_b, t_e])$ represents the cumulative weight of all weighted temporal instances of edge (u, v) in the interval $[t_b, t_e]$. Thus, for interval $[t_{min}, t_{max}]$, i.e., for the entire *lifetime of the network*, the frequency of an edge in an unweighted temporal network equals its number of temporal occurrences.

We visually represent (weighted) temporal networks using the link stream model which shows the (weighted) relationship between nodes over time (see Figure 4.1a).

4.3.2 Maximal cliques

In a static network, a set of nodes $X \subseteq V$ is a *clique* when they are fully connected, i.e., for all $u, v \in X$ it holds that $(u, v) \in E$. We say that a clique is *maximal*, when there exists no set of nodes $Y \subseteq V$ such that $X \subseteq Y$, $X \neq Y$, and Y is a clique. This notion of (maximal) cliques was first extended by Viard et al. [139, 140] to the temporal domain as follows.

Definition 1 (δ-clique). *Given a time duration $\delta \in \mathbb{Z}^{>0}$, a δ-clique of the temporal network $\mathcal{G}$ is a vertex set and time interval pair $C = (X, [t_b, t_e])$ with $X \subseteq V_{\mathcal{G}}$ and $t_b, t_e \in T$ such that for all $u, v \in X$, where $u \neq v$, and $\tau \in [t_b, t_e - \delta + 1]$, it holds that $f((u, v), [\tau, \tau + \delta - 1]) \geq 1$.*

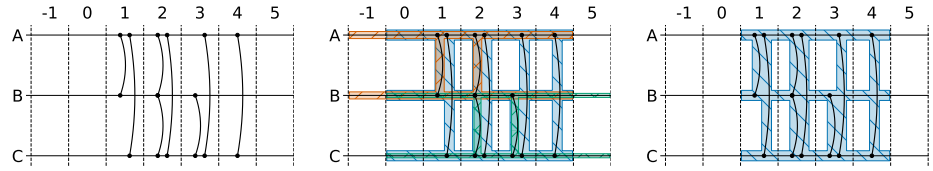
Definition 2 (δ-maximal clique). *A δ-clique $C = (X, [t_b, t_e])$ is considered δ-maximal, when there exists no δ-clique $C' = (X', [t'_b, t'_e])$ such that either: $X \subseteq X'$, $X \neq X'$, $t'_b \leq t_b$, and $t'_e \geq t_e$ holds; or $X = X'$ and ($t'_b < t_b$ or $t'_e > t_e$) holds.*

In other words, the vertex set X of a δ-clique is fully connected during every δ-time interval in the interval $[t_b, t_e]$; and a δ-maximal clique can neither extend its time interval for the current node set nor extend the node set without shrinking the interval. In line with Banerjee & Pal [8], we refer to cliques that cannot extend their time interval, regardless of whether the node set is maximal, as *duration-wise maximal*.

Note that, compared to previous related work, we changed the use of δ in the definition of a δ-clique, to more closely align with its (natural) interpretation. The intended interpretation of δ, as also stated textually by previous works, is that "every node pair must be connected at least after every δ timestamps in the time interval". Consider a periodic network with measurements, i.e., edges, occurring every 20 seconds. Given such a network, the periodicity of found cliques has three phases: (1) *pre-periodicity*, where δ is too small to connect subsequent events at timestamps 20 and 40 within one clique; (2) *exact periodicity*, where for every edge in the clique both timestamps 20 and 40 are required to establish a clique over time interval [20, 40]; and (3) *overlap periodicity*, where some edges having an occurrence at timestamp 20 and others at 40 is sufficient for a clique with time interval [20, 40]. Given the interpretation of δ above, the natural assumption would be that, $\delta \leq 19$ corresponds to pre-periodicity, $\delta = 20$ to exact periodicity, and $\delta \geq 21$ corresponds to overlap periodicity. Although this holds for Definition 1, this did not hold for previous works where $\delta = 20$ corresponded to overlap periodicity and $\delta = 19$ to exact periodicity [70, 5, 6, 7, 8].

4.3.2.1 Solving a practical problem in the temporal domain

Although theoretically sound, the δ-maximal clique and Banerjee & Pal's (δ, γ)-maximal clique definitions create a practical problem where some maximal cliques are in practice a strict subgraph of other maximal cliques. This problem originates from the fact that the definitions allow the time interval to be expanded beyond the scope of the temporal edge instances themselves. For example, consider a network consisting of a single temporal edge instance $(3, u, v)$. With δ set to 3, the δ-clique $C = (\{u, v\}, [1, 5])$ is a valid and δ-maximal clique, despite including a length 2 time interval before and after



(a) Link stream representation of temporal network

(b) (δ, γ)-maximal cliques (for existing definition)

(c) (δ, γ)-maximal cliques (for our definition)

Figure 4.1: Given $\delta = 4$ and $\gamma = 2$ and temporal network (a) shown as a link stream model, then (b) highlights the three maximal (δ, γ)-cliques $C_1 = (\{A, B\}, [-1, 4])$ in orange (with backslash pattern), $C_2 = (\{B, C\}, [0, 5])$ in green (with cross pattern), and $C_3 = (\{A, B, C\}, [0, 4])$ in blue (with forward slash pattern), that are by the current definitions (δ, γ)-maximal. Note that the edges included in C_1 and C_2 are also all included in C_3. Therefore, C_1 and C_2 are effectively subgraphs of C_3. Finally, (c) shows the only remaining (δ, γ)-maximal clique, when the time interval is bounded by the first and last temporal edge occurrences included. This clique ($C_4 = (\{A, B, C\}, [1, 4])$) is found by the proposed Definition 3. Notice, that C_4 is the bounded form of C_3 from (b) and that the bounded forms of cliques C_1 and C_2 are no longer maximal.

the only actual edge instance. By allowing these empty intervals, the interpretation of a clique becomes more complex as one cannot assume that the first and last interactions of the clique members actually occur at the borders of the time interval. Moreover, the link stream model shown in Figure 4.1b, demonstrates how empty intervals lead to the aforementioned practical problem of some maximal cliques being effectively subgraphs of others.

We combat this problem by adding bounds on the start and end of the time interval, determined by the actual temporal edge instances, to the (δ, γ)-clique definitions of Banerjee & Pal. We additionally define γ such that it describes a threshold on the cumulative weight of the temporal edge instances, instead of a threshold on the number of temporal instances. Given our definition of frequency in Section 4.3.1, the following definition now holds both for weighted and unweighted networks.

Definition 3 ((δ, γ)-clique)**.** *Given a time duration $\delta \in \mathbb{Z}^{>0}$ and a cumulative weight threshold $\gamma \in \mathbb{R}$, a (δ, γ)-clique of the temporal network $\mathcal{G}$ is a vertex set and time interval pair $C = (X, [t_b, t_e])$ with $X \subseteq V_\mathcal{G}$ and $t_b, t_e \in T$ such that for all $u, v \in X$, where $u \neq v$, and $\tau \in [t_b, t_e - \delta + 1]$, it holds that $f((u, v), [\tau, \tau + \delta - 1]) \geq \gamma$ and there exist temporal edge instances (t_b, u, v) and (t_e, u', v') with $u, v, u', v' \in X$.*

Note that this also redefines δ-cliques for a temporal network given $\gamma = 1$ and that

the extension to δ-maximal cliques follows Definition 2. The extension for (δ, γ)-cliques to maximal cliques is subsequently similar to that of δ-cliques.

Definition 4 ((δ, γ)-maximal clique)**.** *A (δ, γ)-clique $C = (X, [t_b, t_e])$ is considered* (δ, γ)-maximal*, when there exists no (δ, γ)-clique $C' = (X', [t'_b, t'_e])$ such that either: $X \subseteq X'$, $X \neq X'$, $t'_b \leq t_b$, and $t'_e \geq t_e$ holds; or $X = X'$ and ($t'_b < t_b$ or $t'_e > t_e$) holds.*

4.4 Fast (δ, γ)-maximal clique enumeration

In this section we propose a new (δ, γ)-maximal clique enumeration algorithm. The overall methodology, described in Section 4.4.1, is inspired by the state-of-the-art two-phase approach introduced by Banerjee & Pal [6, 8]. In Section 4.4.2, we introduce our approach to the initial so-called *stretching phase*, which determines all 2-node duration-wise maximal (δ, γ)-cliques. Our contribution with respect to previous work lies in the fact that it accommodates weighted networks while also reducing the time complexity of this step from $\mathcal{O}(\gamma m)$ to $\mathcal{O}(m)$. Then in Section 4.4.3, we propose a new approach for the so-called *bulking phase* which grows the node sets of the duration-wise maximal cliques. This approach utilises already computed values to efficiently compute node set extensions and prune branches of the search tree, resulting in substantial speedups, as we will demonstrate in Section 4.6.

4.4.1 Overall approach

The state-of-the-art (δ, γ)-maximal clique enumeration algorithm introduced by Banerjee & Pal [6, 8], consists of two phases: (1) the stretching phase and (2) the shrink and bulk phase. The stretching phase takes, for each edge, the set of temporal instances as input and produces the set C^s consisting of all 2-node duration-wise maximal (δ, γ)-cliques. In other words, it finds the maximally stretched time intervals of each edge that satisfy the δ- and γ-constraints of a (δ, γ)-clique. Subsequently, the shrink and bulk phase continually expands the node sets (of the cliques in C^s) one node at a time, by combining the cliques in C^s. As the process expands the node sets, it also shrinks the time interval when required to continue to conform to the δ- and γ-constraints. Through this node expansion process, the set R of all (δ, γ)-maximal cliques is produced.

Our new proposed enumeration algorithm follows the same overall approach of first stretching the time intervals of individual edges before expanding the node sets. However, our corrected definition of a (δ, γ)-clique and the expansion to weighted networks, require us to propose entirely new (and faster) approaches for both phases.

For example, note that during the bulk phase, (δ, γ)-cliques no longer strictly shrink their time interval during node expansions. After all, the time interval of a clique is now constrained by the actual temporal instances, whereas for Banerjee & Pal [6, 8] the time interval extended as far as the δ- and γ-constraints remained satisfied, i.e., it extended to the *maximum temporal growth* of the clique. Thus, if temporal instances introduced by node expansions are within the constraints of the maximum temporal growth, the time interval can grow due to node expansion during our new bulk phase. Therefore, for our proposed method, we refer to the second phase as the *bulking phase*.

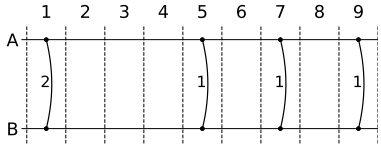
We describe our proposed approaches for the stretching and bulking phases in detail in Sections 4.4.2 and 4.4.3, respectively. The pseudo code describing our proposed methods are presented, respectively, in Algorithms 1, 2 and Algorithms 3–7 in Section 4.8.1. Lines throughout all algorithms are numbered sequentially. Hereafter, we refrain from referring to specific algorithms and only specify the relevant lines.

4.4.2 Stretching phase

The stretching phase initializes the (δ, γ)-maximal clique enumeration process by first exploring the potential temporal growth for every edge. In other words, the stretching phase enumerates all 2-node duration-wise maximal (δ, γ)-cliques. It does so by evaluating the temporal instances of each edge separately (line 3), while skipping any edges for which the cumulative weight of the instances is lower than γ (line 4). The algorithm proposed by Banerjee & Pal [6, 8] for this phase is highly reliant on γ indicating the number of occurrences of temporal edge instances. Since we cannot rely on this for weighted networks, we instead propose a new (repeated) three step process (see lines 10–62): (1) initialization; (2) discovery of the start of a valid (δ, γ)-clique; and (3) finding the end of said clique such that it is duration-wise maximal. During this process, we continually increase two indices (bi and ei) on the temporal instances of the edge, so that the indicated instances are never more than δ timestamps apart. By keeping track of the current cumulative weight of the range of instances specified by those indices (*current_weight*), we can find the set of all 2-node duration-wise maximal (δ, γ)-cliques of the given edge. We demonstrate the execution of the process through two toy examples in Figure 4.2.

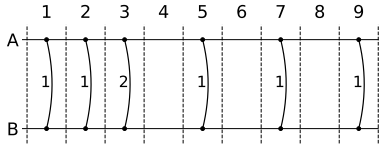
Below we describe the steps of the stretching process and how they ensure that exclusively the set of all duration-wise maximal (δ, γ)-cliques is added to result set C^s.

1. First, the indices bi and ei, i.e., the begin and end index, respectively, are initialized to point to the first temporal edge instance in the temporal ordering. Furthermore, the *current_weight* is initialized to the weight of this first instance (lines 11, 12).



bi	t_{bi}	ei	t_{ei}	cw	C^s	step(s)
0	1	0	1	2	$\{\}$	1
0	1	1	5	3	$\{\}$	2
0	1	1	5	3	$\{\}$	3a, 3b
1	5	1	5	1	$\{[1,5]\}$	3d(ii)
1	5	2	7	2	$\{[1,5]\}$	2
1	5	3	9	3	$\{[1,5]\}$	2
1	5	3	9	3	$\{[1,5]\}$	3a, 3b
x	x	x	x	x	$\{[1,5],$ $[5,9]\}$	3c

(a) $\delta = 5, \gamma = 3$



bi	t_{bi}	ei	t_{ei}	cw	C^s	step(s)
0	1	0	1	1	$\{\}$	1
0	1	1	2	2	$\{\}$	2
0	1	2	3	4	$\{\}$	2
0	1	3	5	5	$\{\}$	3a
1	2	3	5	4	$\{\}$	3b
1	2	3	5	4	$\{\}$	3a, 3b
2	3	4	7	4	$\{\}$	3d(iii)
2	3	4	7	4	$\{\}$	3a, 3b
3	5	4	7	2	$\{[1,7]\}$	3d(ii)
3	5	5	9	3	$\{[1,7]\}$	2
x	x	x	x	x	$\{[1,7]\}$	2

(b) $\delta = 5, \gamma = 4$

Figure 4.2: The stretching process for two toy example link stream models. Edge weights are depicted alongside each edge, cw indicates the *current_weight*, and "step" indicates which step in the process produced the values presented in that row. Note that, for the cliques added to C^s, we only show their time interval in the tables.

Note that during the remainder of the process, whenever bi and ei are increased, the *current_weight* is respectively decreased and increased accordingly.

2. Second (lines 14–24), ei is increased until the *current_weight* is greater than or equal to γ. If at any point in this process the δ-constraint (i.e., $t_{ei} - t_{bi} < \delta$) is no longer satisfied, bi is increased until the constraint is satisfied once more (lines 18–21). Additionally, if ei ever exceeds the number of temporal edge instances, we know that no more valid (δ, γ)-cliques can be found and the process is stopped (line 16).

   Once the *current_weight* is greater than or equal to γ, we know that there exists a valid (δ, γ)-clique (according to Definition 3) with t_b set to the current t_{bi}. Additionally, t_{ei} now indicates the first timestamp from t_{bi} where the γ weight is reached. Because knowledge of this timestamp plays a vital role for determining temporal overlap between cliques in the bulking phase, it is stored as a clique property ($tbMax$, which is set to the current value of t_{ei}).

3. Finally, in order to find the duration-wise maximal clique whose time interval starts at t_b, the indices are systematically increased to ensure that the (δ, γ)-clique with time interval $[t_b, t_{ei}]$ always remains valid (lines 25–60). This is accomplished by ensuring that the cumulative weight of the instances in time interval $[t_{bi}, t_{ei}]$ always remains at least γ and that δ-constraint (i.e., $t_{ei} - t_{bi} < \delta$) always holds. To this end, the following steps are (repeatedly) performed:

   (a) First, ei is increased as long as doing so does not compromise the δ-constraint (lines 28–31). Since at this point a minimum γ weight is guaranteed for each δ-time interval up to $[t_{bi}, t_{bi} + \delta - 1]$, ei can be safely increased as long as t_{ei} does not exceed $t_{bi} + \delta - 1$, i.e., as long as the δ-constraint is satisfied.

   (b) Next, bi is continually increased as long as doing so does not drop the *current_weight* below γ, i.e., as long as the γ-constraint is satisfied (lines 32–35).

   (c) If ei now points to the last temporal edge instance, no further extensions to the clique time interval are possible. Therefore, the (δ, γ)-clique with t_e set to the current value of t_{ei} is duration-wise maximal and is added to result set C^s. Since no further temporal edge instances exist, the process is subsequently stopped (lines 36–39).

   Step 3b ensured that t_{bi} now indicates the last timestamp from which the γ weight can be reached in the clique interval. Similar to $tbMax$ in step 2, knowledge of this timestamp plays a vital role during the bulking phase and therefore $teMin$, set to the current t_{bi}, is stored as a clique property.

   (d) If the process was not stopped in step 3c, we next consider two possible situations. Either the indices bi and ei were updated during steps 3a and 3b, or they were not. If they were updated, we continue our process by returning to step 3a. On the other hand, if the indices remained unchanged through steps 3a and 3b, we must check whether any further growth is possible (lines 41–58). In other words, we must check if growth to the earliest time interval for which the γ minimum weight is not yet guaranteed, i.e., $[t_{bi} + 1, t_{bi} + \delta]$, is possible. If not, we know that the (δ, γ)-clique with time interval $[t_b, t_{ei}]$ is duration-wise maximal.

   Because bi was not updated in step 3b, we know that the cumulative weight for time interval $[t_{bi} + 1, t_{bi} + \delta - 1]$ is below γ. Thus, it need only be checked that a temporal edge instance exists at timestamp $t_{bi} + \delta$ and that it has sufficient weight. This leaves us with three possible scenarios, listed below.

i. First, there may exist edge instance(s) at $t_{bi} + \delta$, but with insufficient weight. In this scenario a new duration-wise maximal (δ, γ)-clique is added to C^s based on the same indices described in step 3c. If there are no further temporal edge instances beyond $t_{bi} + \delta$, then the process is stopped (lines 45–48). Otherwise (lines 49–54), further duration-wise maximal (δ, γ)-cliques may exist. To find these, the search process is resumed at step 2 with $bi = bi + 1$, ei set to the last temporal edge instance considered during the check (i.e., the last instance up to and including timestamp $t_{bi} + \delta$), and the *current_weight* set according to the range of instances denoted by these indices.
Note that we may presume that the earliest possible new duration-wise maximal (δ, γ)-clique starts at edge instance $bi + 1$. After all, if this were not the case, then such a clique would need to include instance bi and the time interval $[t_{bi}, t_{bi} + \delta - 1]$ would need to have sufficient weight. However, we have previously established this not to be the case. Additionally, since we also know that at this point $t_{bi+1} \geq t_{bi} + 1$ and $t_{ei} \leq t_{bi} + \delta$, it follows that $t_{ei} - t_{bi+1} < \delta$. Therefore, it is guaranteed that the time interval $[t_{bi+1}, t_{ei}]$ would be included in any duration-wise maximal (δ, γ)-clique starting at t_{bi+1}. Thus, by continuing our search from these indices, we guarantee future cliques are maximal on the left side of the time interval and that we do not miss any duration-wise maximal (δ, γ)-clique. In other words, it guarantees we find exclusively the set of all 2-node duration-wise maximal (δ, γ)-cliques.

ii. Second, the next temporal edge instance may be after $t_{bi} + \delta$ (lines 49–54). Again, in this scenario a new duration-wise maximal (δ, γ)-clique is added to C^s based on the same indices described in step 3c and the search process is resumed at step 2 with the indices set as described above for scenario (i).

iii. Third, there may exist edge instance(s) at $t_{bi} + \delta$ with sufficient weight. In other words, the clique time interval can be extended to include $t_{bi} + \delta$. To check for further extensions, the search process is resumed at step 3a with variables updated according to the observed valid extension (which also align with those described above for scenario (i)).

Each loop of our stretching algorithm increases at least one of the two indices in each iteration. Therefore, in the worst case we require a combined $2m$ iterations. Furthermore, given that all operations are themselves $\mathcal{O}(1)$, the time complexity of our proposed algorithm is $\mathcal{O}(m)$. This improves upon Banerjee & Pal's [6, 8] stretching

phase complexity of $O(\gamma m)$ Thus our proposed approach is more efficient and is able to process both weighted and unweighted networks. We demonstrate that our linear complexity holds in practice in Figure 4.3. The figure shows that the practical time complexity lies between a linear best case and linear worst case.

Note that $tbMax - \delta + 1$ and $teMin + \delta - 1$ are equivalent to respectively the t_b and t_e properties used in the existing temporal clique definitions (see Definition 1). We will refer to these clique properties in the remainder of this chapter as *outer borders*, or $tbMin$ and $teMax$, respectively. See Figure 4.4 for a visual representation of the various timestamp properties now defined for each duration-wise maximal (δ, γ)-clique in C^s.

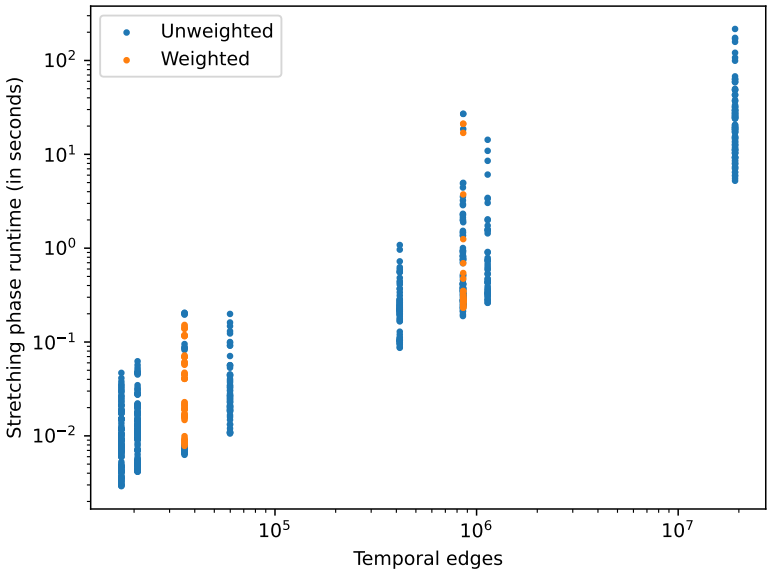


Figure 4.3: Average runtimes of the stretching phase of our algorithm for each real-world dataset (see Table 4.1) and parameter setting used in the experiments.

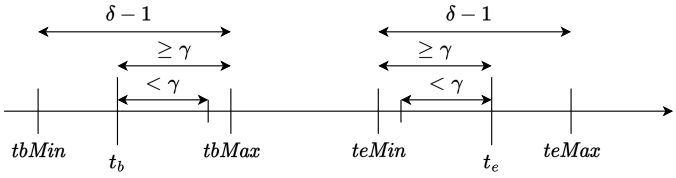


Figure 4.4: A visual representation of the clique time interval and properties on a timeline.

4.4.3 Bulking phase

Following the stretching phase, the set C^s now contains all 2-node duration-wise maximal (δ, γ)-cliques. During the bulking phase, we recursively expand the node sets of these cliques while updating their time intervals to remain duration-wise maximal, to find the set of all (δ, γ)-maximal cliques (R). Our proposed bulking method is entirely independent of γ, meaning that the value of γ does not affect the time complexity of either phase. Furthermore, the bulking phase's independence of γ, means that no special accommodations need to be considered for weighted networks. We improve upon the algorithm introduced by Banerjee & Pal [6, 8], by introducing (efficient) pruning of branches that will not include any new (δ, γ)-maximal cliques. Additionally, by following a predefined search order, determined by the node labelling, we are able to further improve our pruning efficiency. Moreover, it allows us to easily prune duplicate search branches, thereby improving on the space complexity by no longer requiring intermediate storage of all cliques of given sizes. Our proposed method has been designed so that the pruning can be computed based almost entirely on values computed for the basic recursive expansion.

We first describe the recursive clique expansion loop, duplicate branch pruning and the effective re-use of computed cliques in Section 4.4.3.1. Next, in Section 4.4.3.2 we explain how we determine the duration-wise maximal time interval overlap resulting from combining three known cliques for node expansions. Then, we discuss how we efficiently determine which branches may be pruned in Section 4.4.3.3. Finally, in Section 4.4.3.4, we argue how our chosen node labeling scheme may improve the efficiency of our approach, for example, by increasing the number of pruned branches.

4.4.3.1 Recursive search and preventing duplicate branches

The bulking process consists, at its basis, of recursive single-node extensions to the node sets of the duration-wise maximal (δ, γ)-cliques in C^s. Such a simple recursive process would however end up in many duplicate states/branches. In order to prevent us from processing such duplicate states, we only continue the recursive search for clique extensions where the new node has a higher label than all nodes in the cliques' current node set (see lines 136 and 218). Thus, node set $\{a, b, c, d\}$ can only be found by extending the clique with node set $\{a, b\}$ to $\{a, b, c\}$ and then $\{a, b, c, d\}$. Therefore, each state is visited only one time and no duplicate work is performed.

Due to slight differences for the node extension (and pruning) logic, the recursive process is split between an outer-recursion step (Algorithms 4, 5) and an inner-recursion loop (Algorithms 6, 7). In the outer-recursion step, we determine all extensions from a 2-node clique (C) to 3-node cliques based on their shared neighbors (lines 104–108). Since this essentially extends a single edge (u, v) to a triangle (u, v, w)

of three edges, we can perform this extension entirely based on the duration-wise maximal cliques already in C^s. Given sufficient time interval overlap, we combine cliques $C, C_u, C_v \in C^s$, with node sets $\{u, v\}$, $\{u, w\}$, and $\{v, w\}$, while ensuring that the resulting 3-node cliques remain duration-wise maximal (see lines 147–178).

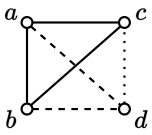
Subsequently, the inner-recursion loop extends at each step a clique (C) by one new node based on the extensions $(C_p \in E_{prev})$ that were computed in the previous recursive step (lines 187–190). We may rely on extensions from the previous recursive step since, for an extension from $\{a, b, c\}$ to $\{a, b, c, d\}$ to be possible, we must have also determined a valid extension to $\{a, b, d\}$ in the previous step (see Figure 4.5 for a visual representation). Thus, the inner-recursion loop attempts to combine valid extensions from the previous step with the clique currently under consideration. Note, that the only edge of the new extension node set ($C.X \cup C_p.X$) not yet covered by cliques C and C_p, is the edge connecting their respective extensions, i.e., the edge connecting nodes $\max(C.X)$ and $\max(C_p.X)$ (the dotted line in Figure 4.5). Thus, given sufficient time interval overlap, the inner recursion loop combines cliques C, $C_p \in E_{prev}$, and $C_e \in C^s$, where $C_e.X = \{\max(C.X), \max(C_p.X)\}$, while again ensuring that the resulting cliques remain duration-wise maximal (see lines 230–262).

Thus, through both the outer- and inner-recursive steps, we ensure that all considered cliques are duration-wise maximal. To establish whether a clique (C) is also (δ, γ)-maximal, we check that it is not temporally dominated by any of its extended cliques C_{new} (see lines 133–135 and 215–217). We say that C is *temporally dominated* by C_{new} when $C_{new}.t_b \leq C.t_b$ and $C.t_e \leq C_{new}.t_e$ (see also lines 71–76). If no such node extension exists, we know that C is a (δ, γ)-maximal clique according to Definition 4. In such cases, we store the clique C in result set R, such that at the end of our search process, R contains all (δ, γ)-maximal cliques (see lines 140–142 and 222–224). Note that the C_{new} extensions may themselves also turn out to be (δ, γ)-maximal, but with shorter timespans. Consequently, at the end of the recursive search all (δ, γ)-maximal clques are stored in the result set R.

Note that both the outer- and inner-recursion steps require combining three known cliques. Thus, the recursion effectively re-uses already computed cliques. To combine three cliques we need to determine their duration-wise maximal time interval overlap. We describe the logic and time complexity of this process in Section 4.4.3.2 below.

4.4.3.2 Duration-wise maximal time interval overlap of three cliques

In the previous subsection, we described how the recursive process relies at each step on determining the duration-wise maximal time interval overlap between three cliques. It is therefore crucial to establish a method of efficiently computing such an overlap. Let us refer to the three cliques under consideration as C_1, C_2, and C_3. We propose a



$C.X = \{a, b, c\}$
$C_p.X = \{a, b, d\}$
$C_e.X = \{c, d\}$

Figure 4.5: Inner-recursion extension visualization

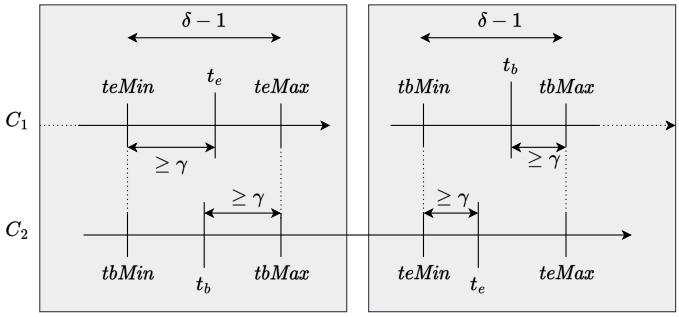


Figure 4.6: Time interval overlap extremes.

five step approach to computing the duration-wise maximal time interval overlap.

First, we determine if there exists a valid time interval overlap for C_1 and C_2. For any overlap between two time intervals to exist, we know that the left and right borders of one must be before and after the respective right and left borders of another. However, we want to check not only whether there is an overlap but also ensure that the overlap is sufficient such that at least γ weight is included for all edges in the respective node sets. Here, the previously introduced properties $tbMin$, $tbMax$, $teMin$, and $teMax$ provide our solution (see Figure 4.4 for a reminder of how these properties relate to one another). We determine if a valid overlap exists by checking whether

$$C_1.teMax \geq C_2.tbMax \text{ and } C_1.tbMin \leq C_2.teMin.$$

Proof. Figure 4.6 shows the extreme cases where $C_1.teMax = C_2.tbMax$ or $C_1.tbMin = C_2.teMin$. Note that both $tbMin$ and $teMax$ indicate the extent to which the time interval may be freely grown. Therefore, our overlap time intervals in these extreme cases would become $[C_1.teMin, C_2.tbMax]$ or $[C_1.tbMin, C_2.teMax]$. As shown in Figure 4.6, the edges for both $C_1.X$ and $C_2.X$ are guaranteed at least γ minimum weight for these time intervals. For non-extreme cases, it then follows that the overlap time intervals will be at least of equal (δ) length. After all, even if one of the time intervals themselves is less than δ in length, their potential growth is up to at least δ length due to how $tbMin$ and $teMax$ are defined. Given that we know that, for both C_1 and C_2, every δ duration has a minimum γ weight for each edge in their respective node sets, this is guaranteed to also hold for the overlap time interval for all non-extreme cases. □

Second, having determined that there exists a valid overlap between cliques C_1 and C_2, we must next determine their shared outer borders ($tbMin^{new}$, $teMax^{new}$). In other words, we determine the bounds on their combined potential time interval growth. Since the time interval overlap is defined for the combined set of edges from C_1 and

C_2, the potential growth of the time interval overlap is subject to the limitations on this growth for both C_1 and C_2. Thus, we take the maximum $tbMin$ and the minimum $teMax$, i.e., we shrink the borders to comply with both limitations.

In the third and fourth steps, we essentially repeat the first two steps but now for respectively the time interval of the overlap of C_1, C_2 and the time interval of C_3. Following our earlier proof, this checks whether there exists an overlap between cliques C_1, C_2, and C_3 such that all edges defined by their respective node sets have at least γ weight. Furthermore, the time interval $[tbMin^{new}, teMax^{new}]$ now determines the maximum time interval for which this holds. These first four steps are described in our algorithms on respectively lines 147–156 and 230–240.

For the final fifth step, we now need to determine the time interval overlap borders (t_b and t_e) within time interval $[tbMin^{new}, teMax^{new}]$, such that the resulting clique is duration-wise maximal. Recall that we already know that any time interval within $[tbMin^{new}, teMax^{new}]$ will constitute a valid (δ, γ)-clique. Therefore, the first and last edge occurrences, for any of the edges in the merged node set, within this time interval, constitute the duration-wise maximal time interval borders. In order to find these, we rely on the knowledge that each of the cliques under consideration (C_1, C_2, and C_3) are themselves duration-wise maximal, i.e., we rely on the fact that none of their edges have an occurrence before and after their respective t_b and t_e within this interval. This leads us to consider three possible scenarios, as depicted in Figure 4.7.

The first scenario is also the simplest. Here, all three t_b occur after $tbMin^{new}$ or all three t_e occur before $teMax^{new}$, i.e., they are all within the overlap time interval. Hence, all if-statements on lines 157–178 and 241–262 evaluate to "False". Consequently, we can simply choose the respective minimum and maximum values of t_b and t_e.

The second and third scenarios are more complex. Here, one or more t_b or t_e occur outside the time interval $[tbMin^{new}, teMax^{new}]$. In these cases, we can not fully rely on previously computed values. Instead, we must determine the earliest (or latest) edge occurrence in the time intervals depicted with a "?" in Figure 4.7. In these scenarios, one or two of the if statements on lines 157–178 and 241–262 must be processed. In other words, for the cliques whose t_b or t_e are outside the time interval we must determine the earliest edge occurrence, e.g., $t_b^x = \min(\{t \mid (t, u, v) \in E^{\mathcal{T}} \wedge u, v \in C_x.X \wedge t \geq tbMin^{new}\}$.

Given cliques that represent a single edge, we can determine the earliest (or latest) timestamp by performing a binary search over the edge occurrences of said edge. Such a binary search has time complexity $\mathcal{O}(\log(k))$, where k is the (average) number of temporal occurrences of an edge. For the outer-recursion step (lines 157–178), which deals solely with cliques representing a single edge, this leads to a time complexity of $\mathcal{O}(\log(k))$ for scenario 2 and $\mathcal{O}(2 \times \log(k))$ for scenario 3.

For the inner-recursion (lines 241–262), we may be dealing with more edges. In

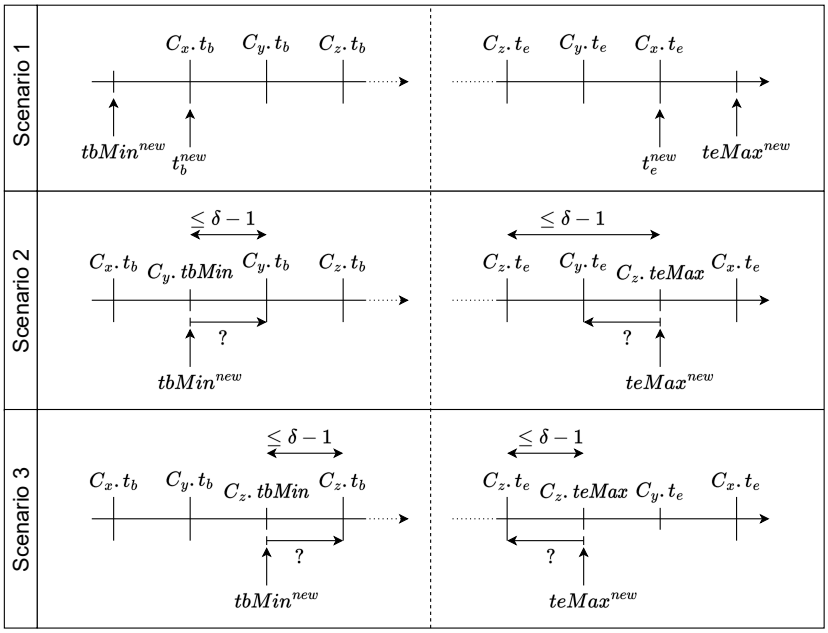


Figure 4.7: Time interval border selection scenarios.

the worst case we may need to check $\frac{|X_{new}|\times(|X_{new}|-1)}{2} - 1$ edges. This leads to a worst case time complexity of $O(2 \times (\frac{|X_{new}|\times(|X_{new}|-1)}{2} - 1) \times \log(k))$. Given this worst case time complexity, determining the new t_b and t_e of the time interval overlap, is by far the most expensive operation, in terms of time complexity, during our recursive expansion of cliques. However, note that we may stop checking edges if at any point the earliest/latest occurrence found corresponds to the outer border. Especially for temporal networks that are periodic, such as face-to-face networks where edges are created from periodic measurements (e.g., every 60 seconds), this allows us to often stop early, resulting in much better performance.

Following the fifth step that determined the new t_b and t_e, we can now define and return our new duration-wise maximal overlap clique(s) on lines 179–183 and 263–267.

4.4.3.3 Efficient branch pruning

In Section 4.4.3.1 we explained that we prevent our recursive search from processing duplicate branches by disallowing node extensions to nodes with a smaller label than the current maximum label in the node set. As a direct consequence, many branches

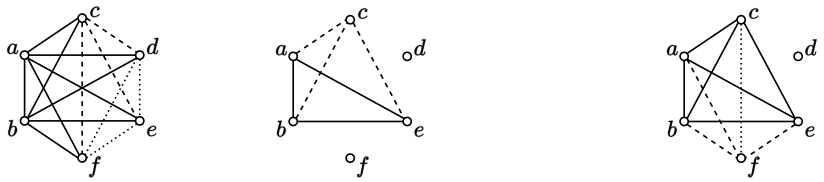
of the search tree no longer include any (δ, γ)-maximal cliques. After all, when a node extension exists for a smaller labeled node such that the current state is not maximal, this inevitably prevents us from reaching the maximal clique from that branch. Here, we discuss how we may efficiently identify and prune such branches.

For a branch to be pruned, it must be proven that neither the root state of the branch nor any of its sub-states (i.e., all recursively reachable child-states) are (δ, γ)-maximal. Specifically, given that all states in our search tree are duration-wise maximal, we only need to show that all sub-states are temporally dominated by another state with a larger node set. The vast majority of information required to show that a branch can be pruned, is obtained as a natural part of the recursive process in *neigbhoring branches*. Neighboring branches are those whose root states have the same parent state. For example, in Figure 4.8a, states $\{a, b, c\}$, $\{a, b, d\}$, $\{a, b, e\}$, and $\{a, b, f\}$ are neighboring branches with parent state $\{a, b\}$. Note that for the outer recursion step, which processes single links, neighboring branches are considered to be all cliques connecting to the shared neighbors of the link. Thus, in Figure 4.8a, when processing $\{a, b\}$ the neighboring branches are $\{a, c\}$, $\{b, c\}$, $\{a, d\}$, $\{b, d\}$, etc. Furthermore, note that only neighboring branches with smaller maximum node labels may provide the evidence needed to cut a branch. After all, only branches with smaller maximum node labels may include additional nodes. For example, for state $\{a, b, e\}$, only $\{a, b, c\}$ and $\{a, b, d\}$ may include larger nodes sets due to the inclusion of nodes c and d, which $\{a, b, e\}$ may no longer include. Thus, to determine whether branch $\{a, b, e\}$ can be pruned, we should process $\{a, b, c\}$ and $\{a, b, d\}$ first. Therefore, we enforce a search order which processes child-states in order of increasing maximum node label.

Let us assume that we are dealing with a network whose static representation is the graph shown in Figure 4.8a. Additionally, let us assume we are currently processing a clique (C) with node set $\{a, b, c\}$. Then, in Figure 4.8a the edges included in cliques in the current and neighboring branch states are shown as solid lines. Furthermore, the edges included by direct extensions to C are shown as dashed lines and the remaining edges as dotted lines. Using this as our running example, we explain how we determine whether the neighboring branch (C_b) with node set $\{a, b, e\}$ can be pruned.

First, we confirm that the root of the neighboring branch is not (δ, γ)-maximal. Since extensions to the current state already need to be computed as part of the recursive search process, this can be confirmed efficiently. After all, if, from our current state (C), an extension to C_{new} with node set $\{a, b, c, e\}$ exists that temporally dominates the root state C_b (see Figure 4.8b), then we know that C_b is not (δ, γ)-maximal. If no such extension exists, we can not determine at this stage whether the root state is (δ, γ)-maximal and we do not prune the branch (see lines 119 and 201).

Next, we need to ensure that all sub-states, i.e., all extensions, of C_b are not (δ, γ)-maximal. In our example, this means we must show that, applying the same extensions



(a) Static graph

(b) Check whether the root state is temporally dominated by inclusion of node c.

(c) Check whether extension to f is temporally dominated by considering shared (dashed) and unique (dotted) edges.

Figure 4.8: Bulking phase branch cut example

to C_b and C_{new}, maintains the latter's temporal dominance. To this end, we must first confirm whether the same extensions are possible. Note that all edges unique to C_{new} and its extensions, compared to C_b and its extensions, are those connected to node c. Figure 4.8a indicates that these unique edges are each covered by at least one of the single node extensions to the current state (dashed lines). Therefore, it is simple to establish that C and C_b have the same possible extensions, by confirming that an extension from C exists for each node to which C_b may be extended. Moreover, if C_{new} also has sufficient time interval overlap with the *minimum extension growth* of C, then we know that extensions exist from C_{new} to every node that C_b may be extended to. The minimum extension growth of C is defined as the smallest outer borders of single node extensions to C (i.e., $\max(E^r.tbMin)$ and $\min(E^r.teMax)$ with E^r the set of all single-node extensions to C). Consequently, it is then ensured that, as long as C_b's extensions are temporally dominated, any larger (multi-)node extensions must either exist for both the current state and the branch root state or for neither. Thus, in our example we need only confirm that the extension to $\{a, b, c, f\}$ exists and has sufficient time interval overlap with C_{new}. If this holds, we say that C_b is *spatial growth dominated* by C_{new} (see lines 77–84, 121, and 203).

Upon confirming that C_b is spatial growth dominated, we next need to show that its extensions are also temporally dominated, i.e., that C_b is *temporal growth dominated*. We may determine this by analysing the impact on the potential growth of the time interval, by edges that are added through node extensions. Recall from Section 4.4.3.2, that the addition of edges, i.e., combining cliques, may either maintain, grow, or shrink the time interval ($[t_b, t_e]$). Importantly, the potential growth of the interval borders is only limited by the outer borders ($tbMin$ and $teMax$) and the outer borders themselves may only shrink with edge additions. Moreover, the time interval may only shrink if the outer border shrinkage restricts it as such (e.g., in scenarios 2 and 3 in Figure 4.7).

To determine whether C_b is temporal growth dominated, we need to establish the effects of all edges with which it may be extended. For our example, Figure 4.8c shows

the only possible extension to C_b, i.e., the extension to node f. Here, the newly added edges are depicted with dashed and dotted lines. Specifically, the dotted lines show the edges that are uniquely added through extensions to C_{new}. These unique edges are the only reason that the outer borders of C_{new} may be shrunk further than the outer borders of C_b. Note that all of these edges are included in at least one of the single-node extensions (E^r). Therefore, the potential shrinkage of the outer borders of C_{new} through extensions, is determined by the minimum extension growth of C. To determine whether the branch root state is temporal growth dominated, we thus check if its actual potential growth may exceed the minimum extension growth of C (see lines 85–101).

The most obvious reason for the actual temporal growth of C_b to not exceed the minimum extension growth of C, is for its outer borders to be temporally dominated by the minimum extension growth (see lines 86–88). If this is not the case, we must check that none of the potential extensions to C_b have time interval borders that exceed the minimum extension growth (see lines 89–99). Specifically, this means we must check the time intervals of cliques representing edge(s) with which C_b may be extended (i.e., cliques representing the dashed edges in Figure 4.8c). These edges can be split into two categories: those covered by neighboring branches (e.g., (a, f) and (b, f) in $\{a, b, f\}$); and those connecting the extension nodes (e.g., edge (e, f)). For the former, we select the relevant cliques, that have sufficient time interval overlap to be used in a potential extension to C_b, from those previously computed on lines 115 and 197, and subsequently check whether their time intervals exceed the minimum extension growth on lines 89–91. Only if this is not the case, do we check the latter edges, i.e., those connecting the node extension cliques (see lines 92–99). Selecting relevant cliques from C^s for these edges and checking them against the minimum extension growth, is the only part of our pruning process that requires us to consider cliques not essential to the recursive search at that time. By checking these last, we minimize how often we need to consider "new" cliques. If none exceed the minimum extension growth of C, than C_b is, practically speaking, temporal growth dominated and the branch may be pruned.

In summary, we determine whether a neighboring branch may be pruned by confirming that its root state is temporally dominated, spatial growth dominated, and temporal growth dominated. We can determine this by relying almost exclusively on information that is readily at hand in the current state of the recursive search.

4.4.3.4 Custom node labeling for efficient search and pruning

Here, we discuss how a custom node labeling can be used to improve the efficiency of the bulking phase. Recall that the bulking process always starts from a 2-node clique

adding a single node at a time. Consequently, both the (potential) depth and width of the recursive search trees are dependent on the number of shared neighbors of the two nodes. Furthermore, recall that our recursive process skips any extensions to nodes with lower labels. Therefore, we can reduce the average depth and width of the recursive search trees by providing the nodes with the most neighbors the highest labels, thereby visiting them last. Although we still visit each state exactly one time, regardless of which node labeling is used, there are benefits from the reduced width of the search trees for the pruning process. Each node that a clique may be extended to adds conditions that must be checked and satisfied for a branch to be pruned. Therefore, having fewer potential extensions means there are fewer conditions to satisfy. Additionally, this custom node labeling lends itself better to future parallelization. After all, by reducing the width and depth of the most extreme search trees, the duration of processing each individual search tree, for the cliques in C^s, becomes more balanced. Parallelization can utilise this balance, by dividing parts of the C^s set over multiple processes with less risk of one process being excessively slower than the others.

Thus, in this chapter we use a custom node labeling based on the number of neighbors of a node, with fewer neighbors leading to a lower label.

4.5 Data

For our experiments we collected a mixture of smaller network datasets that were used in previous research to evaluate temporal clique enumeration algorithms, as well as a new set of larger networks. Aside from their size, we selected the datasets so that (weighted) temporal cliques would have meaning within their respective contexts, i.e., so that clique results can be meaningfully interpreted. All datasets were pre-processed to sort by timestamp and exclude self-edges. Resulting size statistics and some basic information on the networks, including their source, are provided in Table 4.1.

The `Hypertext`, `Infectious I`, and `Infectious II` datasets are periodic networks of face-to-face interactions between conference attendees. Each edge indicates that two conference attendees were in close physical proximity to one another at a given time, with measurements having occurred at specific intervals. Cliques in these networks may represent groups of attendees having an ongoing conversation. Temporal cliques specifically, may allow us to more accurately capture these groups by excluding people who happen to pass by the group at the time of one measurement.

We also included several non-periodic datasets. Two of them capture co-listening of songs (`Last.fm songs`) and bands (`Last.fm bands`) by users of the Last.fm social network. Temporal cliques in these networks may indicate a lasting shared interest in

Table 4.1: Summary of dataset statistics. Network type abbreviations: *f2f* = face-to-face, *com* = communication, *trr* = trust rating, *hyp* = hyperlink, and *cli* = co-listening. The "Wei." column indicates whether the dataset includes edge weights and the "Per." column indicates which datasets are periodic.

Dataset	$\|V\|$	$\|E\|$	$\|E^{\mathcal{T}}\|$	Lifetime	Type	Wei.	Per.
Hypertext [76, 1]	113	2,196	20,818	2.5 days	*f2f*	No	Yes
College message [115, 93]	1,899	13,838	59,835	193 days	*com*	No	No
Bitcoin [87, 86, 93]	5,881	21,492	35,592	5.21 years	*trr*	Yes	No
Infectious I [76, 1]	10,972	44,517	415,912	80 days	*f2f*	No	Yes
Infectious II [76, 1]	410	2,765	17,298	8h	*f2f*	No	Yes
Facebook-wosn-wall [142, 88]	45,813	183,412	855,542	4.28 years	*com*	No	No
Reddit hyperlinks [85, 93]	67,180	309,667	858,488	40 months	*hyp*	Yes	No
Enron [82, 88]	86,978	297,456	1,134,990	4.5 years	*com*	No	No
Last.fm bands [88, 83]	174,077	894,388	19,146,398	4.35 years	*cli*	No	No
Last.fm songs [88, 83]	1,084,602	4,413,377	19,150,606	4.35 years	*cli*	No	No

the same songs or bands by groups of users. Additionally, we included three communication networks, the `College message`, `Facebook-wosn-wall`, and `Enron` datasets. Respectively, they capture the private messages sent on an online social network at the University of California, Facebook wall posts between online social media users, and email communication between employees of the Enron Corporation. Temporal cliques in these networks may indicate friend groups or (Enron) departments, between whom more frequent communication with all members is to be expected.

To experiment with edge weighting, two weighted datasets were included. The first dataset captures the trust between Bitcoin users (`Bitcoin`) through ratings of one another. The second dataset consists of hyperlinks between subreddits (`Reddit hyperlinks`), where weight determines the sentiment (positive or negative) that accompanied the source subreddit. By requiring a minimum weight on the (temporal) cliques we can find sufficiently strongly connected groups of users and subreddits, where this is determined by the actual strength of the connections and not simply their frequency. For `Reddit hyperlinks`, this allows us to identify those groups of subreddits whose on balance sentiment towards one another is exclusively positive.

Finally, to analyze how our algorithm performs for increasingly dense networks, we generated a set of synthetic temporal networks. Since we enumerate temporal cliques, we must consider both the structural density as well as the temporal density. For any given synthetic temporal network, we use the well-known Erdös-Rényi [122] and Barabasi-Albert [10] random graph models to generate the underlying static networks to control for the structural density. Each static edge is assigned at least one timestamp and a Poisson distribution is used to determine the number of additional timestamps. The exact timestamps are then determined such that they are either: (1) temporally

close to a timestamp of neighboring edges, to enforce temporal density and simulate group connectivity; or (2) temporally close to already determined timestamps for the same static edge, in order to simulate the burstiness of temporal networks [9]. By choosing between these two options for each additional timestamp, we also indirectly try to simulate the long pauses that can exist between bursts of activity. For both options the new timestamp is determined using a normal distribution centered on the existing timestamp to which we aim to be temporally close. We use a sigma that is half the size for the former option compared to the latter, to ensure that the simulated group interactions indeed have very close, if not identical, timestamps. Pseudo code describing the exact process for generating synthetic temporal network we used is provided in Algorithm 8 in Section 4.8.1.

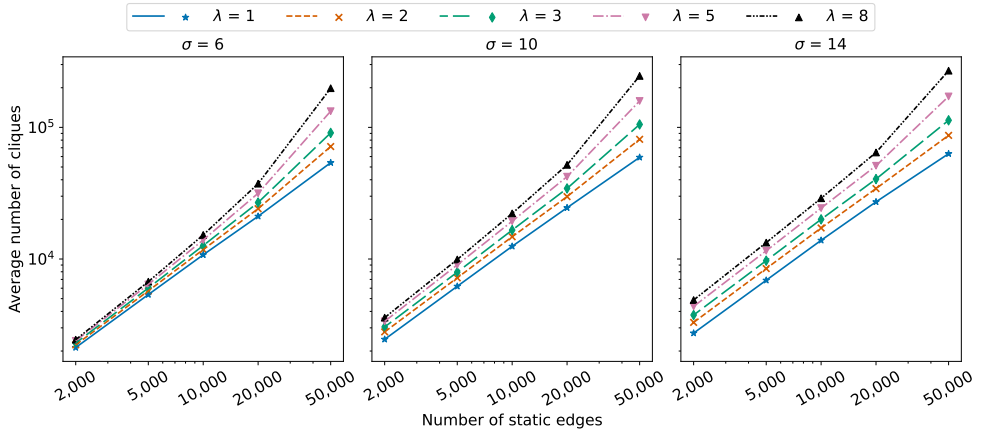
Synthetic temporal networks were generated for a range of parameter settings, but always with $1,000$ nodes. The structural density settings were chosen to approximate densities of $[0.002, 0.005, 0.01, 0.02, 0.05]$, i.e., average degrees ranging between 2 to 50. For the temporal density settings, λ values for the Poisson distribution were chosen from $[1, 2, 3, 5, 8]$ and σ values for the Normal distributions from $[6, 10, 14]$. For both Erdös-Rényi and Barabasi-Albert random graphs and for each combination of parameter settings, a hundred synthetic temporal graphs were generated. Figure 4.9 shows for each of the parameter settings the average number of cliques found. Additional clique properties are reported in Figures 4.19 and 4.20 in Section 4.8.2.

4.6 Experiments

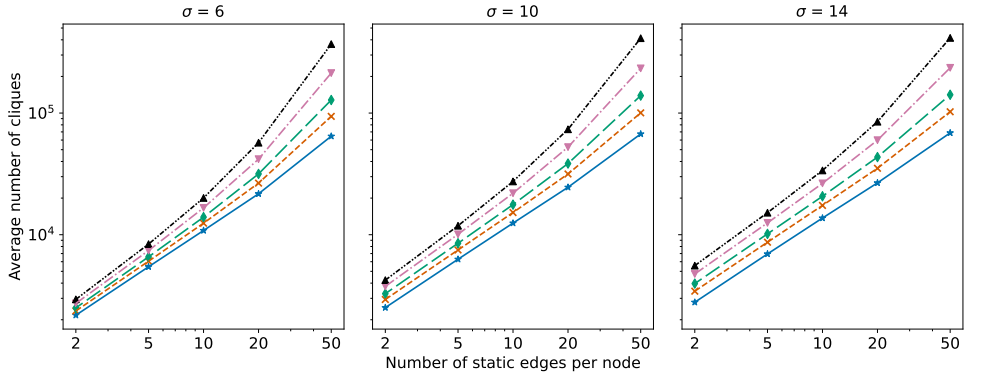
In this section we present the experimental evaluation of our algorithm, in which we compare our algorithm to existing (state-of-the-art) methods. In the following subsections we discuss, respectively, the experimental setup and results.

4.6.1 Experimental setup

As discussed in Section 4.2, several algorithms have been proposed for temporal clique enumeration since the first introduction of δ-cliques. We evaluated our proposed algorithm against each of them for the $\gamma = 1$ setting, i.e., the purely temporal setting. Specifically, this means we compare against the algorithms introduced by Viard et al. [140], Himmel et al. [70], and the two-phase approach algorithm of Banerjee & Pal [8] that inspired our method. For higher γ values, we can only compare to the algorithm proposed by Banerjee & Pal [8]. Since these implementations rely on the existing (non-instance-bounded) definitions of temporal cliques (see Section 4.3.2), a post-processing step is required to obtain the set of cliques that satisfies Definition 4. The implemented post-processing step is simple but inefficient. Therefore, the time



(a) Erdös-Rényi random graphs



(b) Barabasi-Albert random graphs

Figure 4.9: Average number of (δ, γ)-maximal cliques for each density parameter setting.

required for the post-processing and storing of result cliques, are excluded from the runtimes we report. The reported runtimes do include the time required to read in the dataset. As an additional consequence of the difference in definition, the δ value reported alongside the results is the one used by our algorithm, while the existing algorithms were executed with a value one lower.

For Banerjee & Pal [8], we applied a minor fix to their implementation, which for larger γ did not properly account for cases where the same link occurs multiple times at the same timestamp. Subsequently, for all experiments, Himmel et al. [70] and Banerjee & Pal [8] obtained, after post-processing, the same sets of cliques as our proposed algorithm. The implementation of Viard et al. [140], however, did not

implement the exact periodicity phase discussed in Section 4.3.2. Instead, at a δ representing exact periodicity, it provides the results of pre-periodicity. Since we have chosen δ values that often correspond to exact periodicity, this means that the clique set provided by Viard et al. [140] did not match. Despite this and for the sake of a complete comparison with existing methods, we still report on their performance.

All experiments were performed on a server with 16 Intel Xeon E5-2630v3 cores and 512GB RAM. However, note that all implementations, including that of our own algorithm, run sequentially and therefore only used a single thread. Alongside runtimes, we report maximum space usage based on the Python "resource" package using the *ru_maxrss* property. Furthermore, note that due to hardware technicalities, space usage below 76MB is measured as 76MB. Since cases with such low memory usage are of little interest, we simply report all such cases as "$\leq$ 76". A 24 hour time limit and a 200GB memory usage limit were applied, before post-processing. Consequently, some results are missing and an indication of which limit was exceeded is shown. All runtime and memory statistics reported in this paper were averaged over five runs.

The (modified) implementations of all algorithms used in the experiments of this study can be found at `https://github.com/hdboekhout/Fast-delta-gamma-clique`.

4.6.2 Results

For our experiments we consider the basic temporal setting in Section 4.6.2.1. We then compare our proposed algorithm to that of Banerjee & Pal [8] for larger γ values in the unweighted setting in Section 4.6.2.2. Next, in Section 4.6.2.3 we compare the weighted and unweighted settings for our algorithm. Finally, we analyze the runtime performance of our algorithm, on its own and relative to Banerjee & Pal, on synthetic networks of increasing structural and temporal density in Section 4.6.2.4.

4.6.2.1 The basic temporal setting

To evaluate the performance of our proposed algorithm, we first compare its performance against existing δ-maximal clique enumeration algorithms, i.e., the basic temporal setting. For our algorithm, this means we enumerate $(\delta, 1)$-maximal cliques in unweighted (versions of the) networks. Each algorithm was applied to each dataset from Table 4.1 for a range of δ values. For the periodic datasets (i.e., `Hypertext`, `Infectious I`, and `Infectious II`), the δ values were chosen in line with the periodicity of measurements. For the non-periodic datasets, δ values were chosen as a set number of hours, days, months, or even years. For each dataset and δ value, we report in Tables 4.2 and 4.3 the number of $(\delta, 1)$-maximal cliques found (that satisfy Definition 4), the maximum cardinality (i.e., number of nodes in the largest clique) and

maximum time interval duration among those cliques, and the runtime (in seconds) and maximum RAM usage (in MB) of each algorithm.

Results from Tables 4.2 and 4.3 show that for all but a few instances, our proposed algorithm is faster than existing methods. Especially for larger datasets and at relatively low δ values, we significantly outperform Banerjee & Pal. Note that, for the instances where we do not improve upon them, we still required fewer iterations, i.e., processed search tree states (see the left panel of Figure 4.10). However, depending on the specific scenario, computing the time interval borders of our corrected definition of (δ, γ)-cliques (as described in Section 4.4.3.2), may lead to computationally heavier iterations than for Banerjee & Pal. Thus, although rare, when insufficient pruning occurs to offset this, our algorithm can under perform.

Figure 4.10 shows the relationship between the number of iterations (i.e., search tree states) that must be processed and the runtime. For each dataset we observe that, as the number of required iterations increases, the relative amount of time spent on node expansion computation increases as well. Furthermore, the time spent on node expansion tends to far exceed that spent on the pruning decision making, thus highlighting its relative efficiency. Two factors appear to have the most significant impact on the performance of our algorithm. First, the maximum cardinality, which

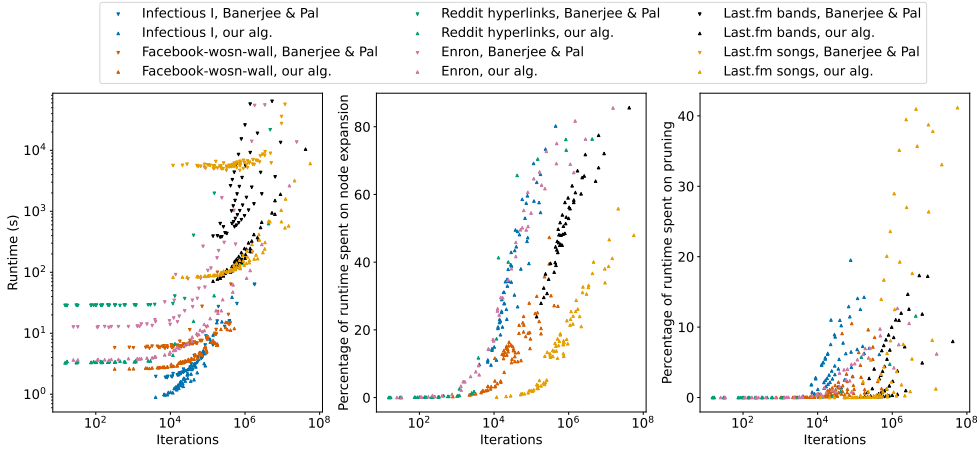


Figure 4.10: Runtime vs. the number of iterations (search space states) that are processed (left panel). The centre and right panel show, respectively, the share of runtime of our proposed algorithm used on node expansion and pruning decision making with respect to the number of iterations. Results for all δ and γ values tested and reported on in this results section for the given datasets are included, except when they resulted in zero (δ, γ)-maximal cliques.

Table 4.2: Results for the maximal $(\delta, 1)$-clique count (N), maximum cardinality (C), maximum clique duration (D), and the computation time (in seconds) and maximum space usage (in MB) for the small datasets.

		Clique properties			Viard et al. [140]		Himmel et al. [70]		Banerjee & Pal [8]		Our algorithm	
Dataset	δ	N	C	D	Runtime (s)	Max. RAM (MB)	Runtime (s)	Max. RAM (MB)	Runtime (s)	Max. RAM (MB)	Runtime (s)	Max. RAM (MB)
Hypertext	60	7,001	7	7,521	13.98	188	3.70	≤76	29.58	116	**0.45**	≤76
	120	5,697	7	7,901	14.73	177	2.88	≤76	13.59	115	**0.41**	≤76
	180	5,106	7	11,161	16.54	195	2.60	≤76	8.84	115	**0.39**	≤76
	240	4,727	7	11,161	16.89	195	2.36	≤76	7.22	116	**0.39**	≤76
	300	4,479	7	11,161	17.47	198	2.25	≤76	5.33	116	**0.38**	≤76
	360	4,257	7	11,161	18.27	206	2.16	≤76	4.80	116	**0.39**	≤76
	420	4,112	7	11,161	19.28	217	2.12	≤76	4.28	116	**0.39**	≤76
	480	3,989	7	11,161	19.96	225	2.06	≤76	4.10	115	**0.39**	≤76
	540	3,899	7	16,521	20.92	236	2.02	≤76	3.74	116	**0.39**	≤76
	600	3,815	7	16,521	22.42	256	2.00	≤76	3.51	116	**0.39**	≤76
College message	3,600	33,350	4	14,562	25.82	249	25.07	≤76	11.07	150	**0.61**	≤76
	43,200	23,713	5	316,619	31.82	422	19.50	≤76	2.64	145	**0.54**	≤76
	88,640	19,925	5	718,855	38.69	605	17.92	≤76	1.57	143	**0.54**	≤76
	259,200	16,262	5	1,804,213	60.71	1,163	16.66	≤76	1.05	143	**0.56**	≤76
	604,800	14,583	6	5,124,654	103.34	2,342	16.17	≤76	0.93	144	**0.66**	≤76
Bitcoin	3,600	25,541	7	3,592	13.82	111	124.77	≤76	1.88	158	**0.60**	≤76
	43,200	24,178	8	47,301	14.85	123	125.00	≤76	1.86	158	**0.64**	≤76
	88,640	23,580	8	168,288	15.57	138	124.84	≤76	1.84	159	**0.67**	≤76
	259,200	22,558	8	259,173	17.37	174	125.49	≤76	1.83	160	**0.71**	≤76
	604,800	21,608	8	610,794	22.16	252	125.34	≤76	1.92	162	**0.80**	≤76
Infectious I	60	115,612	6	3,641	168.67	2,991	466.49	156	27.32	329	**6.36**	229
	120	88,394	6	4,941	252.29	4,291	461.02	143	15.83	328	**6.86**	210
	600	68,807	10	10,001	1,618.69	21,795	465.47	140	**12.87**	481	15.42	189
	1,200	61,530	13	10,001	8,493.14	90,185	485.72	152	**30.63**	923	39.05	183
	6,000	21,720	16	10,741	$> 200GB$		581.36	165	63.38	2,846	**14.94**	139
	12,000	20,749	16	11,301	$> 200GB$		580.86	165	63.08	2,864	**5.21**	137
Infectious II	60	6,446	5	1,741	5.80	114	2.12	≤76	1.88	117	**0.35**	≤76
	120	5,358	6	2,661	10.08	186	1.88	≤76	0.96	118	**0.42**	≤76
	180	5,227	6	3,921	14.94	271	1.79	≤76	0.73	119	**0.50**	≤76
	240	5,240	7	4,681	21.76	372	1.79	≤76	0.68	121	**0.58**	≤76
	300	5,334	8	4,681	29.93	487	1.84	≤76	0.72	122	**0.67**	≤76
	360	5,512	8	5,761	40.46	625	1.94	≤76	**0.78**	126	0.81	≤76
	420	5,835	8	5,761	55.27	825	2.09	≤76	**0.85**	131	0.95	≤76
	480	6,004	9	7,581	71.67	1,034	2.25	≤76	**1.04**	134	1.13	≤76
	540	6,133	9	8,501	94.75	1,302	2.42	≤76	**1.08**	137	1.30	≤76
	600	6,266	9	8,501	121.87	1,625	2.60	≤76	**1.19**	140	1.50	≤76

Table 4.3: Results for the maximal $(\delta, 1)$-clique count (N), maximum cardinality (C), maximum clique duration (D), and the computation time (in seconds) and maximum space usage (in MB) for the larger datasets.

		Clique properties			Viard et al. [140]		Himmel et al. [70]		Banerjee & Pal [8]		Our algorithm	
Dataset	δ	N	C	D	Runtime (s)	Max. RAM (MB)	Runtime (s)	Max. RAM (MB)	Runtime (s)	Max. RAM (MB)	Runtime (s)	Max. RAM (MB)
Facebook-wosn-wall	86,400	506,882	4	3,373,951	305.85	2,547	10,286.43	532	562.87	771	**11.97**	619
	604,800	380,549	5	21,687,693	397.67	4,488	10,354.96	469	63.70	704	**11.27**	553
	2,628,000	281,712	6	79,749,744	757.84	11,377	10,492.18	425	20.31	661	**11.52**	493
	7,884,000	218,340	7	118,451,848	1,724.46	30,867	10,539.35	406	14.79	660	**12.41**	470
	31,536,000	170,460	9	129,313,589	8,333.04	200,573	10,287.60	381	**14.00**	691	15.08	459
	157,680,000	131,066	10	131,845,082	$> 200GB$		10,375.97	380	14.68	744	**11.63**	433
Reddit hyperlinks	3,600	839,208	5	5,190	3,952.44	1,988	28,488.17	825	> 24h		**140.38**	4.765
	86,400	789,675	9	255,291	4,326.71	3,582	28,385.90	833	> 24h		**151.55**	4.923
	604,800	701,663	17	73,872,286	31,687.53	127,349	27,969.37	886	21,697.68	12,109	**628.46**	5.143
Enron	3,600	983,610	12	29,588	2,761.12	21,014	43,035.89	966	> 24h		**80.17**	2.664
	86,400	721,768	13	1,054,487	8,489.00	73,895	41,450.11	926	14,019.07	2,439	**107.12**	2.288
	604,800	591,242	15	39,517,939	$> 200GB$		39,910.54	895	55,401.08	5,981	**286.77**	1.899
	2,628,000	599,179	18	66,570,802	$> 200GB$		42,054.39	1,009	13,654.13	49,110	**2,634.49**	1.732
Last.fm bands	86,400	5,937,774	5	7,213,683	> 24h		> 24h		> 24h		**897.09**	26,059
	604,800	4,101,745	7	79,771,958	> 24h		> 24h		63,689.40	10,251	**944.14**	20,509
	2,628,000	4,899,608	10	134,178,398	> 24h		> 24h		13,433.54	24,517	**1,904.25**	22,541
	7,884,000	12,153,968	14	136,945,742	> 24h		> 24h		18,478.72	135,215	**10,458.64**	43,456
Last.fm songs	86,400	14,715,165	4	3,342,331	> 24h		> 24h		> 24h		**575.71**	20,637
	604,800	11,552,954	5	24,493,086	> 24h		> 24h		56,838.23	16,692	**536.07**	17,565
	2,628,000	8,647,218	6	120,376,971	> 24h		> 24h		35,852.11	15,040	**675.74**	14,769
	7,884,000	7,429,520	8	134,177,875	> 24h		> 24h		27,454.15	16,404	**1,044.68**	14,037
	31,536,000	9,853,895	11	136,828,008	> 24h		> 24h		22,874.81	38,107	**3,186.39**	17,576
	157,680,000	10,491,336	17	156,295,069	> 24h		> 24h		$> 200GB$		**6,078.19**	17,644

can drastically increase the number of iterations; and second, the effect of the chosen δ on the extent to which we may prune the search space. Results from Tables 4.2 and 4.3 suggest that, when different δ values lead to similar maximum cardinality, performance is lowest for mid-sized δ. For smaller δ there are relatively smaller cliques in general, but there is also less overlap between the time intervals to account for, leading to more pruning. Meanwhile, for larger δ that approach (or exceed) the full lifetime of the dataset, temporal growth dominance becomes a given, which aids pruning.

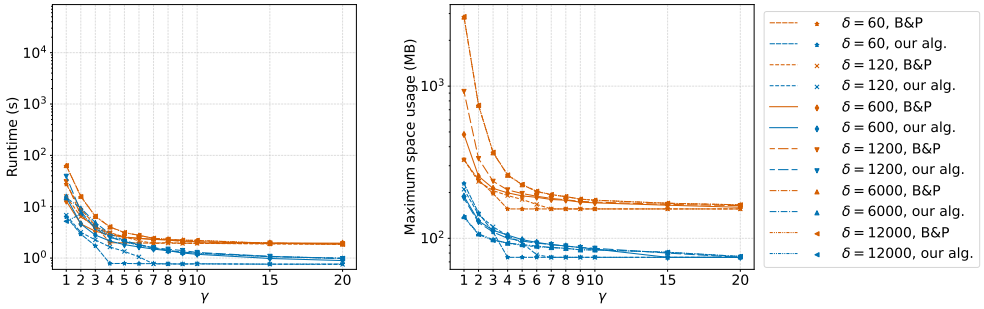
In terms of space (i.e., RAM) requirements, we observe that our algorithm generally requires less memory than Banerjee & Pal. Only on larger datasets with many (δ, γ)-maximal cliques, do we sometimes require more space due to the additional clique properties that are stored. This could easily be compensated for by writing the (δ, γ)-maximal cliques to a file as soon as they are found, instead of storing them in a set during enumeration. After all, unlike existing methods, we prevent duplicates through our search order, removing the necessity of checks for duplicates. However, we did not implement this in order to maintain a fair comparison with existing methods for our experiments and to allow for easier integration with future coding.

Compared to Himmel et al., we require more space due to the use of additional data structures. However, Table 4.3 shows that the approach by Himmel et al. requires more computation time and scales poorly to larger datasets. Finally, Viard et al. shows worse performance in both runtime and space requirements, with space requirements exploding for larger δ.

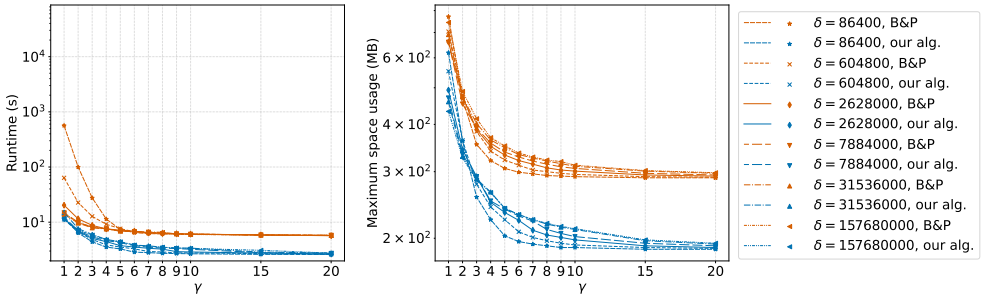
4.6.2.2 Comparison for larger γ in the unweighted setting

Having demonstrated that our algorithm outperforms existing algorithms in (nearly) all instances in the basic temporal setting, we next compare against Banerjee & Pal's algorithm for a range of γ values in the unweighted setting. Performance results for the six largest datasets are shown in Figures 4.11 and 4.12. Information on the (δ, γ)-clique properties are provided in Figures 4.17 and 4.18 in Section 4.8.2.

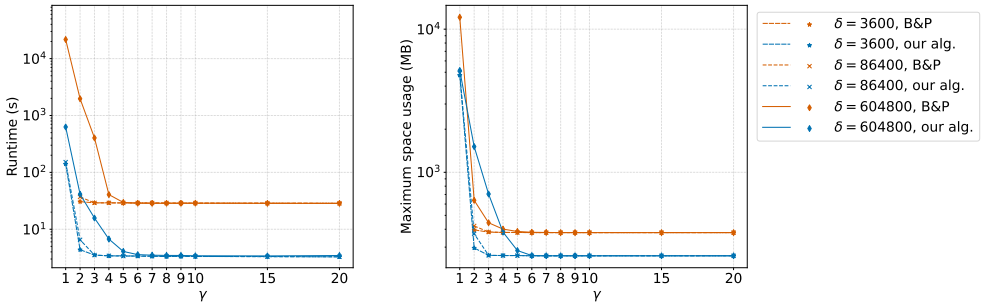
The results clearly demonstrate that, as γ increases, the number of cliques and consequently the runtime and memory requirements decrease. Furthermore, they demonstrate that our algorithm outperforms Banerjee & Pal both computationally and in space requirements. Moreover, for the instances where Banerjee & Pal performed better at $\gamma = 1$, our algorithm performs better at higher γ values.



(a) Infectious I

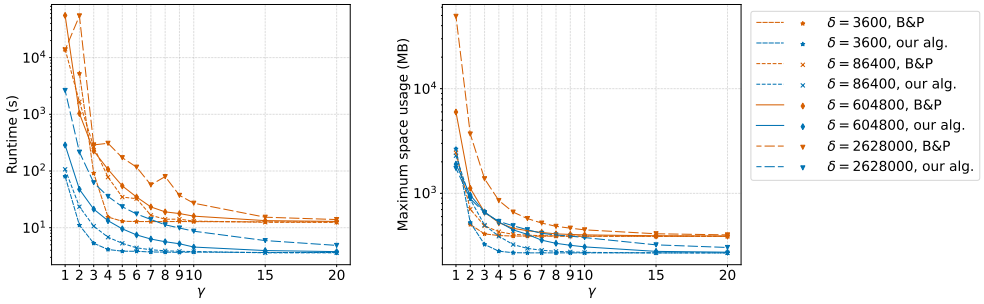


(b) Facebook-wosn-wall

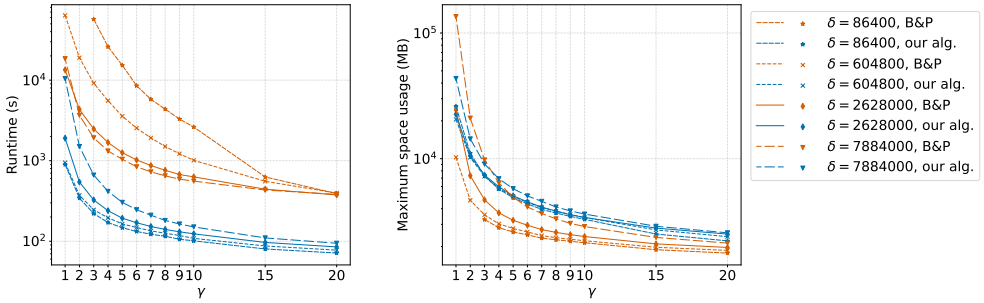


(c) Reddit Hyperlinks

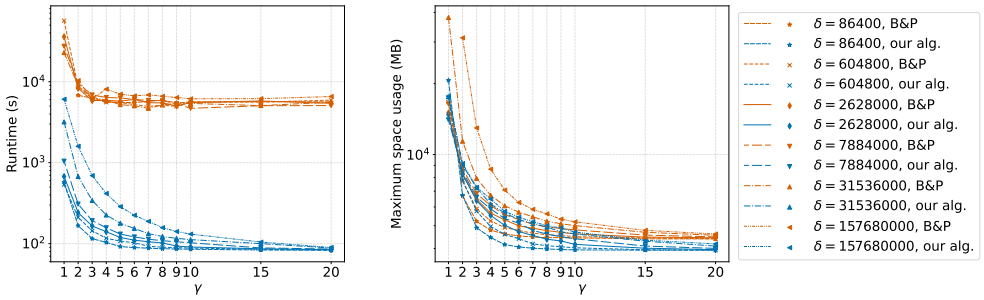
Figure 4.11: Performance comparison between Banerjee & Pal [8] and our algorithm for increasing γ. Results for $\gamma = 1$ correspond to those reported in Tables 4.2 and 4.3.



(a) Enron



(b) Last.fm bands



(c) Last.fm songs

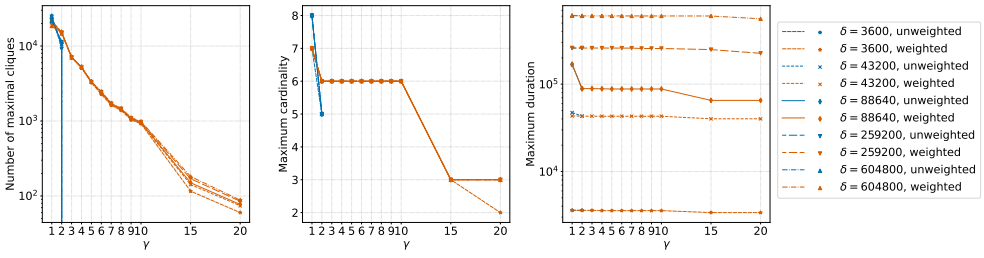
Figure 4.12: Performance comparison between Banerjee & Pal [8] and our algorithm for increasing γ. Results for $\gamma = 1$ correspond to those reported in Tables 4.2 and 4.3.

4.6.2.3 Comparing the weighted and unweighted settings

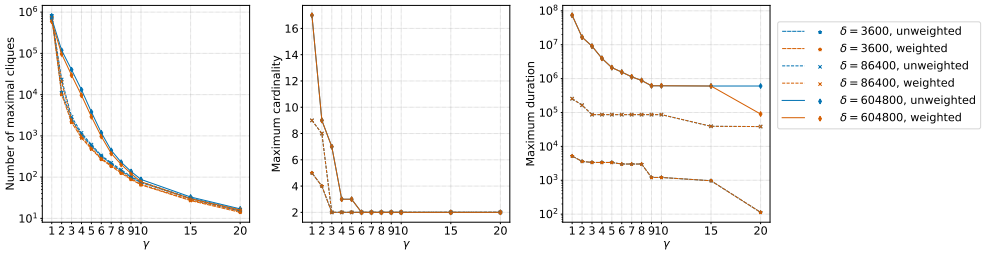
Next, we compared the weighted and unweighted settings for the two weighted datasets (`Bitcoin` and `Reddit hyperlinks`), given identical δ and γ values. Figure 4.13 shows the number of maximal cliques, the maximum cardinality, and the maximum duration of the enumerated cliques.

For `Bitcoin` in the unweighted setting, we observe that there exist no node pairs with more than two edges connecting them within a week (our largest δ). Therefore, no more cliques are found at $\gamma > 2$. On the contrary, given that `Bitcoin` weights range from -10 to 10, the weighted setting provides a larger range of results. We observe that as γ increases, thereby requiring increasingly positive relations between the clique members, we find fewer and fewer maximal cliques. Thus, in the weighted setting, γ can be used to fine-tune exactly how positively we would like the group members to rate one another, whereas in the unweighted setting we determine how frequently ratings must occur between members irrespective of their sentiment.

Unlike `Bitcoin` that has a range of weights, `Reddit hyperlinks` has but two possible weights, -1 and 1. As a consequence, the difference in results between the weighted and unweighted settings is much smaller. We observe that the weighted setting always finds fewer cliques, since cliques that include negative ties get excluded.

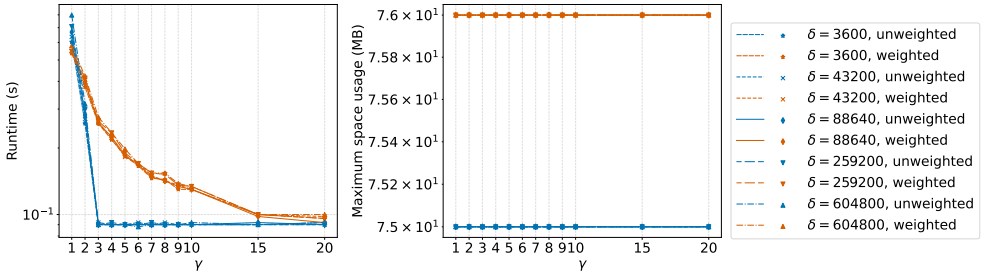


(a) Bitcoin

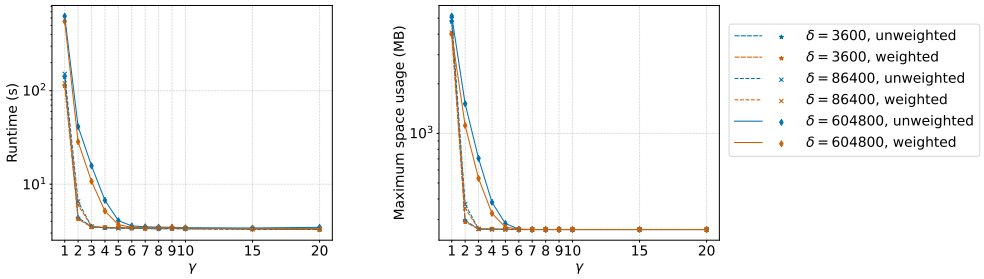


(b) Reddit Hyperlinks

Figure 4.13: Properties of enumerated (δ, γ)-maximal cliques, comparing the weighted and unweighted interpretation of two networks for the same δ and γ values.



(a) Bitcoin



(b) Reddit Hyperlinks

Figure 4.14: Performance of (δ, γ)-maximal clique enumeration, comparing the weighted and unweighted interpretation of two networks for the same δ and γ values.

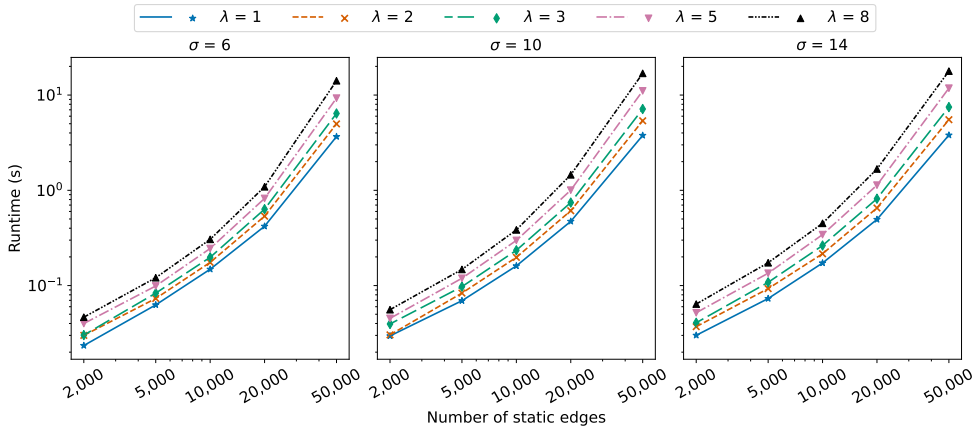
Thus, whereas for `Bitcoin` the weighted and unweighted settings provided different uses, for `Reddit hyperlinks`, the weighted setting provides more of a refinement (or filtering) on the unweighted results. Interestingly, in most cases this refinement does not seem to affect the maximum cardinality nor the maximum duration of the cliques.

Performance results for the weighted and unweighted settings are shown in Figure 4.14. The figure shows that the slight decrease in (δ, γ)-maximal cliques we observed for `Reddit hyperlinks`, corresponds to a similar slight decrease in runtime and space usage. This confirms that γ itself has no bearing on the runtime or space complexities of our algorithm.

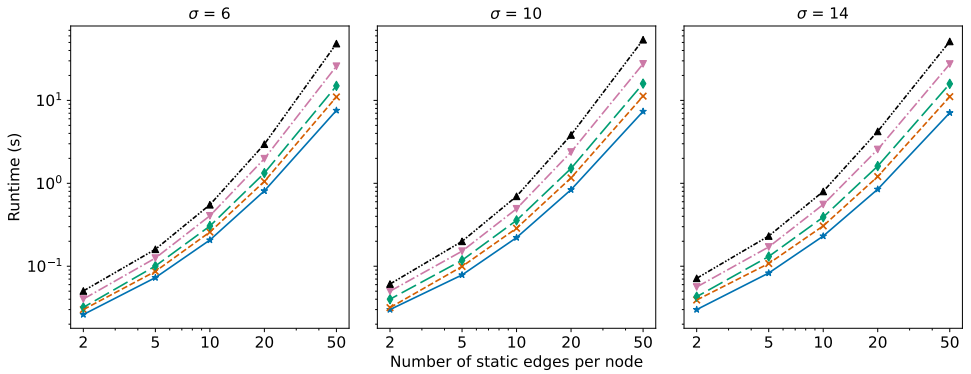
In short, the results demonstrate that the weighted setting allows better fine-tuning of the (positive) strength of connections beyond mere frequency. In other words, the weighted setting allows us to consider the sentiment of a connection when forming cliques. Furthermore, results confirmed that γ itself (whether weighted or unweighted) has no direct impact on the runtime or space complexities of our algorithm. However, due to its influence on the number of (valid) (δ, γ)-cliques, it does impact the performance indirectly.

4.6.2.4 Performance on synthetic temporal network densities

So far, we have established that our proposed algorithm, in all but a few cases, outperforms existing state-of-the-art algorithms. However, we also observed that performance suffered for larger maximum clique cardinality and for mid-size δ. This naturally raises the question how our proposed algorithm performs for increasingly dense networks. Therefore, we analyze the performance of our algorithm for a set of synthetic temporal networks with increasing structural (i.e., static) and temporal densities. Figure 4.15 shows the average runtime of our algorithm for $\delta = 10$ over, respectively, a hundred Erdös-Rényi and Barabasi-Albert random graphs for each



(a) Erdös-Rényi random graphs



(b) Barabasi-Albert random graphs

Figure 4.15: Average runtime (in seconds) for $\delta = 10$ for each density parameter setting for the synthetic datasets. All plots are log-log plots.

structural and temporal density parameter setting. Here, we use σ values that are less than, equal to, and greater than the chosen δ, as well as an initial timestamp range that is ten times the chosen δ (i.e., $[100, 200]$), to simulate mid-size δ performance.

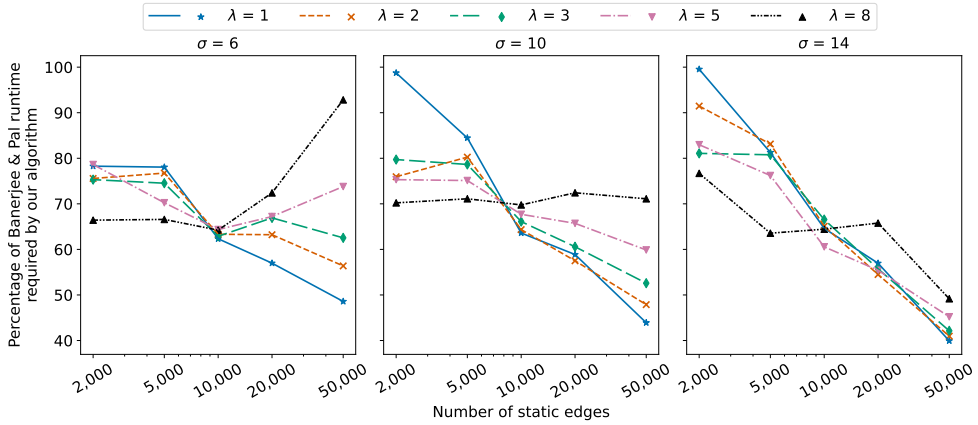
Figure 4.15 shows that as the structural and temporal densities increase, so does the runtime. We observe a sharper increase for Barabasi-Albert random graphs. However, this is to be expected as Barabasi-Albert random graphs also demonstrate more significant increases in average and maximum clique cardinality than Erdös-Rényi random graphs (see Figures 4.19 and 4.20 in Section 4.8.2). Interestingly, we see very little variance in performance for different σ. This appears to be caused by a trade-off between the number of cliques and their cardinality. As σ increases, the average clique cardinality and duration decrease (see Figures 4.19 and 4.20 in Section 4.8.2) and, consequently, the number of cliques increases (see Figure 4.9). In this trade-off, the increase in the number of cliques appears to weigh slightly heavier, as runtimes appear to increase slightly for larger σ.

When compared to the performance of the algorithm introduced by Banerjee & Pal, Figure 4.16 shows that our algorithm always performs better. Importantly, the relative improvement is better at higher structural (i.e., static) density. Thus, our algorithm is more scalable, i.e., better suited to larger networks. Another interesting observation is that, while high temporal density leads to comparatively better performance at low structural density, low temporal density is preferred at high structural densities. We likely observe better relative performance for higher temporal density at low structural densities because of the slower stretching phase for Banerjee & Pal's algorithm. After all, at low structural density, the amount of time spent on the stretching phase relative to the total runtime, is larger. Consequently, our algorithm benefits more from its linear time stretching phase at the low structural densities, i.e., for sparse networks, and this benefit only increases the greater the temporal density.

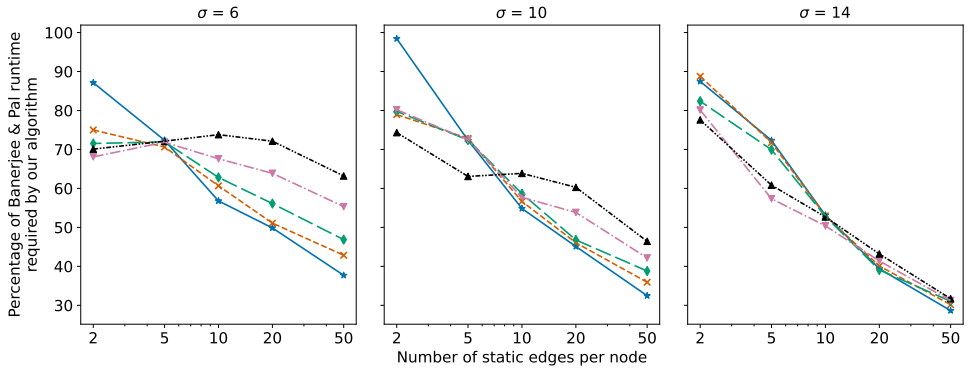
4.7 Conclusions

In this chapter, we introduced a (δ, γ)-maximal clique enumeration algorithm that improves upon the state-of-the-art two-phase approach introduced by Banerjee & Pal [8]. Furthermore, we reformulated the definition of (δ, γ)-cliques to fix two issues in the temporal domain and to follow a more natural interpretation of δ. Additionally, we extended the definition to cover weighted networks.

Our proposed algorithm improves the time complexity of the initial stretching phase such that it is linear with respect to the number of temporal edges, while also accounting for edge weighting. Although our corrected definition requires more complex considerations, during clique node set expansion, in the subsequent bulking



(a) Erdös-Rényi random graphs



(b) Barabasi-Albert random graphs

Figure 4.16: Average percentage of runtime by Banerjee & Pal's algorithm that is required by our algorithm for (δ, γ)-maximal clique enumeration, for $\delta = 10$ and for each density parameter setting for the synthetic datasets.

phase, we improved its computation time by introducing an efficient pruning method. Furthermore, by employing a custom node labeling and associated search order, we were able to reduce the space requirements and skip any duplicate search states.

Our experiments empirically demonstrated, in all but a few cases, a substantial performance improvement by our algorithm in both computational time and space requirements over existing methods. These improvement were especially clear for larger and denser networks and for smaller δ, demonstrating better scalability. Furthermore, we showed that by extending the methodology to be able to handle weighted networks, we have opened up more nuanced options for imposing minimum requirements on

clique formation. Thus, in summary, the method we introduced in this chapter is easier to interpret, scales better to larger networks, and allows one to impose more real-world restrictions on cliques through weighted networks.

Our algorithm, of course, has its limitations. Our experiments on synthetic datasets showed that, like most clique enumeration algorithms, more temporally and structurally dense networks take longer to process. This is for a large part due to there simply being more cliques to enumerate. However, the increased density also increases the number of conditions that must be satisfied in order the prune. Consequently, fewer search tree branches can be pruned and thus the workload increases. However, our experiments showed that applying more strict conditions on the cliques to be enumerated, i.e., higher weights (γ) or lower time frames (δ), greatly reduces the workload as fewer node pairs are sufficiently connected within sufficient time and therefore the practical density decreases.

Given that computational performance is tied to the number of search states processed, future improvements on this method could consider other approaches to node labeling that (further) optimize the amount of pruning that is achieved. Such methods could try to combine pivot selections along the lines of those used for static cliques [133], while also taking into account the temporal dominance. Another potential improvement can be made by parallelizing the two phases of the algorithm. After all, in the stretching phase each edge can be processed independently, while in the bulking phase processing each clique in C^s independently may at worst lead us to process cliques whose search tree branches could have been pruned.

4.8 Appendix

In this section we provide the algorithm pseudo-code in Section 4.8.1 and supplementary result figures in Section 4.8.2.

4.8.1 Algorithm pseudo code

Algorithm 1 Stretch phase wrapper function

Require: Given (weighted) temporal graph $\mathcal{G} = (V, E, \mathcal{T}, \mathcal{W}^{\mathcal{T}})$, let all_times and $all\_weights$ be dictionaries of lists such that for each edge $(t_i, u_i, v_i, w_i) \in E^{\mathcal{T}\mathcal{W}^{\mathcal{T}}}$, with o_i indicating its place in the temporally ordered sequence of edge instances of (u_i, v_i), it holds that $all\_times[(u_i, v_i)][o_i] = t_i$ and $all\_weights[(u_i, v_i)][o_i] = w_i$.
Ensure: C^s is the set of all 2-node (δ, γ)-cliques that are duration-wise maximal

```
procedure STRETCHPHASE(δ, γ)
    if δ > 0 then
        for (u, v) ∈ E do
            if sum(all_weights[(u, v)]) ≥ γ then
                STRETCH({u, v}, δ, γ, all_times[(u, v)], all_weights[(u, v)])
            end if
        end for
    end if
end procedure
```

Algorithm 2 Stretch phase function for a given link

```
10: procedure STRETCH(node_set = {u, v}, δ, γ, times, weights)
11:     bi, ei ← 0, 0
12:     current_weight ← weights[bi]
13:     while True do
14:         while current_weight < γ do                ▷ Find next set of edges with sufficient weight
15:             ei ← ei + 1
16:             if ei = len(times) then return
17:             end if
18:             while times[ei] − times[bi] ≥ δ do
19:                 current_weight ← current_weight − weight[bi]
20:                 bi ← bi + 1
21:             end while
22:             current_weight ← current_weight + weight[ei]
23:         end while
24:         tb, tbMax ← times[bi], times[ei]           ▷ Remember clique start borders

25:         done ← False
26:         while not done do
27:             old_bi, old_ei ← bi, ei
28:             while ei + 1 < len(times) and times[ei + 1] − times[bi] < δ do
29:                 ei ← ei + 1
30:                 current_weight ← current_weight + weight[ei]
31:             end while
32:             while bi + 1 ≤ ei and current_weight − weights[bi] ≤ γ do
33:                 current_weight ← current_weight − weight[bi]
34:                 bi ← bi − 1
35:             end while
36:             if ei + 1 = len(times) then
37:                 C^s.addClique(node_set, (tb, tbMax, times[bi], times[ei]))
38:                 return
39:             end if

40:             if bi = old_bi and ei = old_ei then
41:                 bt, et ← times[bi] + 1, ei
42:                 current_weight ← current_weight − weight[bi]
43:                 while current_weight < γ do
44:                     et ← et + 1
45:                     if et = len(times) then
46:                         C^s.addClique(node_set, (tb, tbMax, times[bi], times[ei]))
47:                         return
48:                     end if
49:                     if times[et] − bt ≥ δ then
50:                         C^s.addClique(node_set, (tb, tbMax, times[bi], times[ei]))
51:                         done ← True
52:                         et ← et − 1                ▷ Reduce the index as its weight was not yet added
53:                         break
54:                     end if
55:                     current_weight ← current_weight + weight[et]
56:                 end while
57:                 bi ← bi + 1
58:                 ei ← et
59:             end if
60:         end while
61:     end while
62: end procedure
```

Algorithm 3 Bulk phase wrapper and helper functions

Require: C^s is the set of all 2-node duration-wise maximal (δ, γ)-cliques, which are defined as triplets of the node set, time interval, and outer borders of the clique, i.e., $C = (X, [t_b, t_e], (tbMin, tbMax, teMin, teMax))$. Note that clique properties are accessed in this pseudo-code by using the $C.t_b$ notation. Furthermore, if E represents a set of cliques, then $E.t_b$ represents a list of all values of t_b of the cliques in E.

Ensure: R is the set of all (δ, γ)-maximal cliques

procedure BULKPHASE(δ)
  $D \leftarrow \emptyset$
  for $C \in C^s$ **do** ▷ Cliques processed in increasing order based on $\max(X)$
    if $c \notin D$ **then**
      $D \leftarrow D \cup$ BULKRECURSIVEOUTER(δ, C)
    end if
  end for
end procedure

procedure ISTEMPORALLYDOMINATED(C, C_{dom})
  if $C_{dom}.t_b \leq C.t_b$ **and** $C.t_e \leq C_{dom}.t_e$ **then**
    return $True$ ▷ Time interval of C is fully covered by C_{dom}
  end if
  return $False$
end procedure

procedure ISSPATIALGROWTHDOMINATED(C, N^e, C_{new}, E^r)
  if $\{u \in N(C.X) \setminus N^e \mid u > \max(C.X)\} = \emptyset$ **then**
    if $\max(E^r.tbMin) \leq C_{new}.teMin$ **and** $C_{new}.tbMax \leq \min(E^r.teMax)$ **then**
      return $True$ ▷ All potential extensions to $C.X$ are covered by N^e
    end if
  end if
  return $False$
end procedure

procedure ISTEMPORALGROWTHDOMINATED(C, E^r, N^r, C^r, C^p)
  if $\max(E^r.tbMin) \leq C.tbMin$ **and** $C.teMax \leq \min(E^r.teMax)$ **then**
    return $True$ ▷ Minimum potential growth E^r exceeds potential growth of C
  end if

  if $\min(C^r.t_b) < \max(E^r.tbMin)$ **or** $\min(E^r.teMax) < \max(C^r.t_e)$ **then**
    return $False$ ▷ Original time intervals exceed minimum potential growth of E^r
  end if

  for $C_i \in \{C_j \in C^s \mid C_j.X \subseteq N^r\}$ **do** ▷ Check edges between not yet added neighbors
    if $C_i.teMax \geq \min(C^p.tbMax)$ **and** $C_i.t_b < \max(E^r.tbMin)$ **then**
      return $False$ ▷ Sufficient overlap with C, yet before potential growth of E^r
    end if
    if $C_i.tbMin \leq \max(C^p.teMin)$ **and** $C_i.t_e > \min(E^r.teMax)$ **then**
      return $False$ ▷ Sufficient overlap with C, yet after potential growth of E^r
    end if
  end for
  return $True$ ▷ No actual growth beyond potential of E^r possible for C, safe to cut
end procedure

Algorithm 4 Bulk phase outer recursion function

procedure BULKRECURSIVEOUTER(δ, C)
  $E, E^e, N^e, C^e, C^a \leftarrow \emptyset, \emptyset, \emptyset, \emptyset, \emptyset$
  for $n \in$ SHAREDNEIGHBORS(C) **do** ▷ Process in increasing node label order
    $\{u, v\} \leftarrow C.X$
    $O_{uv} \leftarrow$ OUTEROVERLAPCLIQUE(C, u, v, n, δ)
    for $(\{C_u, C_v\}, C_{new}) \in O_{uv}$ **do**
      $E \leftarrow E \cup C_{new}$
      if $n > \max(C.X)$ **then** ▷ Limit branch cut logic to actual potential extensions
        $E^e \leftarrow E^e \cup C_{new}$
        $N^e \leftarrow N^e \cup n$
        $C^e \leftarrow C^e \cup \{(C_u, C_{new}), (C_v, C_{new})\}$
      end if
    end for
    $C^a \leftarrow C^a \cup \{C_k \in C^s \mid C_k.X \in \{\{u, n\}, \{v, n\}\} \wedge$
      $C_k.tbMin \leq \max(\{C_i.teMin \mid (C_i, C_x) \in C^e\}) \wedge$
      $C_k.teMax \geq \min(\{C_i.tbMax \mid (C_i, C_x) \in C^e\})\}$
  end for

  $D \leftarrow \emptyset$
  for $(C_n, C_{new}) \in C^e$ **do** ▷ Determine which $C_n \in C^s$ can be cut
    if ISTEMPORALLYDOMINATED(C_n, C_{new}) **then**
      $E^r \leftarrow \{C_k \in E^e \mid \max(C_k.X) \geq \max(C_n.X)\}$
      if ISSPATIALGROWTHDOMINATED(C_n, N^e, C_{new}, E^r) **then**
        $N^r \leftarrow \{u \in N^e \mid u \geq \max(C_n.X)\}$
        $C^r \leftarrow \{C_k \in C^a \mid \max(C_k.X) \geq \max(C_n.X)\}$
        $C^p \leftarrow \{C_i \mid (C_i, C_x) \in C^e \wedge \max(C_i.X) \leq \max(C_n.X)\}$
        if ISTEMPORALGROWTHDOMINATED(C_n, E^r, N^r, C^r, C^p) **then**
          $D \leftarrow D \cup C_n$ ▷ The branch defined by clique C_n may be cut
        end if
      end if
    end if
  end for

  $MAXIMAL, D_{local} \leftarrow True, \emptyset$
  for $C_{new} \in E$ **do** ▷ Extension cliques processed in increasing order based on $\max(X)$
    if ISTEMPORALLYDOMINATED(C, C_{new}) **then**
      $MAXIMAL \leftarrow False$
    end if
    if $C_{new} \notin D_{local}$ **and** $\max(C_{new}.X) > \max(C.X)$ **then**
      $D_{local} \leftarrow D_{local} \cup$ BULKRECURSIVEINNER(δ, C_{new}, E)
    end if
  end for

  if $MAXIMAL$ **then**
    $R \leftarrow R \cup C$
  end if
  return D
end procedure

Algorithm 5 Bulk phase outer recursion helper function, for determining node extensions from 2-node cliques with valid time interval overlap

procedure OUTEROVERLAPCLIQUE(C, u, v, n, δ)
  $O_{uv} \leftarrow \emptyset$
  for $C_u \in \{C_k \in C^s \mid C_k.X = \{u, n\} \wedge C_k.tbMin \leq C.teMin \wedge C_k.teMax \geq C.tbMax\}$ **do**
    $tbMin^{new} \leftarrow \max(C.tbMin, C_u.tbMin)$ ▷ Shrink potential growth
    $teMax^{new} \leftarrow \min(C.teMax, C_u.teMax)$
    $tbMax^{new} \leftarrow tbMin^{new} + \delta - 1$
    $teMin^{new} \leftarrow teMax^{new} - \delta + 1$
    for $C_v \in \{C_k \in C^s \mid C_k.X = \{v, n\} \wedge C_k.tbMin \leq teMin^{new} \wedge C_k.teMax \geq tbMax^{new}\}$ **do**
      $tbMin^{new} \leftarrow \max(tbMin^{new}, C_v.tbMin)$ ▷ Shrink potential growth
      $teMax^{new} \leftarrow \min(teMax^{new}, C_v.teMax)$
      $tbMax^{new} \leftarrow tbMin^{new} + \delta - 1$
      $teMin^{new} \leftarrow teMax^{new} - \delta + 1$

      $t_b^{uv}, t_b^{un}, t_b^{vn} \leftarrow C.t_b, C_u.t_b, C_v.t_b$
      if $t_b^{uv} < tbMin^{new}$ **then** ▷ Time interval C outside outer border
        $t_b^{uv} \leftarrow \min(\{t \mid (t, u, v) \in E^{\mathcal{T}} \wedge t \geq tbMin^{new}\})$
      end if
      if $t_b^{un} < tbMin^{new}$ **then** ▷ Time interval C_u outside outer border
        $t_b^{un} \leftarrow \min(\{t \mid (t, u, n) \in E^{\mathcal{T}} \wedge t \geq tbMin^{new}\})$
      end if
      if $t_b^{vn} < tbMin^{new}$ **then** ▷ Time interval C_v outside outer border
        $t_b^{vn} \leftarrow \min(\{t \mid (t, n, v) \in E^{\mathcal{T}} \wedge t \geq tbMin^{new}\})$
      end if
      $t_b^{new} \leftarrow \min(t_b^{uv}, t_b^{un}, t_b^{vn})$ ▷ New t_b is first occurrence after $tbMin^{new}$

      $t_e^{uv}, t_e^{un}, t_e^{vn} \leftarrow C.t_b, C_u.t_b, C_v.t_b$
      if $t_e^{uv} > teMax^{new}$ **then** ▷ Time interval C outside outer border
        $t_e^{uv} \leftarrow \max(\{t \mid (t, u, v) \in E^{\mathcal{T}} \wedge t \leq teMax^{new}\})$
      end if
      if $t_e^{un} > teMax^{new}$ **then** ▷ Time interval C_u outside outer border
        $t_e^{un} \leftarrow \max(\{t \mid (t, u, n) \in E^{\mathcal{T}} \wedge t \leq teMax^{new}\})$
      end if
      if $t_e^{vn} > teMax^{new}$ **then** ▷ Time interval C_v outside outer border
        $t_e^{vn} \leftarrow \max(\{t \mid (t, n, v) \in E^{\mathcal{T}} \wedge t \leq teMax^{new}\})$
      end if
      $t_e^{new} \leftarrow \max(t_e^{uv}, t_e^{un}, t_e^{vn})$ ▷ New t_e is last occurrence before $teMax^{new}$

      $C_o \leftarrow Clique(\{u, v, n\}, [t_b^{new}, t_e^{new}], (tbMin^{new}, tbMax^{new}, teMin^{new}, teMax^{new})))$
      $O_{uv} \leftarrow O_{uv} \cup (\{C_u, C_v\}, C_o)$
    end for
  end for
  return O_{uv}
end procedure

Algorithm 6 Bulk phase inner recursion function

185: **procedure** BULKRECURSIVEINNER(δ, C, E_{prev})
186: $E, E^e, N^e, C^e, C^a \leftarrow \emptyset, \emptyset, \emptyset, \emptyset, \emptyset$
187: **for** $C_p \in E_{prev} \setminus \{C\}$ **do** ▷ Process in increasing node label order
188: $O_e \leftarrow$ INNEROVERLAPCLIQUE(C, C_p, δ)
189: **for** $C_{new} \in O_e$ **do**
190: $E \leftarrow E \cup C_{new}$
191: **if** $\max(C_p.X) > \max(C.X)$ **then**
192: $E^e \leftarrow E^e \cup C_{new}$
193: $N^e \leftarrow N^e \cup \max(C_p.X)$
194: $C^e \leftarrow C^e \cup (C_p, C_{new})$
195: **end if**
196: **end for**
197: $C^a \leftarrow C^a \cup \{C_k \in E_{prev} \mid \max(C_p.X) = \max(C_k.X) \wedge$
$C_k.tbMin \leq \max(\{C_i.teMin \mid (C_i, C_x) \in C^e\}) \wedge$
$C_k.teMax \geq \min(\{C_i.tbMax \mid (C_i, C_x) \in C^e\})\}$
198: **end for**

199: $D \leftarrow \emptyset$
200: **for** $(C_n, C_{new}) \in C^e$ **do** ▷ Determine which $C_n \in E_{prev}$ can be cut
201: **if** ISTEMPORALLYDOMINATED(C_n, C_{new}) **then**
202: $E^r \leftarrow \{C_k \in E^e \mid \max(C_k.X) \geq \max(C_n.X)\}$
203: **if** ISSPATIALGROWTHDOMINATED(C_n, N^e, C_{new}, E^r) **then**
204: $N^r \leftarrow \{u \in N^e \mid u \geq \max(C_n.X)\}$
205: $C^r \leftarrow \{C_k \in C^a \mid \max(C_k.X) \geq \max(C_n.X)\}$
206: $C^p \leftarrow \{C_i \mid (C_i, C_x) \in C^e \wedge \max(C_i.X) \leq \max(C_n.X)\}$
207: **if** ISTEMPORALGROWTHDOMINATED(C_n, E^r, N^r, C^r, C^p) **then**
208: $D \leftarrow D \cup C_n$ ▷ The branch defined by clique C_n may be cut
209: **end if**
210: **end if**
211: **end if**
212: **end for**

213: $MAXIMAL, D_{local} \leftarrow True, \emptyset$
214: **for** $C_{new} \in E$ **do** ▷ Extension cliques processed in increasing order based on $\max(X)$
215: **if** ISTEMPORALLYDOMINATED(C, C_{new}) **then**
216: $MAXIMAL \leftarrow False$
217: **end if**
218: **if** $C_{new} \notin D_{local}$ **and** $\max(C_{new}.X) > \max(C.X)$ **then**
219: $D_{local} \leftarrow D_{local} \cup$ BULKRECURSIVEINNER(δ, C_{new}, E)
220: **end if**
221: **end for**

222: **if** $MAXIMAL$ **then**
223: $R \leftarrow R \cup C$
224: **end if**
225: **return** D
226: **end procedure**

Algorithm 7 Bulk phase inner recursion helper function, for determining node extensions to $C.X$ with $\max(C_p.X)$ with valid time interval overlap

procedure INNEROVERLAPCLIQUE(C, C_p, δ)
  $O_e \leftarrow \emptyset$
  $X_{new} \leftarrow C.X \cup C_p.X$
  if $C_p.tbMin \leq C.teMin$ **and** $C_p.tbMax \geq C.tbMax$ **then**
    $tbMin^{new} \leftarrow \max(C.tbMin, C_p.tbMin)$ ▷ Shrink potential growth
    $teMax^{new} \leftarrow \min(C.teMax, C_p.teMax)$
    $tbMax^{new} \leftarrow tbMin^{new} + \delta - 1$
    $teMin^{new} \leftarrow teMax^{new} - \delta + 1$

    $a, b \leftarrow \max(C.X), \max(C_p.X)$
    for $C_e \in \{C_k \in C^s \mid C_k.X = \{a, b\} \wedge C_k.tbMin \leq teMin^{new} \wedge C_k.teMax \geq tbMax^{new}\}$ **do**
      $tbMin^{new} \leftarrow \max(tbMin^{new}, C_e.tbMin)$ ▷ Shrink potential growth
      $teMax^{new} \leftarrow \min(teMax^{new}, C_e.teMax)$
      $tbMax^{new} \leftarrow tbMin^{new} + \delta - 1$
      $teMin^{new} \leftarrow teMax^{new} - \delta + 1$

      $t_b^{uv}, t_b^{un}, t_b^{vn} \leftarrow C.t_b, C_u.t_b, C_v.t_b$
      if $t_b^{uv} < tbMin^{new}$ **then**
        $t_b^{uv} \leftarrow \min(\{t \mid (t, u, v) \in E^{\mathcal{T}} \wedge u, v \in C.X \wedge t \geq tbMin^{new}\})$
      end if
      if $t_b^{un} < tbMin^{new}$ **then**
        $t_b^{un} \leftarrow \min(\{t \mid (t, u, v) \in E^{\mathcal{T}} \wedge u, v \in C_p.X \setminus C.X \wedge t \geq tbMin^{new}\})$
      end if
      if $t_b^{vn} < tbMin^{new}$ **then**
        $t_b^{vn} \leftarrow \min(\{t \mid (t, a, b) \in E^{\mathcal{T}} \wedge t \geq tbMin^{new}\})$
      end if
      $t_b^{new} \leftarrow \min(t_b^{uv}, t_b^{un}, t_b^{vn})$ ▷ New t_b is first occurrence after $tbMin^{new}$

      $t_e^{uv}, t_e^{un}, t_e^{vn} \leftarrow C.t_b, C_u.t_b, C_v.t_b$
      if $t_e^{uv} > teMax^{new}$ **then**
        $t_e^{uv} \leftarrow \max(\{t \mid (t, u, v) \in E^{\mathcal{T}} \wedge u, v \in C.X \wedge t \leq teMax^{new}\})$
      end if
      if $t_e^{un} > teMax^{new}$ **then**
        $t_e^{un} \leftarrow \max(\{t \mid (t, u, v) \in E^{\mathcal{T}} \wedge u, v \in C_p.X \setminus C.X \wedge t \leq teMax^{new}\})$
      end if
      if $t_e^{vn} > teMax^{new}$ **then**
        $t_e^{vn} \leftarrow \max(\{t \mid (t, a, b) \in E^{\mathcal{T}} \wedge t \leq teMax^{new}\})$
      end if
      $t_e^{new} \leftarrow \max(t_e^{uv}, t_e^{un}, t_e^{vn})$ ▷ New t_e is last occurrence before $teMax^{new}$

      $C_o \leftarrow Clique(X_{new}, [t_b^{new}, t_e^{new}], (tbMin^{new}, tbMax^{new}, teMin^{new}, teMax^{new}))$
      $O_e \leftarrow O_e \cup C_o$
    end for
  end if
  return O_e
end procedure

Algorithm 8 Synthetic temporal network generating function

Require: Let G by an Erdös-Rényi or Barabasi-Albert random graph with edge set E_G, let $N_G(u)$ indicate the set of nodes neighboring node u in G, let $R(X)$ indicate a random choice from the elements in set X, and let $E_H(u, v)$ indicate the set of temporal edges in H for static edge (u, v).

Ensure: H is a random temporal graph whose underlying static network is G

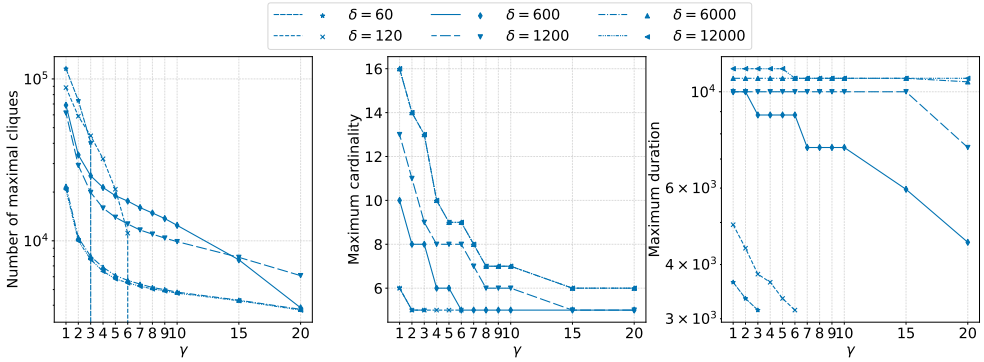
procedure GENERATERANDOMTEMPORALGRAPH($G, \lambda_{in}, \sigma_{in}$)
  $E_H \leftarrow \emptyset$
  for $(u, v) \in E_G$ **do**
    $T_N \leftarrow \emptyset$ ▷ The set of known timestamps from neighboring edges
    for $w \in N_G(u) \setminus \{v\}$ **do**
      $T_N \leftarrow T_N \cup \{(t, u, w) \in E_H\}$
    end for
    for $w \in N_G(v) \setminus \{u\}$ **do**
      $T_N \leftarrow T_N \cup \{(t, w, v) \in E_H\}$
    end for

    if $T_N = \emptyset$ **then**
      $t_{new} \leftarrow$ random integer from range $[100, 200]$
    else
      $t_{new} \leftarrow RandomDistribution(\mu = R(T_N), \sigma = \frac{1}{2}\sigma_{in})$
    end if
    $E_H \leftarrow E_H \cup (t_{new}, u, v)$

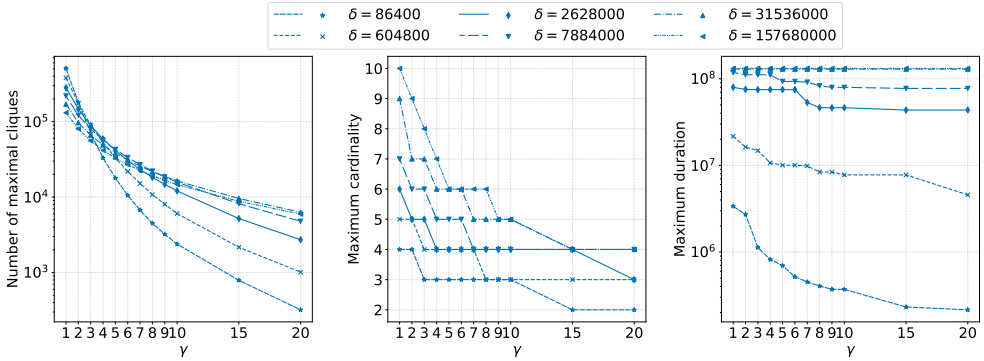
    $ts \leftarrow PoissonDistribution(\lambda = \lambda_{in})$
    $i \leftarrow 0$
    while $i < ts$ **do**
      $t_{new} \leftarrow RandomDistribution(\mu = R(E_H(u, v)), \sigma = \sigma_{in})$
      if $T_N \neq \emptyset$ **then**
        $t_{new} \leftarrow R(t_{new}, RandomDistribution(\mu = R(T_N), \sigma = \frac{1}{2}\sigma_{in}))$
      end if

      if $(t_{new}, u, v) \notin E_H$ **then**
        $E_H \leftarrow E_H \cup (t_{new}, u, v)$
        $i \leftarrow i + 1$
      end if
    end while
  end for
end procedure

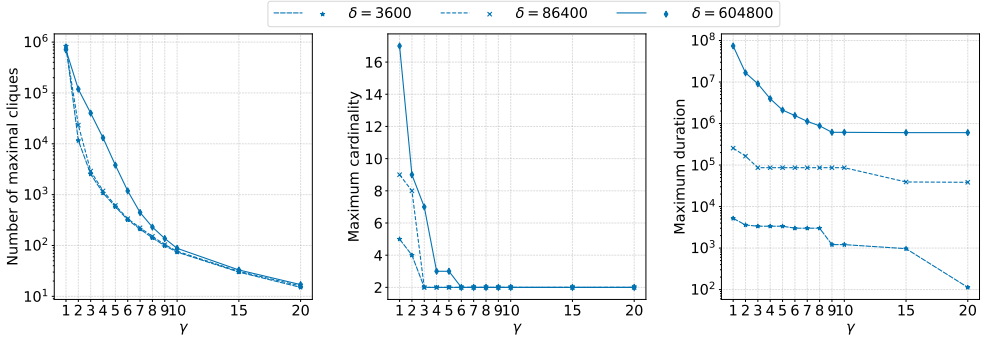
4.8.2 Supplementary result figures



(a) Infectious I

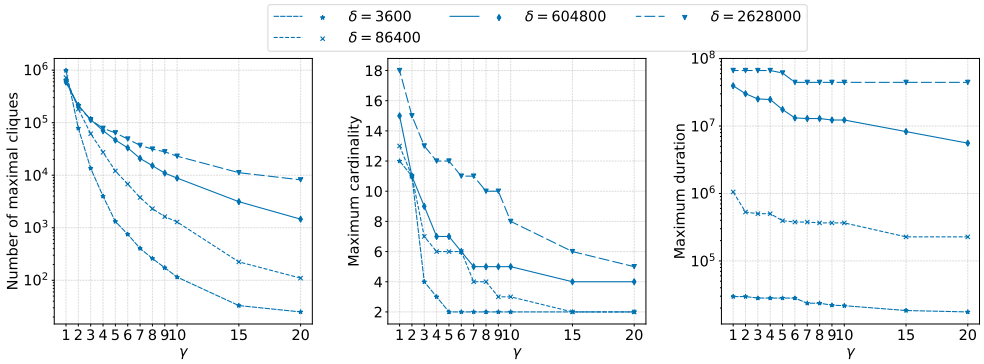


(b) Facebook-wosn-wall

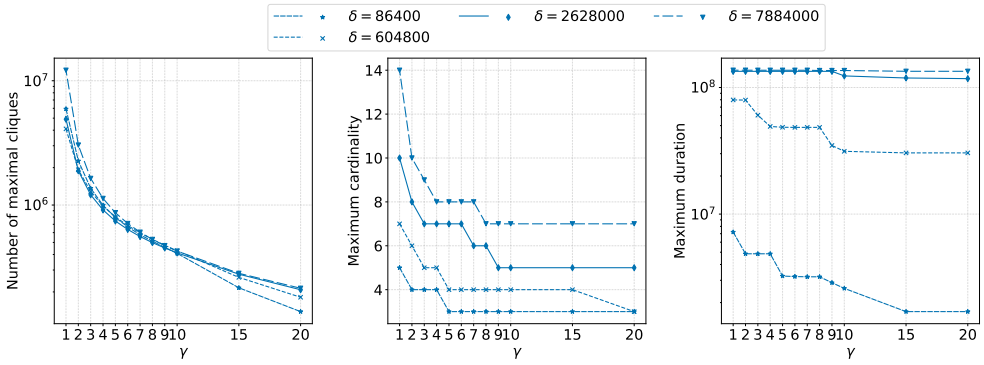


(c) Reddit Hyperlinks

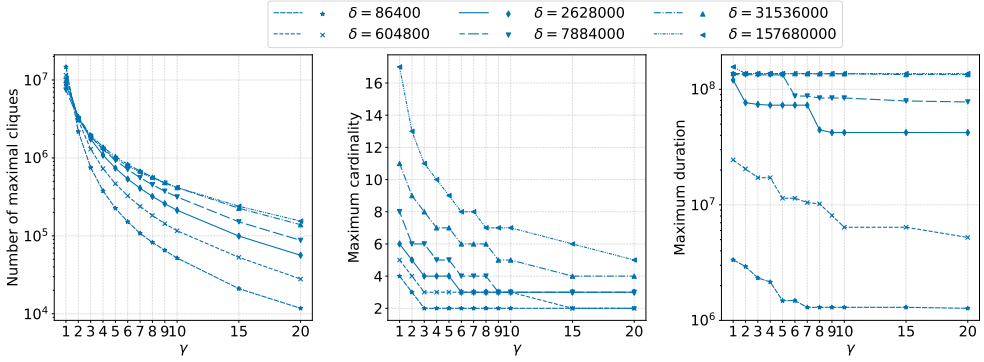
Figure 4.17: Properties of the set of enumerated cliques for a range of γ values. The left panels show the number of (δ, γ)-maximal cliques found and the middle and right panels show, respectively, their maximum cardinality and maximum duration.



(a) Enron



(b) Last.fm bands



(c) Last.fm songs

Figure 4.18: Properties of the set of enumerated cliques for a range of γ values. The left panels show the number of (δ, γ)-maximal cliques found and the middle and right panels show, respectively, their maximum cardinality and maximum duration.

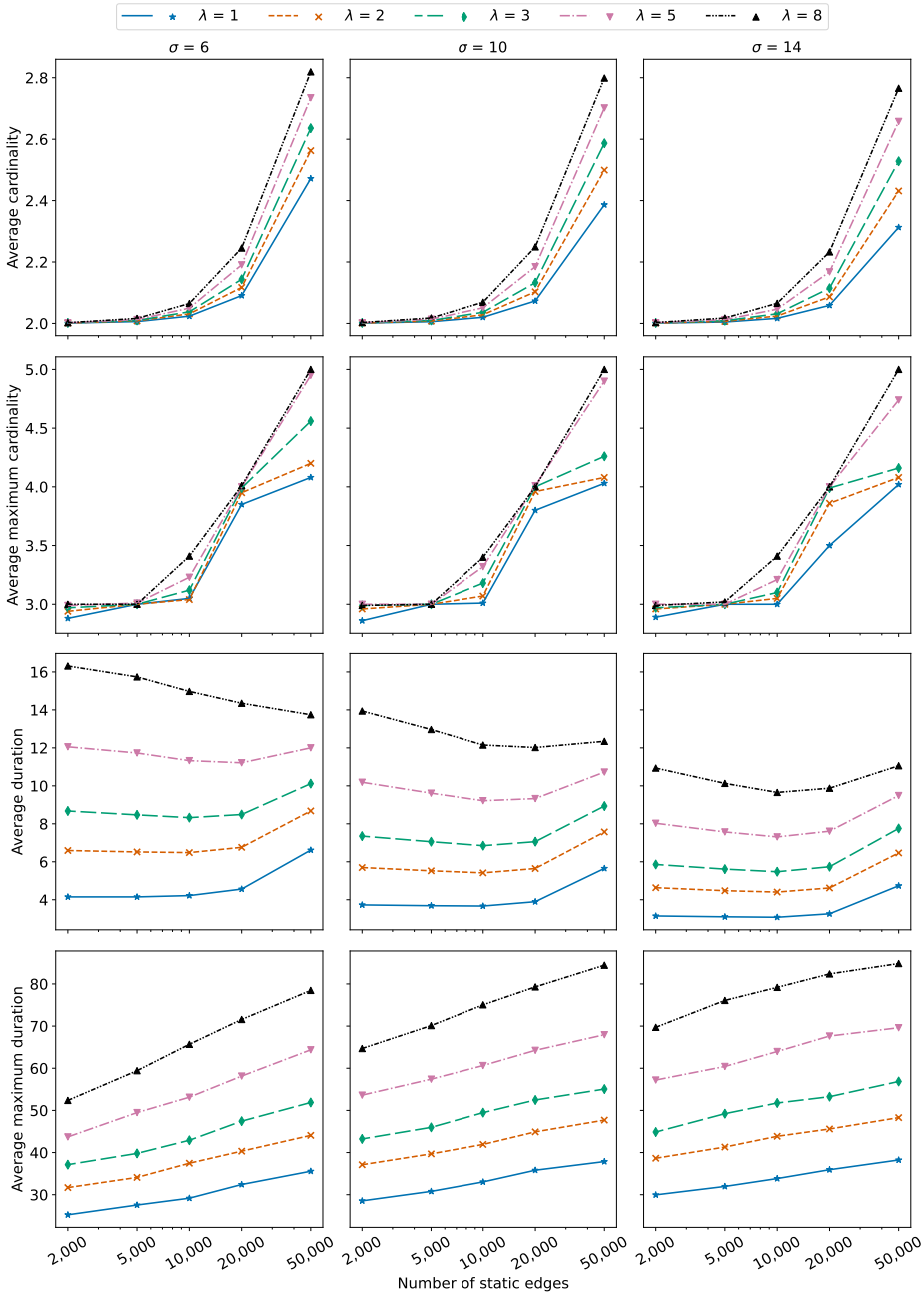


Figure 4.19: Average clique properties (y-axis) over all 100 synthetic networks for each of the structural and temporal density parameter settings for Erdös-Rényi graphs.

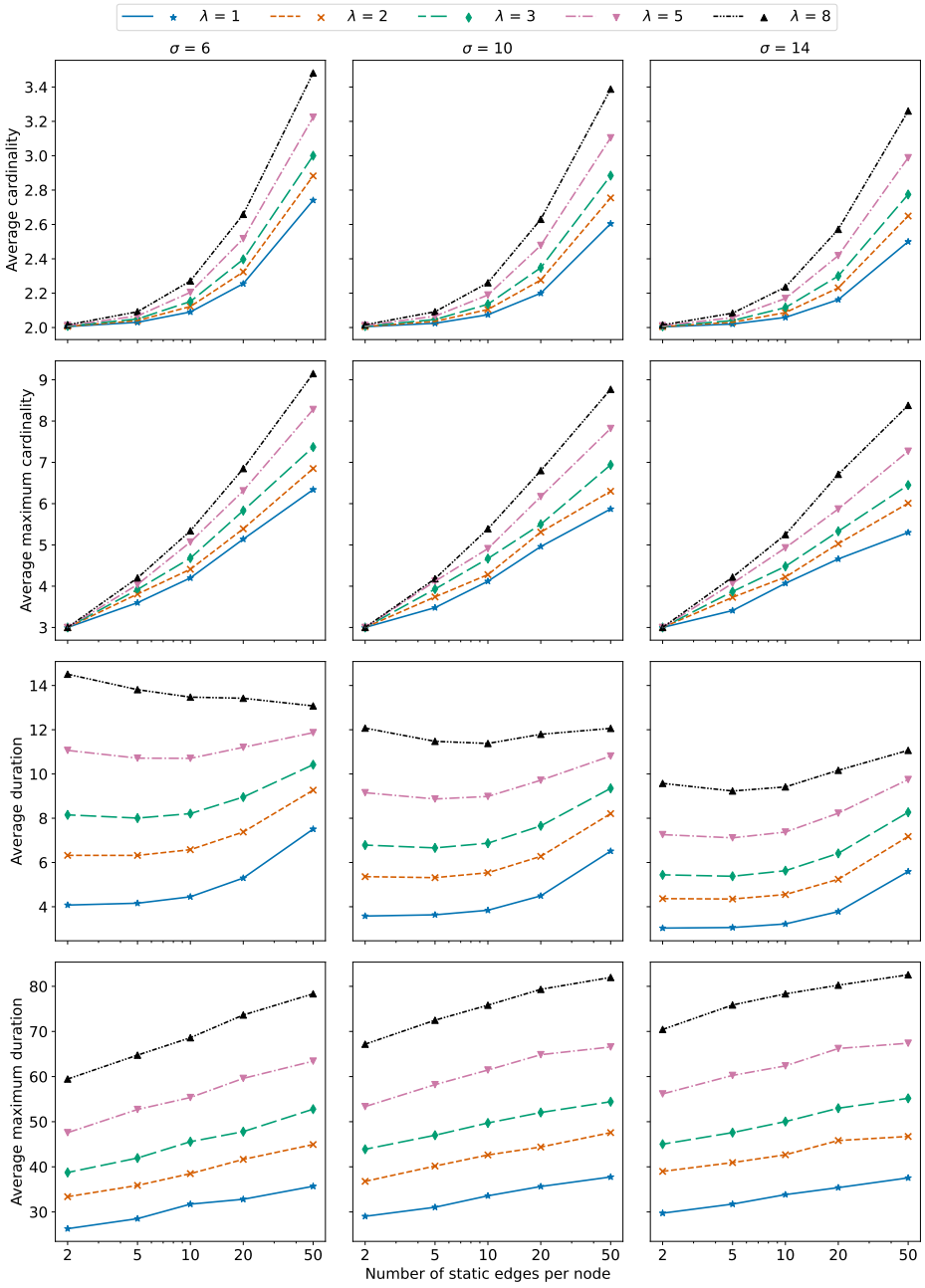


Figure 4.20: Average clique properties (y-axis) over all 100 synthetic networks for each of the structural and temporal density parameter settings for Barabasi-Albert graphs.