Diversified Temporal Subgraph Pattern Mining

Yi Yang[1,2] Da Yan[1] Huanhuan Wu[1] James Cheng[1] Shuigeng Zhou[2] John Lui[1]
[1]Department of Computer Science and Engineering
[1]The Chinese University of Hong Kong
[2]School of Computer Science, Fudan University
[1]{yyang,yanda,hhwu,jcheng,cslui}@cse.cuhk.edu.hk
[2]{yyang1,sgzhou}@fudan.edu.cn

ABSTRACT

Many graphs in real-world applications, such as telecommunications networks, social-interaction graphs and co-authorship graphs, contain temporal information. However, existing graph mining algorithms fail to exploit these temporal information and the resulting subgraph patterns do not contain any temporal attribute. In this paper, we study the problem of mining a set of diversified temporal subgraph patterns from a temporal graph, where each subgraph is associated with the time interval that the pattern spans. This problem motivates important applications such as finding social trends in social networks, or detecting temporal hotspots in telecommunications networks. We propose a divide-and-conquer algorithm along with effective pruning techniques, and our approach runs 2 to 3 orders of magnitude faster than a baseline algorithm and obtains high-quality temporal subgraph patterns in real temporal graphs.

1. INTRODUCTION

Many graphs in real world applications contain temporal information. For example, telecommunication companies record huge amounts of phone call and SMS records every day, where each phone call or SMS record contains attributes about the sender, the recipient, and the time when the phone call was made or the SMS was transmitted. As another example, online social networking companies keep logs about the interactions between users and the time when each interaction occurred. However, most existing graph mining algorithms do not consider temporal information in a graph, and thus fail to exploit those temporal attributes for detecting important temporal patterns such as social trends, temporal communication hotspots, and evolving social structures.

In this paper, we study the mining of subgraph structures with temporal information. Specifically, we define the concept of *temporal subgraph pattern*, which consists of a set of vertices S and a time interval $I = [t_s, t_e]$, indicating that all the vertices in S closely interact with each other during the period of time from t_s to t_e. Mining a set of diversified temporal subgraph patterns from a temporal graph motivates numerous new applications, and we describe a few of them as follows.

Evolving Social Groups. In an online social network, people join and leave social groups from time to time. For example, during the period of FIFA World Cup, people who are interested in football will actively discuss events on FIFA World Cup in an online forum. When the World Cup competition ends, people may change to interact more actively with another group of people, e.g., to share things happened in another event, or to discuss projects with colleagues/classmates. Such interesting social groups can be detected by mining temporal subgraph patterns from a temporal graph constructed from the interaction records of an online forum. Moreover, the detected social activities contain temporal information, which can help social media companies to recommend new applications and products to their users before a similar event happens next time.

Temporal Hotspots. In a telecommunication company, each connection between two users is recorded along with the time of the connection. By mining temporal subgraph patterns from a temporal graph extracted from user connections, the telecommunication company can detect communication hotspots in different time periods, and allocate more resources to hotspot regions in their peak time periods to improve services. The company may also use the information of hotspot for marketing or promotion activities.

In fact, temporal clustering has been extensively studied in the spatial setting [4, 8, 10, 15], which detects clusters of people or animals that move together for a reasonably long period of time. This work studies the problem in the context of a temporal graph, where the distance between two vertices are evaluated on a graph rather than in a geo-spatial setting.

Since it is important to find social groups (or subgraph patterns) where people closely interact with each other, many definitions of dense subgraph patterns have been proposed, such as k-core [3], k-truss [26], γ-dense subgraph [7, 11] and γ-quasi-clique [1]. Among them, k-core and k-truss can be efficiently computed, but for a specific value of k, there is only one k-core and k-truss subgraph for a graph. Thus, they only provide us a global view of the dense parts of a graph, rather than individual dense groups. A better definition is γ-dense subgraph or γ-quasi-clique, which qualifies a subset of vertices to form a social group if each vertex interacts with most of the other vertices in the subset.

This paper follows the definition of γ-quasi-clique. Since computing γ-quasi-clique is NP-hard [20], existing solutions are either fast approximation algorithms [29], or exact algorithms with heuristic pruning rules [16]. Moreover, computing γ-quasi-cliques becomes much harder in the context of a temporal graph, as the time information makes the problem much more challenging.

The main contributions of this paper are summarized as follows:

- To our knowledge, this is the first work that studies the problem of mining dense subgraph patterns in a temporal graph. We formally define the problem of mining *top k diversified temporal subgraph patterns*, and propose efficient algo-

KDD '16 San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

rithms for the mining task.

- We design several effective pruning rules to remove vertices and time periods that do not belong to any temporal subgraph patterns, which allow us to terminate a useless search as early as possible.
- We propose a heuristic search strategy. Based on the search strategy, a quick search algorithm and a complete search algorithm are proposed. We then compare them with a baseline approach on several real datasets. The experimental results demonstrate that our algorithms obtain high-quality mining results, while the quick search algorithm is 2 to 3 orders of magnitude faster than the baseline algorithm.

The rest of this paper is organized as follows. Section 2 defines some notations and the problem. Section 3 introduces the baseline algorithm. Section 4 proposes the framework of our solution, including the mining algorithms. Section 5 proposes some pruning rules for mining dense subgraph patterns in temporal graphs. Section 6 proposes some optimization techniques for our mining algorithms. Section 7 studies the performance of our mining algorithms on real datasets. Section 8 introduces the related works. Finally, Section 9 concludes the paper.

2. PROBLEM DEFINITION

Temporal Graph. In this paper, we consider an undirected temporal graph without self loops. A temporal graph $\mathbb{G} = (V, \mathbb{E})$ consists of a vertex set V and a temporal edge set $\mathbb{E}$. Each temporal edge $e \in \mathbb{E}$ has the form (u, v, I), indicating that there is an undirected edge between vertices u and v during the time interval I. Here, $I = [t_s, t_e]$ indicates that e appears in $\mathbb{G}$ during the time period $[t_s, t_e]$, and we denote the length of I by $|I| = t_e - t_s$.

A time interval I consists of a set of discrete time points or time snapshots (e.g., I can be a minute consisting of 60 seconds). Given a time snapshot t, we define the edge set of $\mathbb{G}$ at time t as $\mathbb{E}(t) = \{(u, v) \mid \exists(u, v, I) \in \mathbb{E} : t \in I\}$, and define the *snapshot* graph of $\mathbb{G}$ at time t as $\mathbb{G}(t) = (V, \mathbb{E}(t))$. Since the degree of a vertex v in a temporal graph $\mathbb{G}$ changes from time to time, we denote the degree of v at time t by $d_v(t)$. We can view $\mathbb{G}(t)$, $\mathbb{E}(t)$ and $d_v(t)$ as functions of t, which show how they change over time.

Given a temporal graph $\mathbb{G} = (V, \mathbb{E})$, a vertex subset $V' \subseteq V$, and a time interval I', we define the temporal subgraph of $\mathbb{G}$ induced by V' and I' as a temporal graph $\mathbb{G}' = (V', \mathbb{E}', I')$, such that for any time snapshot $t \in I'$, $\mathbb{G}'(t)$ is the subgraph of $\mathbb{G}(t)$ induced by V'.

Dense Temporal Graph. We define dense temporal graphs based on the definition of γ-quasi-clique in a non-temporal graph [1], where $0 \leq \gamma \leq 1$ is a user-defined density parameter:

Given a non-temporal graph $G = (V, E)$, if $d_v \geq \gamma \cdot (|V| - 1)$ holds for all $v \in V$, then G is a γ-*quasi-clique*. As a special case, when $\gamma = 1$, G is a clique.

Now we need to incorporate temporal information into the above definition to define dense temporal graphs as follows.

Given a temporal graph $\mathbb{G} = (V, \mathbb{E})$ and a parameter γ, we say that $\mathbb{G}$ is a γ-*quasi-clique during the time interval I* iff

$$d_v(t) \geq \gamma \cdot (|V| - 1)$$

holds for all $v \in V$ and $t \in I$.

A temporal subgraph $\mathbb{G}' = (V', \mathbb{E}', I')$ is considered γ-*dense* iff $\mathbb{G}'$ is a γ-quasi-clique during the time interval I'.

Temporal Coverage. Given a temporal graph $\mathbb{G}$ and its temporal subgraph $\mathbb{G}' = (V', \mathbb{E}', I')$, we define the *coverage set* of $\mathbb{G}'$ on $\mathbb{G}$ as $C(\mathbb{G}') = \{(v, t) \mid v \in V',\ t \in I'\}$, whose size is $|C(\mathbb{G}')| = |V'| \cdot |I'|$. Consider a set of temporal subgraphs of $\mathbb{G}$, denoted by $\mathcal{G} = \{\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_k\}$, where $\mathbb{G}_i$ is induced by a vertex set V_i and a time interval I_i from $\mathbb{G}$. The *coverage set* of $\mathcal{G}$ is the union of the coverage sets of individual subgraphs:

$$C(\mathcal{G}) = \bigcup_{i=1}^{k} C(\mathbb{G}_i) = \{(v, t) | \exists i, \text{ s.t. } v \in V_i, t \in I_i\},$$

and its size is denoted by $|C(\mathcal{G})|$. Although a pair (v, t) may appear in different coverage sets, i.e., $(v, t) \in C(\mathbb{G}_i), C(\mathbb{G}_j)$ with $i \neq j$, it is counted only once when calculating $|C(\mathcal{G})|$. Apparently, $|C(\mathcal{G})|$ is larger if (1) $|V_i|$, $|I_i|$ are larger, and (2) V_i, I_i are different from each other for $i = 1, 2, ..., k$.

The Problem. In this paper, we study how to find k *dense temporal subgraphs* in a temporal graph, such that the k subgraphs are *large and diversified (i.e., maximizing the coverage)*. Formally, given a temporal graph $\mathbb{G}$ and four parameters γ, k, σ and τ, we want to find a set of γ-dense subgraphs $\mathcal{G} = \{\mathbb{G}_1, ..., \mathbb{G}_k\}$ induced by V_i and I_i $(i = 1, 2, ..., k)$ from $\mathbb{G}$, such that (1)$|V_i| \geq \sigma$, (2) $|I_i| \geq \tau$, and (3) $|C(\mathcal{G})|$ is maximized. We call $\mathcal{G} = \{\mathbb{G}_1, ..., \mathbb{G}_k\}$ the top k *diversified temporal subgraph patterns* in $\mathbb{G}$.

EXAMPLE 1. *Assume* $\gamma = 0.8, k = 2, \sigma = 3, \tau = 3$, *and a temporal graph* $\mathbb{G}$ *shown in Figure 1. There are two diversified temporal subgraph patterns,* $\mathbb{G}_1 = (V_1, \mathbb{E}_1, I_1)$ *and* $\mathbb{G}_2 = (V_2, \mathbb{E}_2, I_2)$, *of* $\mathbb{G}$, *where* $V_1 = \{a, b, c\}, I_1 = [0, 3]$ *and* $V_2 = \{c, d, e\}, I_2 = [2, 5]$, *such that (1)* $|V_1| = |V_2| = 3 \geq \sigma$, *(2)* $|I_1| = |I_2| = 3 \geq \tau$, *and (3)* $|C(\mathcal{G})|$ *is maximized with* $\mathcal{G} = \{\mathbb{G}_1, \mathbb{G}_2\}$. *Since* $C(\mathbb{G}_1), C(\mathbb{G}_2)$ *overlap on vertex c at time* $t \in [2, 3]$, *the size of the coverage is* $|C(\mathcal{G})| = 3 \cdot 3 + 3 \cdot 3 - 1 = 17$, *which is shown in Figure 2.*

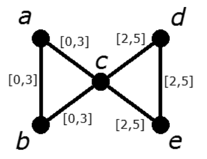


Figure 1: A temporal graph

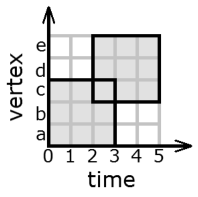


Figure 2: The coverage set

3. THE BASELINE ALGORITHM

A straightforward solution to our problem is to list all the qualified subgraph patterns, and then find k of them that maximize the coverage.

Search Space. Given a temporal graph $\mathbb{G}$ and a parameter γ, a naive solution is to check every subgraph induced by any $V' \subseteq V$ and any $I' \subseteq I$ from $\mathbb{G}$. To do so, we consider the set of discrete time points $T = \{t_1, t_2, \ldots\}$, where $t_1, t_2, \ldots$ are the time points when an edge appears or disappears. If the subgraph induced by V' is γ-quasi-clique at $t_i, t_{i+1}, \ldots, t_j$, then it is γ-dense throughout time interval $[t_i, t_j]$. Thus, it suffices to examine every non-temporal subgraph $\mathbb{G}(t)$ induced by any $V' \subseteq V$ and any $t \in T$. Therefore, the search space for listing qualified patterns is $2^{|V|} \cdot |T|$.

Pruning during Subgraph Listing. Some pruning rules have been proposed to reduce the cost of listing quasi-cliques in a non-temporal graph [16]. These pruning rules check whether a subgraph can be further grown to form a larger qualified pattern. In the algorithm of [16], a mining task consists of a set S of *selected vertices* and a set C of *candidate vertices*. It then selects a vertex $v_1 \in C$, and generates two mining tasks: (1) moving v_1 from C to S, and (2) simply removing v_1 from C. These two tasks refer to two cases, i.e., a candidate vertex v_1 is selected into (resp. excluded from) a pattern. Each task can be further expanded by checking another vertex $v_2 \in C$ in a similar manner. In this way, a mining task is recursively processed, and meanwhile, the pruning rules may terminate a task from further expansion.

Maximizing the Coverage. Selecting k subgraph patterns from all the qualified patterns can be regarded as a maximum set cover problem, which is NP-hard. Using the greedy algorithm of [18], we can obtain a solution with $(1 - 1/e)$ approximation ratio. This algorithm picks k patterns greedily one by one, where each pattern is selected as the one that maximizes the current coverage.

The Baseline Algorithm. A straightforward algorithm works as follows. The algorithm maintains a set $\mathcal{V}$ of vertex sets, where for each vertex set $V' \in \mathcal{V}$, there exists a time point $t \in T$ such that the subgraph of $\mathbb{G}(t)$ induced by V' is a γ-quasi-clique. Initially, $\mathcal{V}$ is empty. We first use the algorithm in [16] to list all γ-quasi-cliques with at least σ vertices in every graph snapshot $\mathbb{G}(t_i)$ ($\forall t_i \in T$). For the γ-quasi-cliques of $\mathbb{G}(t_i)$, let the set of their vertex sets be $\mathcal{V}_i$. We then check $\mathcal{V}_i$ one by one from $i = 1$ to $i = |T|$, and for each $\mathcal{V}_i$, we use it to update $\mathcal{V}$ in two cases. (1) If a vertex set $V' \in \mathcal{V}_i$ is not in $\mathcal{V}$, then we attach V' with a starting time $t_s = t_i$ and add V' to $\mathcal{V}$. (2) If a vertex set $V' \in \mathcal{V}$ is not in $\mathcal{V}_i$, then we attach V' with an ending time $t_e = t_i$, output the subgraph pattern induced by V' and $I' = [t_s, t_e]$ to the result set $\mathcal{R}$ if $|I'| \geq \tau$, and then remove V' from $\mathcal{V}$.

After all qualified temporal subgraph patterns have been collected to $\mathcal{R}$, we call the greedy set-cover algorithm over $\mathcal{R}$, and output the top k subgraph patterns.

Drawbacks of the Baseline Algorithm. The baseline algorithm simply calls an existing algorithm for each graph snapshot $\mathbb{G}(t_i)$, where $i = 1, 2, \ldots, |T|$, which is inefficient since it incurs a lot of redundant computation when there are not many changes between consecutive graph snapshots. Moreover, we need to keep all the quasi-cliques of each snapshot graph (even if they may not last for τ time steps), which consumes a large amount of memory.

4. OUR SOLUTION

In this section, we introduce the main framework of our algorithm for finding the top k diversified temporal subgraph patterns.

4.1 Overview

The high-level idea is to find qualified patterns in a divide-and-conquer manner. We define a mining task as $\mathcal{T} = (\mathbb{G}, S, I)$. Here $\mathbb{G} = (V, \mathbb{E})$ is a temporal graph, S is a subset of V that we have already selected as a subgraph pattern, and I is a time interval. The mining task aims to find all maximal dense subgraphs induced by $V' \subseteq V$ and $I' \subseteq I$ from $\mathbb{G}$, such that $V' \supseteq S$, $|V'| \geq \sigma$ and $|I'| \geq \tau$. We start with the task $\mathcal{T} = (\mathbb{G}, \emptyset, T)$, where we abuse T (w.r.t. $\mathbb{G}$) to also denote the minimal time interval that contains all the time snapshots in T. We process the task $\mathcal{T}$ recursively as follows.

We first remove some vertices and time snapshots of $\mathbb{G}$ by the pruning rules to be introduced in Section 5, as they cannot be included in any qualified subgraph patterns. If all the vertices and time snapshots are removed, then $\mathbb{G}$ does not contain any qualified subgraph patterns, and the task is done. Otherwise, let the remaining graph be $\mathbb{G}' = (V', \mathbb{E}')$, if $\mathbb{G}'$ is dense throughout I, then we add $\mathbb{G}'$ to the result set $\mathcal{R}$, and the task is also done.

If $\mathbb{G}'$ is neither empty nor dense, we divide the task $\mathcal{T}$ recursively into subtasks. Let $\mathcal{I}$ be the set of unpruned time snapshots in I, then we continue with two cases: (Case 1) If $\mathcal{I}$ is a consecutive interval, then we select a vertex v from $V' \setminus S$ according to a total order on vertices (the ordering will be discussed in Section 6.1), and divide $\mathcal{T}$ into two subtasks. One subtask examines the subgraphs that contain v and the other examines the subgraphs that do not contain v. More specifically, $\mathcal{T}$ is divided into $\mathcal{T}_1 = (\mathbb{G}', S \cup \{v\}, \mathcal{I})$ and $\mathcal{T}_2 = (\mathbb{G}' \setminus \{v\}, S, \mathcal{I})$, where $\mathbb{G}' \setminus \{v\}$ is the remaining graph after removing vertex v and all edges incident on v from $\mathbb{G}'$. (Case 2) Otherwise, $\mathcal{I}$ is disjoint, i.e., $\mathcal{I} = [s_1, e_1] \cup [s_2, e_2] \cup \ldots \cup [s_\ell, e_\ell]$, then let $I_i = [s_i, e_i]$ and we divide $\mathcal{T}$ into ℓ subtasks $\mathcal{T}_i = (\mathbb{G}' \cap I_i, S, I_i)$, where $i = 1, 2, \ldots, \ell$ and $\mathbb{G}' \cap I_i$ is a subgraph of $\mathbb{G}'$ induced by V' and I_i. We then process the subtasks recursively.

4.2 Updating the Result Set

We keep at most k subgraph patterns in the result set $\mathcal{R}$ throughout the whole mining process. To do so, we simply add the new qualified subgraphs to $\mathcal{R}$ when $|\mathcal{R}| < k$; otherwise, we maintain $\mathcal{R}$ greedily to increase the coverage $|C(\mathcal{R})|$.

More specifically, we first select a subgraph $\underline{\mathbb{G}}$ from the result set $\mathcal{R}$, where

$$\underline{\mathbb{G}} = \arg\max_{\mathbb{G}' \in \mathcal{R}} |C(\mathcal{R} \setminus \{\mathbb{G}'\})|$$

is the subgraph in $\mathcal{R}$ such that after removing it from $\mathcal{R}$, the remaining coverage is maximized. Let $\mathcal{R}^- = \mathcal{R} \setminus \{\underline{\mathbb{G}}\}$ be the result set after excluding $\underline{\mathbb{G}}$ from $\mathcal{R}$. To decide whether a new qualified subgraph $\mathbb{G}_{\text{new}}$ should be included into $\mathcal{R}$, we replace $\underline{\mathbb{G}}$ with $\mathbb{G}_{\text{new}}$, and check whether the new coverage $|C(\mathcal{R}^- \cup \{\mathbb{G}_{\text{new}}\})|$ becomes larger than the current coverage $|C(\mathcal{R})|$. To allow effective pruning (to be discussed in Section 5), we only replace $\underline{\mathbb{G}}$ with $\mathbb{G}_{\text{new}}$ if

$$|C(\mathcal{R}^- \cup \{\mathbb{G}_{\text{new}}\})| > (1 + 1/k) \cdot |C(\mathcal{R})|, \qquad (1)$$

In other words, we only update the result set if the coverage increases by more than $1/k$ times after updating. According to [2], this updating rule has a guarantee of a $1/4$-approximation ratio w.r.t. the maximum value of $|C(\mathcal{R})|$.

4.3 The Algorithm

Our mining algorithm is sketched in Algorithms 1 and 2. Algorithm 1 first initializes the result set $\mathcal{R}$ (Line 1) and then calls the "Search" procedure detailed in Algorithm 2 with the given temporal graph $\mathbb{G}$, an empty set (of selected vertices) and $\mathbb{G}$'s time interval T (Line 2). Finally, it outputs the computed result set $\mathcal{R}$ (Line 3).

Algorithm 2 shows the recursive procedure of a mining task. It first applies the pruning rules (to be introduced in Section 5) to prune some vertices and time snapshots from $\mathbb{G}$ (Line 1). Then, we continue to process the pruned graph (denoted as $\mathbb{G}_{\text{new}}$) in three cases. (1) If the set of the remaining time points $\mathcal{I}$ is empty, then there is no qualified subgraph and the mining task is done (Lines 2-3); otherwise, (2) if $\mathcal{I}$ contains only one consecutive time interval and $\mathbb{G}_{\text{new}}$ is a γ-quasi-clique through out $\mathcal{I}$, then $\mathbb{G}_{\text{new}}$ is a new qualified subgraph, and the result set $\mathcal{R}$ will be updated as described in Section 4.2 before finishing the mining task (Lines 4-6); otherwise, (3) we continue with two subcases: if $\mathcal{I}$ is a consecutive time interval, then the task is divided into two subtasks according to a selected vertex $v \in V \setminus S$ (Lines 7-10); otherwise, the task is divided into ℓ subtasks according to the disjoint time intervals of $\mathcal{I}$ (Lines 11-13). In both subcases, the subtasks are then processed recursively.

Compared with the baseline algorithm, we now examine the subgraph patterns in the unit of time intervals rather than time points. We also terminate the mining task immediately once we find that no dense subgraph patterns can last for a period of length at least

Algorithm 1 : Top-k Diversified Temporal Subgraph Mining

```
Input: $\mathbb{G} = (V, \mathbb{E})$, $\gamma$, $\sigma$, $\tau$, $k$
Output: $\mathcal{R}$
1.  $\mathcal{R} \leftarrow \emptyset$
2.  **Search**($\mathbb{G}, \emptyset, T$)
3.  **return** $\mathcal{R}$
```

Algorithm 2 : **Search**($\mathbb{G} = (V, \mathbb{E}), S, I$)

```
1.  Apply the pruning rules to shrink $\mathbb{G}$ (see Section 5)
    {Let $\mathbb{G}_{\mathrm{new}}$ be the remaining graph and $\mathcal{I} = I_1 \cup ... \cup I_\ell$ be
    the remaining time snapshots after pruning, where $I_1, ..., I_\ell$
    are the disjoint time intervals of $\mathcal{I}$}
2.  **if** $\mathcal{I} = \emptyset$
3.      **return**
4.  **if** $\ell = 1$ **and** $\mathbb{G}_{\mathrm{new}}$ is a $\gamma$-quasi-clique throughout $\mathcal{I}$
5.      Update $\mathcal{R}$ with $\mathbb{G}_{\mathrm{new}}$ (see Section 4.2)
6.      **return**
7.  **if** $\ell = 1$
8.      select a vertex $v \in V \setminus S$
9.      **Search**($\mathbb{G}_{\mathrm{new}}, S \cup \{v\}, \mathcal{I}$)
10.     **Search**($\mathbb{G}_{\mathrm{new}} \setminus \{v\}, S, \mathcal{I}$)
11. **else**
12.     **for** $i = 1, 2, \ldots, \ell$
13.         **Search**($\mathbb{G}_{\mathrm{new}} \cap I_i, S, I_i$)
```

τ. This pruning opportunity, however, is not utilized by the baseline algorithm. Also, we keep at most k subgraph patterns in the result set throughout our mining procedures, which is more space efficient than the baseline algorithm.

5. PRUNING RULES

In this section, we present our five pruning rules for temporal subgraph mining, which are used in Line 1 of Algorithm 2. In the pruning procedure, we repeat the first four pruning rules to prune the given temporal graph until it cannot be further pruned, and then use the last pruning rule to further prune the remaining graph.

Summary of Pruning Rules. For the temporal graph $\mathbb{G}$ in a given mining task, the first rule prunes the vertices with low degrees and short durations, such that the degree of any vertex in the remaining graph remains at least $\gamma \cdot (\sigma - 1)$ for a period with length at least τ.

The second rule prunes the vertices that are far away from a newly selected vertex v, such that the distances from v to the remaining vertices are within a valid range.

After calculating the bounds on the sizes of the qualified subgraph patterns, the third (resp. fourth) rule removes the snapshot graphs (resp. vertices) that cannot be included in a qualified subgraph pattern according to the bounds.

The last rule is applied only if the time points of the remaining snapshot graphs still form a consecutive interval after the previous pruning. In this case, the task is terminated if no qualified subgraph can increase the coverage by more than $1/k$ times as the updating rule requires.

The Pruning Operations. Before presenting the details of our pruning rules, we first introduce three pruning operations that are commonly used to shrink a temporal graph $\mathbb{G}$.

Operation 1: Edge Removal. Recall that a temporal edge $e = (u, v, I)$ connects vertices u and v within time interval I. After removing this edge, $d_u(t)$ and $d_v(t)$ decrease by 1 for all $t \in I$. Sometimes we may only remove an edge within a given time interval I', and then this edge becomes $(u, v, I \setminus I')$, which will be broken into two edges if $I \setminus I'$ is disjoint after removal. For example, let a temporal edge be $(u, v, [0, 60])$, and if we remove this edge within time interval [20, 50], then this edge is broken into two edges $(u, v, [0, 20])$ and $(u, v, [50, 60])$.

Operation 2: Vertex Removal. If a vertex v is removed from $\mathbb{G}$, then all its adjacent temporal edges are removed. Sometimes we may only remove a vertex within a given time interval I', and thus only its adjacent edges within the time interval I' are removed. Moreover, given a set of disjoint time intervals, $\mathcal{I} = \{[s_1, e_1], \ldots, [s_i, e_i], \ldots, [s_\ell, e_\ell]\}$, if we only remove v w.r.t. $\mathcal{I}$, then v is only removed within time intervals $[s_i, e_i]$ for all $i = 1, 2, ..., \ell$.

Operation 3: Snapshot Graph Removal. Given a set of disjoint time intervals, $\mathcal{I}$, if we remove the snapshot graphs in $\mathcal{I}$, then all the vertices of $\mathbb{G}$ are only removed w.r.t. $\mathcal{I}$, and hence, the snapshot graphs will be empty at any time $t \in \mathcal{I}$.

5.1 The List of Pruning Rules

Now we present our five pruning rules. Let $\mathcal{T} = (\mathbb{G}, S, I)$ be a mining task, where $\mathbb{G} = (V, \mathbb{E})$. For an induced subgraph $\mathbb{G}' = (V', \mathbb{E}', I')$ of $\mathbb{G}$, if $V' \supseteq S$, $|V'| \geq \sigma$, $|I'| \geq \tau$ and $\mathbb{G}'$ is γ-dense throughout I', then we say $\mathbb{G}'$ is a qualified subgraph pattern.

Rule 1: Degree-and-Duration Based Pruning. We first prune the vertices with low degrees or short durations, since they cannot be included in a qualified subgraph pattern.

More specifically, let $v \in V$ be a vertex of $\mathbb{G}$. If $d_v(t) < \gamma \cdot (|S|-1)$, then $d_v(t) < \gamma \cdot (|V'|-1)$ for any superset $V' \supseteq S$. Thus, $d_v(t)$ is too low for v to be included in a γ-dense subgraph at time t. Moreover, if $d_v(t) < \gamma \cdot (\sigma - 1)$, then we also cannot include v in a γ-dense subgraph with size at least σ at time t. Based on these facts, let us define $\mathcal{I}_v^- = \{t \mid t \in I, d_v(t) < \gamma \cdot (\max\{|S|, \sigma\} - 1)\}$, then we can prune v within any time interval in $\mathcal{I}_v^-$.

Let us define $\mathcal{I}_v^+ = \{t \mid t \in I, d_v(t) \geq \gamma \cdot (\max\{|S|, \sigma\} - 1)\}$, and let $[s_1, e_1], [s_2, e_2], ..., [s_\ell, e_\ell]$ be the disjoint time intervals of $\mathcal{I}_v^+$. If $e_i - s_i < \tau$ for some i, where $1 \leq i \leq \ell$, then time interval $[s_i, e_i]$ is too short and v cannot be included in a qualified subgraph pattern at any time $t \in [s_i, e_i]$. Based on this fact, we can prune v within $[s_i, e_i]$ for any $i = 1, 2, ..., \ell$ such that $e_i - s_i < \tau$.

We repeat the above operations for all vertices in $\mathbb{G}$ until no more vertex can be removed within any time interval.

Rule 2: Distance Based Pruning. According to [21], if a vertex is too far away from any selected vertex, it cannot be included in a γ-quasi-clique. Therefore, after selecting a new vertex, we can remove the vertices that are farther away than a distance threshold computed based on γ.

More specifically, if a non-temporal graph $G = (V, E)$ is a γ-quasi-clique, then according to Theorem 1 in [21], we have $\delta_{u,v} \leq f(\gamma)$ for all $u, v \in V$, where $\delta_{u,v}$ is the distance between vertices u and v, and f is a function of γ. As a special case, $f(1) = 1$ and $f(\gamma) = 2$ for $0.5 \leq \gamma < 1$. According to this property, vertices whose distance are larger than $f(\gamma)$ from any selected vertex can be removed. To do so, we define $\delta_{u,v}(t)$ as the distance between vertices u and v in $\mathbb{G}$ at time t, and define $\mathcal{I}_v^*(u) = \{t \mid \delta_{u,v}(t) > f(\gamma)\}$ as the set of time points when the distance between $u, v \in V$ is larger than a given threshold $f(\gamma)$. Then, after adding a new vertex v to S, for every $u \in V \setminus S$, we prune u within any time interval in $\mathcal{I}_v^*(u)$.

Rule 3: Pattern Size Based Pruning. Consider the current task $\mathcal{T} = (\mathbb{G}, S, I)$ again. For a time point $t \in I$, if there does not exist any qualified subgraph pattern $\mathbb{G}' = (V', \mathbb{E}', I')$ such that $t \in I'$, then we call t as a break point. To find break points for any $t \in I$, we first calculate an upper bound $ub(t)$ and a lower bound $lb(t)$ for the size of any qualified subgraph pattern at time t (the calculation will be discussed later in Section 5.2). Obviously, if $ub(t) < \sigma$ or $ub(t) < lb(t)$, then t is a break point. Based on this property, we define $T_b = \{t | t \in I, ub(t) < \sigma \text{ or } ub(t) < lb(t)\}$ as the set of break points.

Then, we can use T_b to find more break points by checking the length of the time interval between any two break points in T_b, which is detailed as follows. If $t_1, t_2 \in T_b$ satisfies $|t_2 - t_1| < \tau$, then for any $t \in [t_1, t_2]$, the time span of any γ-dense subgraph that spans across time t is broken at t_1 and t_2, and hence, these subgraph patterns cannot last for a period with length τ. As a result, all time points $t \in [t_1, t_2]$ are also break points. We denote this expanded set of break points as follows:

$$T_b^+ = \{t | t \in I, \exists t_1, t_2 \in T_b, \text{ s.t. } t_1 \le t \le t_2, t_2 - t_1 < \tau\}.$$

Then, we can prune the snapshot graphs in T_b^+.

Rule 4: Vertex Based Pruning. Recall that V is the vertex set of $\mathbb{G}$. For a vertex $v \in V$ and a time point $t \in I$, if there does not exist any qualified subgraph pattern $\mathbb{G}' = (V', \mathbb{E}', I')$ such that $v \in V'$ and $t \in I'$, then we say that v is disqualified at time t. We will show how to find disqualified vertices in Section 5.3. Moreover, if a vertex v is disqualified at both t_1 and t_2 such that $|t_2 - t_1| < \tau$, then v is also disqualified at any time $t \in [t_1, t_2]$.

Let T_v be the set of time points when v is disqualified. If $v \in S$, then all the time points in T_v are break points and we prune the snapshot graphs in T_v; otherwise (i.e., $v \in V \setminus S$), we prune v at any time $t \in T_v$.

Rule 5: Diversifying Based Pruning. This pruning rule is only called when the set of time intervals (denoted as $\mathcal{I}$) of the remaining snapshot graphs (after repeatedly applying Rules 1-4) consists of only one consecutive time interval.

Recall from Section 4.2 that if Inequality (1) does not hold, we do not update the result set $\mathcal{R}$, and thus can terminate the task earlier. For this purpose, we derive an upper bound on the coverage score after we update $\mathcal{R}$ with any subgraph pattern $\mathbb{G}'$ mined from the current task $\mathcal{T}$.

Let $\Delta(\mathbb{G}') = |C(\mathcal{R}^- \cup \{\mathbb{G}'\})| - |C(\mathcal{R}^-)|$ be the increment of the coverage after adding $\mathbb{G}'$ to $\mathcal{R}^-$. Since we have selected a set S of vertices, we can first calculate $\Delta(\mathbb{G} \cap S)$, where $\mathbb{G} \cap S$ refers to the subgraph of $\mathbb{G}$ induced by S and $\mathcal{I}$. Let $m = \max_{t \in I} ub(t)$ be the maximum allowed number of vertices in a qualified subgraph, and let $\Delta_1, \Delta_2, \ldots, \Delta_{m-|S|}$ be the $(m - |S|)$ largest values of $\Delta(\mathbb{G} \cap \{v\})$ among all $v \in V \setminus S$. Then the upper bound of the coverage is $C^{ub} = |C(\mathcal{R}^-)| + \Delta(\mathbb{G} \cap S) + \sum_{i=1}^{m-|S|} \Delta_i$ (i.e., the coverage after we add into $\mathcal{R}^-$ a pattern $\mathbb{G}'$, which contains S and up to $(m - |S|)$ other vertices). If $C^{ub} < (1 + 1/k) \cdot |C(\mathcal{R})|$, then Inequality (1) cannot hold, and thus, we terminate the mining task $\mathcal{T}$ directly.

5.2 Bounds on Pattern Sizes

Recall that Rule 3 requires the bounds $ub(t)$ and $lb(t)$ on the size (i.e., the number of vertices) of any qualified subgraph pattern at time t. We now derive these bounds.

5.2.1 Degree Based Bounds

The degree of a vertex provides an upper bound on the size of any quasi-clique that contains this vertex. We discuss how to derive the upper bound $u(t)$ for a mining task.

Given a mining task $\mathcal{T} = (\mathbb{G}, S, I)$, for any $t \in I$, let $d_{\min}(t) = \min\{d_v(t) \mid v \in S\}$. Then, $u(t)$ is given by the following lemma.

LEMMA 1. *Let $G' = (V', E')$ be a subgraph of $\mathbb{G}(t)$. If $V' \supseteq S$ and G' is a γ-quasi-clique, then $|V'| \le u(t)$, where $u(t) = \lfloor d_{\min}(t)/\gamma \rfloor + 1$.*

PROOF. According to the definition of γ-quasi-clique, we have $d_v(t) \ge \gamma \cdot (|V'| - 1)$ for all $v \in V'$. Since $V' \supseteq S$, we have $d_{\min}(t) \ge \gamma \cdot (|V'| - 1)$, that is, $|V'| \le \lfloor d_{\min}(t)/\gamma \rfloor + 1 = u(t)$. $\square$

We now derive the lower bound $l(t)$. Let $N_v(t)$ be the set of v's neighbors at time t. Given a set of vertices, S', we first define the *restricted degree* of v w.r.t. S' at time t as $d_v^{S'}(t) = |N_v(t) \cap S'|$, which refers to the number of neighbors of v that are in S' at time t. Let $d_{\min}^S(t) = \min\{d_v^S(t) \mid v \in S\}$. Then, $l(t)$ is given by the following lemma.

LEMMA 2. *Let $G' = (V', E')$ be a subgraph of $\mathbb{G}(t)$. If $V' \supseteq S$ and G' is a γ-quasi-clique, then $|V'| \ge l(t)$, where $l(t) = \lceil (|S| - d_{\min}^S(t) - \gamma)/(1 - \gamma) \rceil$.*

PROOF. According to the definition of γ-quasi-clique, we have $d_v^{V'}(t) \ge \gamma \cdot (|V'| - 1)$ for all $v \in V'$. Since $V' \supseteq S$, we have $d_v^{V'}(t) \le (|V'| - |S|) + d_v^S(t)$, and thus, $|V'| - |S| + d_v^S(t) \ge \gamma \cdot (|V'| - 1)$ for all $v \in V'$. Since $d_v^S(t) \ge d_{\min}^S(t)$, we have $|V'| - |S| + d_{\min}^S(t) \ge \gamma \cdot (|V'| - 1)$, and thus, $|V'| \ge \lceil (|S| - d_{\min}^S(t) - \gamma)/(1 - \gamma) \rceil = l(t)$. $\square$

5.2.2 Sum of Degree Based Bounds

We now make $u(t)$ and $l(t)$ tighter by considering the sum of the restricted degree of individual vertices. Note that if a non-temporal graph $G' = (V', E')$ is a γ-quasi-clique, then for any m vertices in V', the sum of their degrees must be at least $m \cdot \lceil \gamma \cdot (|V'| - 1) \rceil$.

Since $S \subseteq V'$, for any subset of S with m vertices, denoted by S', the sum of all degrees of vertices in S' should be at least $m \cdot \lceil \gamma \cdot (|V'| - 1) \rceil$. Currently, $V' = S$ but the degree sum may be less than $m \cdot \lceil \gamma \cdot (|S| - 1) \rceil$, and we need to include more vertices into V' to make the requirement satisfied for the subset S'. We present how to derive bounds on the number of vertices that needs to be included, using restricted vertex degrees w.r.t. S'. For vertices in S, we compute their sum of restricted degree w.r.t. S' at time t as $\sum_{v \in S} d_v^{S'}(t)$. For vertices in $V \setminus S$ (which are candidates to be included into V'), we sort them in non-increasing order of their restricted degree $d_v^{S'}(t)$, and denote the sorted vertices by $v_1, v_2, \ldots, v_{|V \setminus S|}$. Then, we have the following lemma.

LEMMA 3. *Let $G' = (V', E')$ be a subgraph of $\mathbb{G}(t)$, and S' be a subset of S, if $V' \supseteq S$ and G' is a γ-quasi-clique, then we have*

$$\sum_{v \in S} d_v^{S'}(t) + \sum_{i=1}^{|V' \setminus S|} d_{v_i}^{S'}(t) \ge |S'| \cdot \lceil \gamma \cdot (|V'| - 1) \rceil \qquad (2)$$

PROOF. According to the definition of γ-quasi-clique, we have $\sum_{v \in S'} d_v^{V'}(t) \ge |S'| \cdot \lceil \gamma \cdot (|V'| - 1) \rceil = \text{R.H.S.}$, then we just need to prove that L.H.S. $\ge \sum_{v \in S'} d_v^{V'}(t)$.

Note that $\sum_{v \in S'} d_v^{V'}(t)$ equals the number of edges between S' and V', and thus equals $\sum_{v \in V'} d_v^{S'}(t)$. Thus, we only need to prove that L.H.S. $\ge \sum_{v \in V'} d_v^{S'}(t)$.

We divide V' into two sets S and $V' \setminus S$, then we get

$$\sum_{v \in V'} d_v^{S'}(t) = \sum_{v \in S} d_v^{S'}(t) + \sum_{v \in V' \setminus S} d_v^{S'}(t).$$

Since we have sorted the vertices $v \in V \setminus S$ in descending order of their restricted degrees, we have

$$\sum_{i=1}^{|V' \setminus S|} d_{v_i}^{S'}(t) \ge \sum_{v \in V' \setminus S} d_v^{S'}(t).$$

Thus, according to Inequality (2), we have

$$\text{L.H.S.} \geq \sum_{v \in S} d_v^{S'}(t) + \sum_{v \in V' \setminus S} d_v^{S'}(t) = \sum_{v \in V'} d_v^{S'}(t),$$

which completes the proof. □

Let $u^{S'}(t)$ (resp. $l^{S'}(t)$) be the maximum (resp. minimum) $|V'|$ (computed from S'), such that

$$l(t) \leq l^{S'}(t) \leq |V'| \leq u^{S'}(t) \leq u(t), \tag{3}$$

$$\sum_{v \in S} d_v^{S'}(t) + \sum_{i=1}^{|V' \setminus S|} d_{v_i}^{S'}(t) \geq |S'| \cdot \lceil \gamma \cdot (|V'| - 1) \rceil. \tag{4}$$

To compute $u^{S'}(t)$ and $l^{S'}(t)$, we first compute $\sum_{v \in S} d_v^{S'}(t)$. Then, we sort the vertices in $V \setminus S$ in non-increasing order of their restricted degree $d_v^{S'}(t)$ for every $t \in I$, and hence we obtain the L.H.S. of Inequality (4) for $|V'| = |S|, |S| + 1, ..., u(t)$ by summing up the restricted degree $d_{v_i}^{S'}(t)$ one by one from the sorted list for $i = 1, 2, \dots, u(t) - |S|$. Finally, we compare the sum with R.H.S. to get $u^{S'}(t)$ and $l^{S'}(t)$.

According to Inequality (3), $u^{S'}(t)$ and $l^{S'}(t)$ are tighter than $u(t)$ and $l(t)$. If there is no valid value of $|V'|$ that satisfies Inequalities (3) and (4), then we set $u^{S'}(t) = -1$ and $l^{S'}(t) = |V| + 1$.

Since any subset $S' \subseteq S$ can be selected to compute the bounds $u^{S'}(t)$ and $l^{S'}(t)$, we define

$$u_m(t) = \min_{S' \subseteq S} u^{S'}(t),$$

$$l_m(t) = \max_{S' \subseteq S} l^{S'}(t),$$

then $u_m(t)$ and $l_m(t)$ are tighter than $u^{S'}(t)$ and $l^{S'}(t)$ for any $S' \subseteq S$, and we have the following corollary.

COROLLARY 1. *Let $G' = (V', E')$ be a subgraph of $\mathbb{G}(t)$, if $V' \supseteq S$, $|V'| \geq \sigma$ and G' is a γ-quasi-clique, then we have $l_m(t) \leq |V'| \leq u_m(t)$.*

For efficiency reasons, we only enumerate some subsets $S' \subseteq S$ to compute the bounds, as we will discuss in Section 6.1.

5.2.3 Duration Based Bounds

Note that $u_m(t)$ and $l_m(t)$ are functions of t for $t \in I$, we can further make them tighter using the minimum duration τ.

LEMMA 4. *If there exists $t, t_1, t_2 \in I$, such that*

$$t_1 < t < t_2, \tag{5}$$

$$t_2 - t_1 < \tau \text{ and} \tag{6}$$

$$u_m(t) > \max\{u_m(t_1), u_m(t_2)\}, \tag{7}$$

then no dense subgraph pattern with size $u_m(t)$ at time t can last for a duration with length τ.

PROOF. According to Corollary 1, there does not exist a dense subgraph pattern with size larger than $u_m(t_1)$ at time t_1 or larger than $u_m(t_2)$ at time t_2. That is, there does not exist a dense subgraph pattern with size larger than $\max\{u_m(t_1), u_m(t_2)\}$ at time t_1 or t_2. Then, any dense subgraph pattern with size $u_m(t) > \max\{u_m(t_1), u_m(t_2)\}$ at time t must start after t_1 and end before t_2, and hence, it cannot last for a duration with length $t_2 - t_1 < \tau$. □

According to Lemma 4, we can make u_m tighter if there exists $t, t_1, t_2 \in I$ satisfying Inequalities (5)-(7). In this case, we say u_m is *tighten-able* at time t, and we tighten u_m by setting $u_m(t)$ to $\max\{u_m(t_1), u_m(t_2)\}$. We repeat this operation until u_m is not tighten-able at any time $t \in I$. Then, the upper bound $ub(t)$ used by Rule 3 equals the tightened $u_m(t)$, which can be computed in linear time as follows.

We first represent $u_m(t)$ in s pairs $(t_1, \Delta_1), (t_2, \Delta_2), \dots, (t_s, \Delta_s)$, where s is the number of changes of the value of $u_m(t)$ for $t \in I$, t_i is the time of the i-th change, and Δ_i is the difference of the value of $u_m(t)$ after and before time $t = t_i$. As a special case, t_1 is the starting time of I and Δ_1 equals $u_m(t_1)$. For example, the upper bound $u_m(t)$ shown in Figure 3(a) can be represented in 3 pairs $(0, 8), (20, 6), (40, -4)$. Before explaining how to compute $ub(t)$ in linear time, we need to prove the following lemma.

LEMMA 5. *Let (t_a, Δ_a) and (t_b, Δ_b) be two consecutive pairs in the representation of $u_m(t)$. If*

$$\Delta_a > 0,\ \Delta_b < 0 \text{ and } t_b - t_a < \tau, \tag{8}$$

then $u_m(t)$ is tighten-able at any time $t \in (t_a, t_b)$.

PROOF. Since $\Delta_a > 0$, we have $u_m(t) > u_m(t_a)$ for $t > t_a$; also, since $\Delta_b < 0$, we have $u_m(t) > u_m(t_b)$ for $t < t_b$. Then, let $t_1 = t_a, t_2 = t_b$ and $t \in (t_1, t_2)$, we have $t_1 < t < t_2, t_2 - t_1 < \tau$ and $u_m(t) > \max\{u_m(t_1), u_m(t_2)\}$, and thus, u_m is tighten-able at any time $t \in (t_a, t_b)$. □

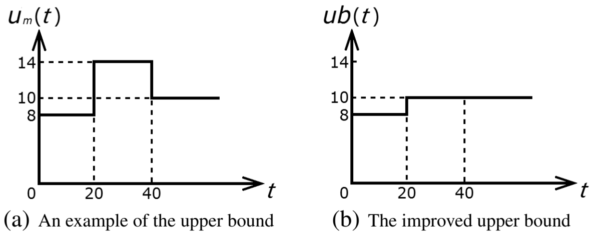
To compute $ub(t)$, we maintain a stack $\mathcal{S}$ of pairs, which is empty initially. Recall that $u_m(t)$ has been represented in s pairs $(t_1, \Delta_1), (t_2, \Delta_2), \dots, (t_s, \Delta_s)$, we push the pairs (t_i, Δ_i) into $\mathcal{S}$ one by one for $i = 1, 2, ..., s$. After we have pushed a pair into $\mathcal{S}$, let the top 2 pairs of $\mathcal{S}$ be (t_a, Δ_a) and (t_b, Δ_b), we then check Inequalities (8) in Lemma 5. If they all hold, then $u_m(t)$ is tighten-able at any $t \in (t_a, t_b)$, and hence, we want to tighten $u_m(t)$ by setting $u_m(t)$ to $\max\{u_m(t_a), u_m(t_b)\}$ for all $t \in (t_a, t_b)$. To handle this operation, we first pop (t_a, Δ_a) and (t_b, Δ_b) out from $\mathcal{S}$. We then define $\Delta = \Delta_a + \Delta_b$, and update $\mathcal{S}$ as follows, if $\Delta > 0$, we push (t_a, Δ) into $\mathcal{S}$; otherwise, if $\Delta < 0$, we push (t_b, Δ) into $\mathcal{S}$. We repeat the previous operations, until Inequalities (8) do not hold for the top 2 pairs of $\mathcal{S}$. Finally, $ub(t)$ is derived as the pairs in $\mathcal{S}$ represent. Apparently, the computation time of $ub(t)$ is linear to s.

To illustrate the computation of $ub(t)$, let $I = [0, 70]$, $\tau = 30$, and $u_m(t)$ be shown in Figure 3(a). To compute $ub(t)$, we first push the pairs $(0, 8), (20, 6), (40, -4)$ into $\mathcal{S}$ one by one. After we have pushed $(40, -4)$ into $\mathcal{S}$, the top 2 pairs are $(20, 6)$ and $(40, -4)$, which satisfy $6 > 0$, $-4 < 0$ and $40 - 20 < \tau$, and thus, u_m is tighten-able at $t \in (20, 40)$. To tighten u_m, we pop $(20, 6)$ and $(40, -4)$ out from $\mathcal{S}$, and we have $\Delta = 6 + (-4) = 2 > 0$, and hence, we push $(20, 2)$ into $\mathcal{S}$. Finally, we get $\mathcal{S} = \{(0, 8), (20, 2)\}$, and $ub(t)$ is derived as the 2 pairs $(0, 8), (20, 2)$ represent, which is shown in Figure 3(b).

Similarly, we have the following lemma on the lower bounds.

LEMMA 6. *If there exists $t, t_1, t_2 \in I$, such that $t_1 < t < t_2$, $t_2 - t_1 < \tau$ and $l_m(t) < \min\{l_m(t_1), l_m(t_2)\}$, then all the dense subgraph patterns with size $l_m(t)$ at time t cannot last for a duration with length τ.*

According to Lemma 6, we can make l_m tighter by a similar procedure, such that $t_1 < t < t_2$, $t_2 - t_1 < \tau$ and $l_m(t) < \min\{l_m(t_1), l_m(t_2)\}$ cannot hold simultaneously for any $t, t_1, t_2 \in I$. Finally, the lower bound $lb(t)$ used by Rule 3 equals the tightened $l_m(t)$.



(a) An example of the upper bound (b) The improved upper bound

Figure 3: An example of u_m before and after improvement.

5.3 Computation of Disqualified Vertices

After deriving the bounds $lb(t)$ and $ub(t)$ on the size of qualified subgraph patterns of $\mathbb{G}$, we can use these bounds to check whether we can include a vertex v in a qualified subgraph pattern at time t. If we cannot include v in a qualified subgraph pattern at time t, then v is disqualified at time t, which will be pruned by Rule 4. To check whether a vertex v is disqualified at time t, we first prove the following lemma.

LEMMA 7. *Let $G' = (V', E')$ be a subgraph of $\mathbb{G}(t)$, if $V' \supseteq S$ and G' is a γ-quasi-clique, then we have $d_v(t) \geq \gamma \cdot (lb(t) - 1)$ for all $v \in V'$.*

PROOF. According to the definition of γ-quasi-clique, we have $d_v(t) \geq \gamma \cdot (|V'| - 1)$ for all $v \in V'$. Since we have $|V'| \geq lb(t)$, we obtain $d_v(t) \geq \gamma \cdot (lb(t) - 1)$. □

According to Lemma 7, if we find that there exists $v \in V, t \in I$, such that $d_v(t) < \gamma \cdot (lb(t) - 1)$, then we cannot include v in a qualified quasi-clique at time t, and hence, v is disqualified at time t. To find more disqualified vertices, we prove the following lemma.

LEMMA 8. *Let $G' = (V', E')$ be a subgraph of $\mathbb{G}(t)$, if $V' \supseteq S$ and G' is a γ-quasi-clique, then we have $ub(t) - |S| + d_v^S(t) \geq \gamma \cdot (lb(t) - 1)$ for all $v \in S$ and we have $ub(t) - |S| - 1 + d_v^S(t) \geq \gamma \cdot (lb(t) - 1)$ for all $v \in V' \setminus S$.*

PROOF. Since G' is a γ-quasi-clique, we have $d_v^{V'}(t) \geq \gamma \cdot (lb(t) - 1)$. Since we have $|V'| \leq ub(t)$, V' contains at most $(ub(t) - |S|)$ vertices that are not in S, and thus, we have $d_v^{V'}(t) \leq ub(t) - |S| + d_v^S(t)$ for $v \in S$ and $d_v^{V'}(t) \leq ub(t) - |S| - 1 + d_v^S(t)$ for $v \in V' \setminus S$. Thus, we obtain $ub(t) - |S| + d_v^S(t) \geq \gamma \cdot (lb(t) - 1)$ for all $v \in S$ and $ub(t) - |S| - 1 + d_v^S(t) \geq \gamma \cdot (lb(t) - 1)$ for all $v \in V' \setminus S$. □

According to Lemma 8, if we find that there exists $v \in S, t \in I$, such that $ub(t) - |S| + d_v^S(t) < \gamma \cdot (lb(t) - 1)$, then we cannot include v in a qualified quasi-clique at time t, and hence, v is disqualified at time t. Also, if we find there exists $v \in V \setminus S, t \in I$, such that $ub(t) - |S| - 1 + d_v^S(t) < \gamma \cdot (lb(t) - 1)$, then v is disqualified at time t.

We can further check whether a vertex $w \in V \setminus S$ can be included in a qualified subgraph pattern at time t by considering the sum of degrees as follows. If we include vertex w in the set of selected vertices, then the set of selected vertices will be $S \cup \{w\}$. Let S' be a subset of S, and $v_1, v_2, \ldots, v_{|V \setminus (S \cup \{w\})|}$ be the vertices in $V \setminus (S \cup \{w\})$ that are sorted by descending order of their restricted degrees $d_v^{S'}(t)$ w.r.t. S' at time t. Let

$$\Sigma = \sum_{v \in S \cup \{w\}} d_v^{S'}(t) + \sum_{i=1}^{m-|S|-1} d_{v_i}^{S'}(t),$$

where m refers to the size of a subgraph pattern. If there does not exists m, such that (1) $lb(t) \leq m \leq ub(t)$, and (2) $\Sigma \geq |S'| \cdot \lceil \gamma \cdot (m-1) \rceil$, then vertex w cannot be included in a qualified subgraph pattern at time t, and hence, w is disqualified at time t.

6. OPTIMIZATIONS

In this section, we provide some optimization techniques that are used to further improve our algorithm.

6.1 Ordering Techniques

Consider a task $\mathcal{T} = (\mathbb{G}, S, I)$, let $\mathcal{I}$ be the set of time intervals of the remaining snapshot graphs after pruning. Recall that when $\mathcal{I}$ is a single consecutive time interval, we will divide $\mathcal{T}$ into two subtasks, $\mathcal{T}_1 = (\mathbb{G}, S \cup \{v_1\}, \mathcal{I})$ and $\mathcal{T}_2 = (\mathbb{G} \setminus \{v_1\}, S, \mathcal{I})$, where v_1 is a selected vertex. Since $\mathcal{T}_2$ may be further divided into $\mathcal{T}_{2,1} = (\mathbb{G} \setminus \{v_1\}, S \cup \{v_2\}, \mathcal{I})$ and $\mathcal{T}_{2,2} = (\mathbb{G} \setminus \{v_1, v_2\}, S, \mathcal{I})$, and $\mathcal{T}_{2,2}$ may be further divided into $\mathcal{T}_{2,2,1} = (\mathbb{G} \setminus \{v_1, v_2\}, S \cup \{v_3\}, \mathcal{I})$ and $\mathcal{T}_{2,2,2} = (\mathbb{G} \setminus \{v_1, v_2, v_3\}, S, \mathcal{I})$, and so on, we can divide the task $\mathcal{T}$ into $|V \setminus S|$ subtasks $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{|V \setminus S|}$ directly, where $\mathcal{T}_i = (\mathbb{G} \setminus \{v_1, v_2, \ldots, v_{i-1}\}, S \cup \{v_i\}, \mathcal{I})$. Therefore, when $\mathcal{I}$ is a consecutive time interval, we need to arrange the $|V \setminus S|$ vertices in order so that we can divide the task $\mathcal{T}$ into $|V \setminus S|$ subtasks directly.

Vertex Ordering. We order the vertices in ascending order of their degrees. That is, the vertices with smaller degrees will be selected first. Since we have selected a set S of vertices, their degrees are more important than the degrees of the vertices we have not selected. According to this intuition, we define a score function of a vertex as the weighted sum of its restricted degree w.r.t. S and its original degree. Formally, the score function of a vertex v in a temporal graph $\mathbb{G}$ at time t is defined as

$$\mathrm{sc}_v^{\mathbb{G}}(t) = |S| \cdot d_v^S(t) + d_v(t).$$

Then, the first selected vertex v_1 is the vertex with the smallest $\sum_{t \in I} \mathrm{sc}_v^{\mathbb{G}}(t)$ among all $v \in V \setminus S$, the second selected vertex v_2 is the vertex with the smallest $\sum_{t \in I} \mathrm{sc}_v^{\mathbb{G} \setminus \{v_1\}}(t)$ among all $v \in V \setminus (S \cup \{v_1\})$, and so on.

Subtasks Ordering. Since the degrees of $v_1, v_2, ...$ are smaller than the remaining vertices, the remaining graph $\mathbb{G} \setminus \{v_1, v_2, ...\}$ is probably denser than the original graph $\mathbb{G}$, and hence, it is easier to find qualified subgraph patterns in the remaining graph. In other words, it is easier to mine dense subgraphs from subtask $\mathcal{T}_i$ with larger i intuitively. According to this intuition, we process the subtask $\mathcal{T}_i$ in descending order of i (i.e. $i = |V \setminus S|, |V \setminus S| - 1, ..., 2, 1$), so that we can find some qualified subgraph patterns more quickly.

The previous discussions are based on the case that $\mathcal{I}$ is a consecutive time interval. When $\mathcal{I}$ is disjoint, we will divide $\mathcal{T}$ into subtasks according to the disjoint time intervals in $\mathcal{I}$. Then, we order the subtasks according to the density of their graphs, where the density is measured by the weighted sum of the degrees of the vertices. Specifically, let $\Sigma = \sum_{v \in V} \sum_{t \in I_i} \mathrm{sc}_v^{\mathbb{G} \cap I_i}(t)/|I_i|$, then a subtask will be processed earlier if its graph has a larger Σ value.

Vertex Subsets Ordering. Recall that any subset $S' \subseteq S$ can be used to derive a bound of pattern size (see Section 5.2.2), we only take $|S|$ subsets into consideration to achieve higher efficiency. We next specify which $|S|$ subsets will be considered.

The first subset is S itself, then the next subset is the previous subset after excluding a vertex with the largest weighted degree. The reason is that the vertices with smaller degrees are harder to satisfy the degree threshold, and hence, they are more likely to affect the bounds of pattern sizes. Formally, let $v_1, v_2, \ldots, v_{|S|}$

be the list of vertices in S sorted by non-increasing order of their weighted degrees $\sum_{t\in I} \mathrm{sc}_v^{\mathbb{G}}(t)$, then the first subset is S itself, the second subset is $S \setminus \{v_1\}$, and so on.

6.2 Search Strategy

Recall that we continue processing a task only when $C^{ub} \geq (1+1/k)\cdot|C(\mathcal{R})|$ holds (see Rule 5 in Section 5.1), a good result set $\mathcal{R}$ may help us to terminate many tasks earlier since the larger $|C(\mathcal{R})|$ is, the less likelihood this condition holds. According to this intuition, we want to collect the first k qualified subgraph patterns (1) as quick as possible, and (2) with the size of the coverage set as large as possible. However, our current search strategy only achieves the first goal, but has not achieved the second goal yet. The reason is that our search is depth first, i.e., we do not proceed to the next subtask until the current subtask is done. Note that all the subgraphs mined from $\mathcal{T} = (\mathbb{G}, S, I)$ contain the set S of vertices, they overlap with each other on any vertex $v \in S$, and hence, our current search strategy cannot get a large initial coverage.

To achieve both goals, we propose a quick search algorithm, which runs with a user defined parameter ℓ. This search strategy proceeds to the next subtask earlier according to the value of ℓ. To specify how the algorithm works, we first define the *hardness* of a task $\mathcal{T}$, denoted as $h(\mathcal{T})$. If $\mathcal{T}$ can be done without dividing into subtasks, i.e., the graph is totally pruned, or is dense throughout its time interval, then $h(\mathcal{T}) = 0$. Otherwise, $h(\mathcal{T})$ is defined in a recursive way as follows. Let $h(\mathcal{T}_i)$ be the hardness of the i-th subtask of $\mathcal{T}$, $i = 1, 2, \ldots, s$, where s is the number of subtasks of $\mathcal{T}$. Let $h_m(\mathcal{T}) = \max\{h(\mathcal{T}_i) \mid 1 \leq i \leq s\}$ be the maximum hardness of the subtasks of $\mathcal{T}$, and let $h_c(\mathcal{T}) = |\{i \mid h(\mathcal{T}_i) = h_m(\mathcal{T}), 1 \leq i \leq s\}|$ be the number of subtasks of $\mathcal{T}$ with the maximum hardness. Then we define $h(\mathcal{T}) = h_m(\mathcal{T})$ if $h_c(\mathcal{T}) = 1$, and $h(\mathcal{T}) = h_m(\mathcal{T}) + 1$ otherwise.

The quick search algorithm only goes into the first subtask that has a large enough hardness, then passes through the remaining subtasks quickly. More specifically, the quick search algorithm with parameter ℓ works as follows. If $\mathcal{T}$ is divided into subtasks, it recursively calls the quick search algorithm with parameter ℓ to handle the subtasks $\mathcal{T}_1, \mathcal{T}_2, \ldots$, one by one. Once a subtask $\mathcal{T}_i$ is finished, it checks whether $h(\mathcal{T}_i) \geq \ell$ or not. Once $h(\mathcal{T}_i) \geq \ell$ holds, it passes through the remaining subtasks quickly as follows. It first checks whether $\ell > 0$ or not. If $\ell = 0$, it terminates immediately without handling the remaining subtasks $\mathcal{T}_{i+1}, \mathcal{T}_{i+2}, \ldots$; otherwise (i.e. $\ell > 0$), it handles the remaining subtasks $\mathcal{T}_{i+1}, \mathcal{T}_{i+2}, \ldots$ by recursively calling the quick search algorithm with parameter $\ell - 1$ only. We have the following theorem on the efficiency of the quick search algorithm.

THEOREM 1. *The quick search algorithm with parameter ℓ handles at most $r_{\max}^{\ell+1} \cdot s_{\max}^{\ell}$ tasks, where $r_{\max}$ is the maximum depth of recursions handling a task, and $s_{\max}$ is the maximum number of subtasks that a task is divided into.*

PROOF. We first prove that there are at most $R(\ell) = r_{\max}^{\ell+1} \cdot s_{\max}^{\ell}$ tasks that need to be handled in a task $\mathcal{T}$ with hardness ℓ. We prove this proposition by induction. The base case is $\ell = 0$. According to the definition of hardness, if $\mathcal{T}$ needs to be divided, it must be divided into at most one subtask $\mathcal{T}_1$, and $\mathcal{T}_1$ has at most one subtask $\mathcal{T}_{1,1}$, and $\mathcal{T}_{1,1}$ has at most one subtask $\mathcal{T}_{1,1,1}$, and so on. Then the total number of tasks needs to be handled in $\mathcal{T}$ is at most $r_{\max}$, which proves the base case of the proposition.

For the induction case, suppose the proposition holds for the tasks with hardness $\ell - 1$, and the hardness of a task $\mathcal{T}$ is ℓ. Then there are at most one subtask of $\mathcal{T}$ with hardness ℓ. Without losing generality, let $h(\mathcal{T}_1) = \ell$, then we have $h(\mathcal{T}_i) \leq \ell - 1$ for $i = 2, 3, \ldots$. Then there are at most one subtask of $\mathcal{T}_1$ with hardness ℓ, and without loss of generality, let $h(\mathcal{T}_{1,1}) = \ell$, then we have $h(\mathcal{T}_{1,i}) \leq \ell - 1$ for $i = 2, 3, \ldots$. Then the same argument can be applied to $\mathcal{T}_{1,1}$, then $\mathcal{T}_{1,1,1}$, and so on. Hence, the total number of tasks that need to be handled in $\mathcal{T}$ is at most $r_{\max} \cdot s_{\max} \cdot R(\ell - 1) = R(\ell)$, which proves the induction case and the proposition holds for all ℓ.

We then prove, by induction, that the quick search algorithm with parameter ℓ handles at most $R(\ell) = r_{\max}^{\ell+1} \cdot s_{\max}^{\ell}$ tasks. The base case is $\ell = 0$. According to the quick search algorithm, if $\mathcal{T}$ needs to be divided, it only handles the first subtask $\mathcal{T}_1$. If $\mathcal{T}_1$ needs to be divided, it only handles the first subtask $\mathcal{T}_{1,1}$, and so on. Then the total number of tasks that need to be handled in $\mathcal{T}$ is at most $r_{\max}$, which proves the base case of the theorem.

For the induction case, suppose the theorem holds for parameter $\ell - 1$, then the algorithm with parameter ℓ first handles some tasks with hardness at most $\ell - 1$, then handles a task with parameter ℓ, and then handles the remaining tasks with parameter $\ell - 1$. By similar arguments in the previous proof, the algorithm handles at most $r_{\max} \cdot s_{\max} \cdot R(\ell - 1) = R(\ell)$ tasks, which proves the induction case and the theorem holds for all ℓ. □

Apparently, both $r_{\max}$ and $s_{\max}$ cannot exceed $|V| + |T|$. So according to Theorem 1, given a constant value ℓ, the running time of the quick search algorithm is a polynomial of $|V|$ and $|T|$. However, the quick search algorithm may miss some qualified subgraph patterns. To avoid missing any qualified subgraph pattern, we also propose a complete search algorithm, which works as follows. It calls the quick search algorithm with parameter $\ell = 0, 1, 2, \ldots$ for the initial task $\mathcal{T}$, until $\ell = h(\mathcal{T})$ holds. Then we have the following theorem on the performance of the complete search algorithm.

THEOREM 2. *The complete search algorithm achieves the $1/4$ approximation ratio on maximizing $|C(\mathcal{R})|$.*

PROOF. We first show that a task with hardness ℓ will be completely handled by the quick search algorithm with parameter ℓ. We prove the proposition by induction. Apparently, this proposition holds for the base case $\ell = 0$. For the induction case, suppose the tasks with hardness $\ell - 1$ will be completely handled by the quick search algorithm with parameter $\ell - 1$, and $h(\mathcal{T}) = \ell$. Recall that the algorithm with parameter ℓ handles the subtasks of $\mathcal{T}$ by three steps: (1) first handles some subtasks whose hardness is at most $\ell - 1$ with parameter ℓ, (2) then handles a subtask whose hardness is ℓ with parameter ℓ, (3) then handles the remaining subtasks whose hardness is at most $\ell - 1$ with parameter $\ell - 1$. Apparently, the subtasks in step (1) and (3) will be completely handled, it remains to show that the subtask in step (2) will be completely handled. Since the same argument can be recursively applied to the smaller subtasks of the subtask in step (2), they will be completely handled. Thus the proposition holds for all cases.

Since the complete search algorithm finally calls the quick search algorithm with parameter $\ell = h(\mathcal{T})$ for the initial task $\mathcal{T}$, this task will be completely handled. Then according to the updating rule described in Section 4.2 and the theoretical results in [2], the complete search algorithm achieves the $1/4$ approximation ratio on maximizing $|C(\mathcal{R})|$, which completes the proof. □

7. EXPERIMENTAL RESULTS

This section studies the performance of our algorithms. The experiments were conducted on a PC with Intel Core i5-3570 CPU at 3.40GHz and 32GB RAM. The operation system is Linux. Our algorithms are implemented using C++ and compiled by g++.

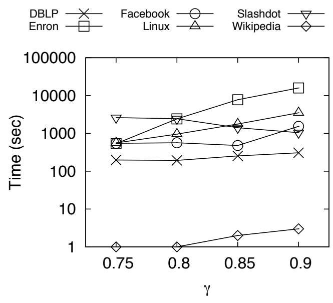


Figure 4: Effect of γ

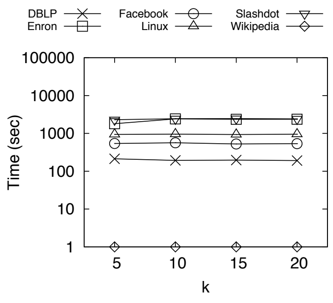


Figure 5: Effect of k

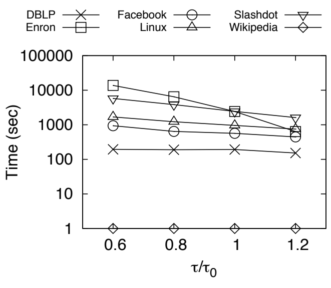


Figure 6: Effect of τ

Datasets. We used six real datasets in our experiments. Five datasets are social networks and one is hyperlinks, which are all downloaded from KONECT (http://konect.uni-koblenz.de/). In which, "DBLP coauthor" dataset is the co-authorship between authors in DBLP, "Enron employees" dataset is the communication between employees in Enron, "Facebook wall posts" dataset is the wall posts between users in Facebook, "Linux kernel mails" dataset is the emails between email addresses in Linux kernel, "Slashdot" dataset is the communication between users in Slashdot, and "Wikipedia simple-En" dataset is the hyperlinks between Wikipedia articles in simple English. The sizes of these datasets are shown in Table 1.

Table 1: Dataset size and duration

Dataset	$\lvert V\rvert$	$\lvert\mathbb{E}\rvert$	Duration
DBLP coauthor	1,314,050	18,986,618	35 years
Enron employees	87,273	1,148,072	4 years
Facebook wall posts	46,952	876,993	4 years
Linux kernel mails	63,399	1,096,440	8 years
Slashdot threads	51,083	140,778	3 years
Wikipedia simple-En	100,312	1,627,472	9 years

7.1 Experiments on Algorithm Efficiency

We first report the efficiency of our algorithms. We report the running time of both the complete search algorithm (denoted as "**Complete**") and the quick search algorithm (denoted as "**Quick**"), compared with the baseline algorithm (denoted as "**Baseline**") described in Section 3. We set $\gamma = 0.8$ and $k = 10$ as the default values, while the default values of σ and τ for each dataset are shown in Table 2. Note that if σ and/or τ are too large, there will be no qualified patterns; while if σ and/or τ are too small, then the coverage of the top k patterns will be too small. We study the effects of different values of the parameters in Section 7.3. We also set $\ell = 2$ as the default value for the quick search algorithm.

Table 2: Default values of σ and τ

Dataset	σ	τ
DBLP coauthor	20	1 year
Enron employees	32	4 months
Facebook wall posts	8	4 months
Linux kernel mails	32	4 months
Slashdot threads	6	4 months
Wikipedia simple-En	18	8 months

Table 3 reports the running time of the three algorithms on each dataset. Note that we terminated an algorithm when it ran for more than 10 days (i.e., 864,000 seconds). The result shows that Quick is 2 to 3 orders of magnitude faster than Baseline. Baseline could not complete the process in 10 days for 5 out of the 6 datasets, showing the hardness of the problem studied in this paper. Complete also achieves good performance for 4 out of the 6 datasets. In fact, for 3 of the 4 datasets, Complete achieves the same performance as Quick, which is because the search space of Complete is already small (as reflected by the running time) and the strategy used in Quick cannot further reduce the search space. However, for the other 3 datasets that take much longer time to process, the optimization of Quick becomes vital and significant reduction in the running time is observed compared with Complete.

Table 3: Running time (in sec)

Dataset	Baseline	Complete	Quick
DBLP coauthor	>864,000	198	198
Enron employees	>864,000	>864,000	2,417
Facebook wall posts	>864,000	560	560
Linux kernel mails	>864,000	7,293	958
Slashdot threads	>864,000	3,697	2,438
Wikipedia simple-En	1,656	1	1

7.2 Experiments on Quality of Results

Theorem 2 shows that theoretically Complete has a $1/4$ approximation ratio, while Quick is heuristic. Thus, we also show practically how good the results obtained by Complete and Quick are. To do this, we compare the results obtained by Complete and Quick with that obtained by an "**Enumerate-all**" algorithm, which lists all the qualified subgraph patterns and then computes the top-k result by the greedy algorithm for the maximum set cover problem. The result set obtained by Enumerate-all is guaranteed to have a $(1 - 1/e)$ approximation ratio of the optimal result. We used the default values of the parameters as in Section 7.1 for the algorithms.

Table 4 reports the coverage (i.e., $|C(\mathcal{G})|$) of the top k patterns obtained by each of the three algorithms on each dataset. The result shows that the top k patterns obtained by Complete achieves a coverage far better than the theoretical $1/4$ approximation ratio, and practically the approximation ratio is over 0.8 in all the datasets and over 0.9 in 4 datasets. In fact, for 2 datasets the results of Complete are exactly the same as Enumerate-all and for another one the approximation ratio is 0.997.

Table 4: Coverage of top-k results

Dataset	Enumerate-all	Complete	Quick
DBLP coauthor	34,401,846,400	34,306,345,600	34,306,345,600
Enron employees	2,307,833,012	2,093,446,876	2,093,446,876
Facebook wall posts	1,297,815,288	1,297,815,288	1,297,815,288
Linux kernel mails	2,258,325,198	1,789,498,564	1,361,365,217
Slashdot threads	615,257,760	530,798,960	530,798,960
Wikipedia simple-En	1,043,388,109	1,043,388,109	1,043,388,109

The results obtained by Quick are exactly the same as those obtained by Complete for 5 out of the 6 datasets, but Quick is much faster than Complete for processing 3 datasets as shown in Table 3. Thus, the results of this experiment demonstrate that Quick is not only fast but also computes high-quality results.

7.3 Effects of Different Parameters

In this set of experiments, we show the performance of our algorithm, Quick, by varying the different parameters. We first vary $\gamma = 0.75, 0.8, 0.85, 0.9$, while keeping the other parameters as their default values as in Section 7.1.

Figure 4 shows that the running time of Quick can vary considerably with varying values of γ. For some datasets, the running time increases when γ increases, because it is easier to collect the first k qualified subgraph patterns into the result set with a smaller γ, and once we have collected k patterns, we can apply Pruning Rule 5 to terminate a task earlier. However, there are also a few datasets for which the running time remains rather stable or even decreases when γ increases, which can be explained as follows. From Lemma 1 we have $u(t) = \lfloor d_{\min}(t)/\gamma \rfloor + 1$, which decreases if γ increases. Since $ub(t)$ is tightened from $u(t)$, and the snapshot graphs at time t will be pruned by Pruning Rule 3 if $ub(t) < \sigma$, more snapshot graphs will be pruned when γ increases. Thus, the result shows that the effect of γ on the performance of the algorithm varies for different datasets, but overall, even if the running time increases, it only increases linearly in terms of γ.

Next we vary $k = 5, 10, 15, 20$. Figure 5 shows that the running time of Quick remains quite stable for different values of k for all the datasets. Note that we could set larger values of k, but the quality of the patterns degrades significantly when k is larger than 20 (i.e., they do not increase the coverage much). We also vary τ to be $0.6, 0.8, 1, 1.2$ times of the default value, i.e., τ_0 in Figure 6. The result shows that the running time of Quick decreases when τ increases, which is because we can prune more short-duration patterns by Rule 1 and obtain a tighter bound by duration-based bounds (see Section 5.2.3).

8. RELATED WORK

Existing work on temporal graphs is mostly related to temporal paths and their applications [6, 12, 13, 19, 22, 23, 24, 28, 27]. More applications of temporal graphs and the sources of temporal graph data can be found in surveys on temporal graphs [6, 9, 17]. None of these works study mining dense subgraph patterns in a temporal graph, and the algorithms of mining dense temporal subgraph patterns are also totally different from any of the existing works.

Many different types of subgraph structures that have high density have been proposed and studied for real-world network analysis, including maximal cliques [5], quasi-clique [1], densest subgraphs [7, 11], k-core [3], k-truss [26], and so on. The well-known definition of quasi-clique was introduced by Abello et al. in [1]. Similar with exact maximal cliques, the problem of enumerating quasi-cliques is NP-hard. In [16], Liu et al. proposed several effective pruning rules for mining quasi-cliques. In [25], Tsourakakis et al. showed that quasi-cliques are high-quality subgraphs. Readers can refer to the survey [14] on dense subgraphs for more comprehensive understanding. All the above discussed works are studied in non-temporal graphs. To our knowledge, our work is the first one for mining quasi-cliques in a temporal graph, and we proposed powerful pruning rules and optimization techniques that make use of time information in a temporal graph.

9. CONCLUSIONS

We proposed the problem of mining the top k diversified temporal subgraph patterns in a temporal graph, and presented a complete search algorithm and a quick search algorithm with effective pruning techniques and search strategies. Our experimental results show that our algorithms are orders of magnitude faster than a baseline algorithm and obtain high-quality results.

10. REFERENCES

[1] J. Abello, M. G. C. Resende, and S. Sudarsky. Massive quasi-clique detection. In *LATIN*, pages 598–612, 2002.

[2] G. Ausiello, N. Boria, A. Giannakos, G. Lucarelli, and V. T. Paschos. Online maximum k-coverage. In *Fundamentals of Computation Theory*, pages 181–192. Springer, 2011.

[3] V. Batagelj and M. Zaversnik. An O(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[4] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. In *ESA*, pages 660–671, 2006.

[5] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[6] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[7] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989.

[8] J. Gudmundsson and M. J. van Kreveld. Computing longest duration flocks in trajectory data. In *GIS*, pages 35–42, 2006.

[9] P. Holme and J. Saramäki. Temporal networks. *CoRR*, abs/1108.1780, 2011.

[10] H. Jeung, H. T. Shen, and X. Zhou. Convoy queries in spatio-temporal databases. In *ICDE*, pages 1457–1459, 2008.

[11] S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP*, pages 597–608, 2009.

[12] G. Kossinets, J. M. Kleinberg, and D. J. Watts. The structure of information pathways in a social communication network. In *KDD*, pages 435–443, 2008.

[13] V. Kostakos. Temporal graphs. *Physica A: Statistical Mechanics and its Applications*, 388(6):1007–1023, 2009.

[14] V. E. Lee, N. Ruan, R. Jin, and C. C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. 2010.

[15] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *PVLDB*, 3(1):723–734, 2010.

[16] G. Liu and L. Wong. Effective pruning techniques for mining quasi-cliques. In *ECML/PKDD*, pages 33–49, 2008.

[17] M. Müller-Hannemann, F. Schulz, D. Wagner, and C. D. Zaroliagis. Timetable information: Models and algorithms. In *ATMOS*, pages 67–90, 2004.

[18] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1):265–294, 1978.

[19] R. K. Pan and J. Saramäki. Path lengths, correlations, and centrality in temporal networks. *Phys. Rev. E*, 84:016105, 2011.

[20] P. M. Pardalos and S. Rebennack. Computational challenges with cliques, quasi-cliques and clique partitions in graphs. In *Experimental Algorithms*, pages 13–22. Springer, 2010.

[21] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *KDD*, pages 228–238, 2005.

[22] N. Santoro, W. Quattrociocchi, P. Flocchini, A. Casteigts, and F. Amblard. Time-varying graphs and social network analysis: Temporal indicators and metrics. *CoRR*, abs/1102.0629, 2011.

[23] J. Tang, M. Musolesi, C. Mascolo, and V. Latora. Temporal distance metrics for social network analysis. In *WOSN*, pages 31–36, 2009.

[24] J. Tang, M. Musolesi, C. Mascolo, and V. Latora. Characterising temporal distance and reachability in mobile and online social networks. *Computer Communication Review*, 40(1):118–124, 2010.

[25] C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *KDD*, pages 104–112, 2013.

[26] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[27] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. *PVLDB*, 7(9):721–732, 2014.

[28] B.-M. B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 14(2):267–285, 2003.

[29] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang. Diversified top-k clique search. In *ICDE*, pages 387–398, 2015.