Mining Stable Quasi-Cliques on Temporal Networks

Longlong Lin, Pingpeng Yuan, *Member, IEEE*, Rong-Hua Li, Jifei Wang, Ling Liu, *Fellow, IEEE*, and Hai Jin, *Fellow, IEEE*

***Abstract*—Real-world networks, such as phone-call networks and social networks, are often not static but temporal. Mining cohesive subgraphs from static graphs is a fundamental task in network analysis and has been widely investigated in the past decades. However, the concepts of cohesive subgraphs shift from static to temporal graphs raise many important problems. For instance, how to detect stable cohesive subgraphs on temporal networks such that the nodes in the subgraph are densely and stably connected over time. To address this problem, we resort to the conventional quasi-clique and propose a new model, called maximal ρ-stable (δ, γ)-quasi-clique, to capture both the cohesiveness and the stability of a subgraph. We show that the problem of enumerating all maximal ρ-stable (δ, γ)-quasi-cliques is NP-hard. To efficiently tackle our problem, we first devise a novel temporal graph reduction algorithm to significantly reduce the temporal graph without losing any maximal ρ-stable (δ, γ)-quasi-clique. Then, on the reduced temporal graph, we propose an effective branch and bound enumeration algorithm, named *BB&SCM*, with four carefully designed pruning techniques to accomplish the enumeration process. Finally, we conduct extensive experiments on seven real-world temporal graphs, and the results demonstrate that the temporal graph reduction algorithm can safely reduce 98% nodes of the temporal graph (with millions of nodes and edges) and *BB&SCM* is at least two orders of magnitude faster than the baseline algorithms. Moreover, we also evaluate the effectiveness of our model against other baseline models.**

***Index Terms*—Quasi-clique, stable cohesive subgraph detection, temporal networks.**

Manuscript received January 17, 2020; revised July 11, 2020 and September 29, 2020; accepted March 28, 2021. Date of publication May 6, 2021; date of current version May 18, 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1004002; in part by NSFC under Grant 61932004 and Grant 61672255; and in part by the Fundamental Research Funds for the Central Universities under Grant HUST2020JYCXJJ068. The work of Ling Liu was supported in part by the National Science Foundation under Grant NSF 2038029, Grant NSF 2026945, and Grant NSF 1564097; and in part by IBM Faculty Award. This article was recommended by Associate Editor R. Wisniewski. *(Corresponding author: Pingpeng Yuan.)*

Longlong Lin, Pingpeng Yuan, Jifei Wang, and Hai Jin are with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: longlonglin@hust.edu.cn; ppyuan@hust.edu.cn; jifeiwang@foxmail.com; hjin@hust.edu.cn).

Rong-Hua Li is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China.

Ling Liu is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TSMC.2021.3071721.

Digital Object Identifier 10.1109/TSMC.2021.3071721

I. Introduction

MANY real-world networks are often temporal networks in nature, whose each edge exists for a certain period of time [1]. For example, in a phone-call network [2], where an edge $(u, v, [t_s, t_e])$ indicates that the two individuals u and v have a phone call during time interval $[t_s, t_e]$. In a face-to-face contact network [3], the connection between two persons was made at a specific time. Mining temporal networks is an indispensable ingredient to better understand the potential phenomena of interactions among entities in a fine-grained manner [4]. Thus, many traditional problems have been extended to temporal networks, such as pattern matching [5], shortest path and reachability queries [6], [7], and minimum spanning tree search [8].

Since it is important to detect cohesive subgraphs where the nodes are densely connected within the subgraph, many cohesive subgraph models have been proposed [9]. However, all of these models ignore the temporal dimension of the network, thereby may fail to detect some significant temporal patterns such as trend of cooperation, evolution of events and traffic hotspots. Until very recently, some works were done on mining temporal cohesive subgraphs [2], [10]. Despite significant success, all of them only considered the cohesiveness but not the stability of a subgraph, thus their solutions cannot be directly applied to detect stable cohesive subgraphs. In this work, we move beyond the static cohesive subgraph models and study the problem of detecting stable cohesive subgraphs in temporal networks. The goal is to detect some subgraphs in temporal networks such that the nodes in each subgraph are densely and stably connected over time (a motivation example is illustrated in Fig. 1). Detecting such subgraphs enables us to reveal stable components in temporal graphs.

Predicting Collaboration Tendency: Scientific collaboration networks (e.g., DBLP) model the collaboration relationships between authors. Finding stable coauthors that frequently and densely collaborate can help us predict their collaboration tendency. For a more granular analysis of the relationship between authors, scientists usually organize coauthorship networks as temporal networks [2], [10]. By identifying the stable cohesive subgraph from the above temporal coauthorship networks, we are able to speculate that the authors in the stable cohesive subgraph are likely to coauthor papers together again in the near future, because they frequently coauthored papers with each other in multiple time periods.

Team Recommendation: Online e-Sports games (e.g., League of Legends) are usually temporal networks [11]. Players would like to cooperate with others in different levels. Some cooperation is frequent while others may be

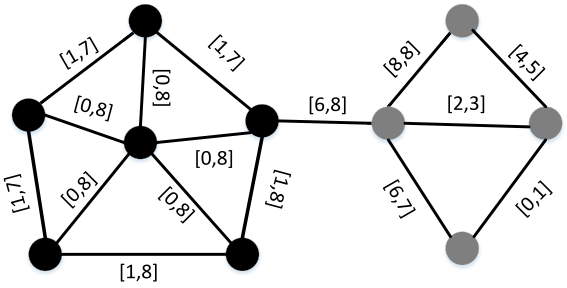


Fig. 1. Motivation example. The time intervals on edges indicate when two nodes interact. If we ignore the time information on edges, the black (or gray) nodes can form a cohesive subgraph. When we consider the temporal properties of edges, the black nodes form a stable cohesive subgraph since their interaction is concentrated and stable over time while the interaction between gray nodes is loose and not stable enough.

occasional. Identifying the stable co-players that frequently play the game can help users determine their cooperation levels. For example, game coaches would like to choose the stable teams to participate in the large competition (e.g., League of Legends Championship Series) because they have frequent and close cooperations and, thereby, are more likely to win the competition.

To model the stable cohesive subgraph in temporal networks, we first define the (δ, γ)-quasi-clique based on the static cohesive subgraph model γ-quasi-clique [12]. A (δ, γ)-quasi-clique consists of vertex set H and time interval T, indicating that all vertices in H densely interact with each other during T from an average interaction intensity perspective. Namely, the average degree of every node of H over all timestamps of T is at least γ times the number of nodes of H. The (δ, γ)-quasi-clique is essentially different from γ-quasi-clique, which incorporates the time information to enable better analysis of the temporal networks. Subsequently, we propose the ρ-stable (δ, γ)-quasi-clique to capture the stable cohesive subgraphs on temporal graphs. A temporal subgraph is a ρ-stable (δ, γ)-quasi-clique if it is a connected temporal graph and its community stability is no less than ρ (Section III-B for details). In a nutshell, our main contributions are summarized as follows.

Novel Stable Cohesive Subgraph Model: We develop a novel temporal model, called maximal ρ-stable (δ, γ)-quasi-clique (*MSQC*), to characterize stable cohesive subgraphs on temporal graphs. We demonstrate that the standard maximal γ-quasi-clique problem is a special case of *MSQC*. Since the problem of enumerating all maximal γ-quasi-cliques in a static graph is NP-hard [13], our problem is also NP-hard.

Efficient and Effective Algorithms: To enumerate efficiently all *MSQC*, we first devise an effective temporal graph reduction algorithm to reduce the original graph size with near-linear time complexity. The temporal graph reduction algorithm applies a powerful stability-based pruning technique to iteratively remove vertices. The core of this pruning is to check whether the stability of the community containing vertex u is less than ρ. We show that a naive solution to tackle this check is intractable. Thus, we propose an efficient dynamic programming algorithm, which can be done in linear time with respect to (w.r.t.) the number of timestamps. Then, on the reduced temporal graph, we proposed distance-based and bound-based pruning to narrow down the search space or terminate tasks earlier. Additionally, excluding-node and including-node pruning strategies are also devised to prune some unqualified nodes during the search. With these techniques, we final develop an effective branch and bound algorithm to find all *MSQC*.

Experimental Results: We thoroughly evaluate our solutions on seven real-world temporal graphs. The results show that our best algorithm is much faster than the baselines on all datasets. For example, on a temporal graph with millions of vertices and edges, our best algorithm consumes less than 600 s to identify all *MSQC* with most parameter settings; however, the baseline algorithms cannot get the results within one day. Additionally, the temporal graph reduction algorithm can prune 98% vertices of the original graph. Furthermore, we also evaluate the effectiveness of our model against three state-of-the-art baseline models. The results show that our model can indeed identify meaningful stable cohesive subgraphs that cannot be found by baseline models.

II. Related Work

Temporal Network Analyzing: Temporal dynamic is an essential feature for real-world networks. Different from general static networks, the edges of temporal networks are associated with timestamps. It thus raises several new challenges for temporal network analysis, including storage, query and mining [1]. Many existing studies have been done for temporal network analysis, including pattern matching [5], shortest path and reachability queries [6], [7], and minimum spanning tree search [8]. Nevertheless, only a few recent researches are tailored to detect subgraphs on temporal networks [2], [3], [10], [14]–[18]. For example, Ma *et al.* [15] modeled the heavy subgraph that maximizes the sum of edge weights in a special temporal graph with unchanging vertices and edges but changing edge weights. Kumar and Calders [14] investigated the properties of temporal cycle and used the temporal cycle to detect fraud. The core decomposition and clique enumeration have also been studied in temporal networks in [3] and [16], respectively. The most related work to ours are [2] and [10]. Specifically, Yang *et al.* [2] investigated the problem of detecting diversified γ-denses in temporal networks, in which γ-dense requires the γ-quasi-clique exists on any timestamp in a given time interval. Note that the γ-dense is different from our model, because our (δ, γ)-quasi-clique requires that the γ-quasi-clique condition is satisfied "on average," resulting in that those temporal subgraphs that are not γ-dense may be (δ, γ)-quasi-cliques by our definition, and our model takes the community stability into account that is not considered by γ-dense. Li *et al.* [10] proposed persistent community model involving a vertex set and an interval set $\mathcal{I}$, which persistently maintains a k-core in any θ-length subinterval of every interval of $\mathcal{I}$. However, the model may not capture potential stable cohesive subgraphs, because persistent k-core poses a very strong constraint. Furthermore, the model simply regards multiple interactions between two individuals as one interaction during any θ-length subinterval. However, our model considers the frequency of interactions within the cohesive subgraph. The experimental evaluation also show that the

two models fail to detect the stable cohesive subgraphs that are found by our model. Thus, our model is the first work that explores the stable cohesive subgraphs on temporal graphs.

Cohesive Subgraph Mining: Cohesive subgraph mining has received much attention over past decades [9]. Notable examples include clique [19], [20], densest subgraph [21]–[23], γ-quasi-clique [12], [24], [25], k-core [26]–[28], and k-truss [29]. The Monte Carlo algorithm with a binary search strategy was employed to find the maximum clique with high probability [20]. Tsourakakis [22] generalized the traditional densest problem [21] to a k-clique densest subgraph problem, where the density is defined as the number of k-clique divided by the size of vertices. Liu and Wong [12] proposed an enumeration algorithm based on the depth-first search and several effective pruning techniques for mining all maximal quasi-cliques. Li *et al.* [27] investigated influence communities by applied k-core. Meanwhile, the cohesive subgraph also is a building block toward more complex tasks, including community mining [30], [31], reachability query [32], and event identification [33].

Community Detection Over Dynamic Networks: Another related line of work is community detection over dynamic networks [34]. However, they differ from the temporal community detection problem as studied in this article. First, the two graphs model different cases. Networks are used to model the interactions between entities. In some cases, the edges represent sequences of instantaneous interactions, which are not continuously active. However, in other cases, edges are active for non-negligible periods of time. Dynamic networks model the first cases as a sequence of edge/node additions/deletions. Temporal networks capture temporal information of interactions occurred in the second cases, resulting in that temporal networks describe data with fine granularity [4]. Second, the solutions for them are different. Research on community detection over dynamic networks usually focus on the efficient maintenance or evolution of community structure. Thus, it typically considers communities on the current snapshot. For example, Epasto *et al.* [35] introduced an efficient incremental algorithm to maintain the densest subgraph on dynamic networks, where the edges are deleted or added in each snapshot. Li *et al.* [36] proposed an efficient algorithm to maintain the k-core over dynamic networks. Qiao *et al.* [37] proposed a framework based on strong and weak events to analyze community evolution in dynamic networks. Unlike these studies, temporal community detection typically considers communities that span a time interval, e.g., the stable cohesive subgraph model proposed in our work.

III. Notations and Problem Definition

A. Notations

In this article, we consider an undirected temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ without self-loops, where V and $\mathcal{E}$ are the set of vertices (nodes) and temporal edges, respectively. Each temporal edge $(u, v, [t_s, t_e]) \in \mathcal{E}$ denotes that u and v have interactions during the time interval $[t_s, t_e]$, in which $[t_s, t_e]$ consists of a set of continuous timestamps. Note that two temporal edges $(u, v, [t_{s_1}, t_{e_1}])$ and $(u, v, [t_{s_2}, t_{e_2}])$ are different if $[t_{s_1}, t_{e_1}] \neq [t_{s_2}, t_{e_2}]$. Let $n = |V|$ and $m = |\mathcal{E}|$ be the number of nodes and temporal edges of $\mathcal{G}$, respectively. $\mathcal{T} = \{t|(u, v, [t_s, t_e]) \in \mathcal{E}, t \in [t_s, t_e]\}$ is the time domain of $\mathcal{G}$. $||\mathcal{T}||$ denotes the total number of timestamps in $\mathcal{T}$. For convenience, we assume that $\mathcal{T}$ is a set of continuous timestamps and let $\mathcal{T} = \{0, 1, 2, \ldots, ||\mathcal{T}|| - 1\}$.

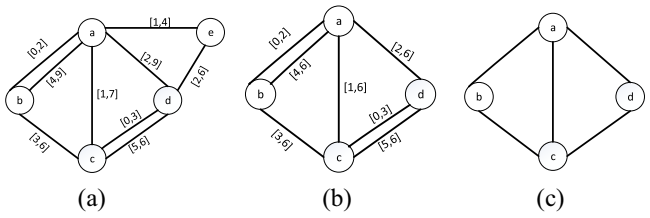


Fig. 2. (a) Temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$. (b) Temporal subgraph $\mathcal{G}_{\{a,b,c,d\}}([0, 6])$. (c) De-temporal graph of $\mathcal{G}_{\{a,b,c,d\}}([0, 6])$.

The de-temporal graph of $\mathcal{G}$ is denoted by $G = (V, E)$, where $E = \{(u, v)|\exists(u, v, [t_s, t_e]) \in \mathcal{E}\}$ and let $\bar{m} = |E|$. Namely, G is a static graph that removes the time information of $\mathcal{G}$. Let $N_u(V) = \{v \in V|(u, v) \in E\}$ be a set of neighbor nodes of u in V. A temporal graph $\mathcal{G}$ is a connected temporal graph if its de-temporal graph G is connected.

Definition 1 (Temporal Subgraph): For a vertex set H, we let $\mathcal{G}_H = (H, \mathcal{E}_H, \mathcal{T}_H)$ be a temporal subgraph of $\mathcal{G}$ induced by H, where $\mathcal{E}_H = \{(u, v, [t_s, t_e]) \in \mathcal{E}|u, v \in H\}$ and $\mathcal{T}_H = \{t|(u, v, [t_s, t_e]) \in \mathcal{E}_H, t \in [t_s, t_e]\}$. More generally, we denote the temporal subgraph $\mathcal{G}_H(T) = (H, \mathcal{E}_H(T), T)$ induced by the pair (H, T), where T is a time interval and $\mathcal{E}_H(T) = \{(u, v, [t_{s'}, t_{e'}])|(u, v, [t_s, t_e]) \in \mathcal{E}_H, [t_{s'}, t_{e'}] = [t_s, t_e] \cap T\}$.

Definition 2 (Average Degree): Given a temporal subgraph $\mathcal{G}_H(T)$, the accumulated degree of a vertex u in $\mathcal{G}_H(T)$ is defined as $d_u^T(H) = \sum_{(u,v,[t_s,t_e]) \in \mathcal{E}_H(T)} ||[t_s, t_e]||$. Furthermore, we let $a_u^T(H) = ([d_u^T(H)]/||T||)$ be the average degree of u in $\mathcal{G}_H(T)$.

Intuitively, the higher the average degree of a node, the more frequently it interacts in the temporal graph. So, the average degree can be used to measure the temporal cohesiveness. For simplicity, we use vertex set H to represent the corresponding temporal subgraph $\mathcal{G}_H$ if the context is clear. For simplicity, we concentrate some frequently used symbols in TableI.

Example 1: Fig. 2(a) shows temporal graph $\mathcal{G}$ with five vertices, nine temporal edges, and $\mathcal{T} = \{0, 1, 2, \ldots, 9\}$. Fig. 2(b) represents the temporal subgraph of $\mathcal{G}$ induced by the pair $(\{a, b, c, d\}, [0, 6])$. The de-temporal graph of Fig. 2(b) is illustrated in Fig. 2(c). For vertex c, $d_c^{[0,6]}(\{a, b, c, d\}) = ||[3, 6]|| + ||[0, 3]|| + ||[5, 6]|| + ||[1, 6]|| = 4 + 4 + 2 + 6 = 16$ and $a_c^{[0,6]}(\{a, b, c, d\}) = (16/7)$. Clearly, the temporal subgraph $\mathcal{G}_{\{a,b,c,d\}}[0, 6]$ is a connected temporal graph.

B. Problem and Complexity

One well-known cohesive subgraph model is γ-quasi-clique, which has been widely applied in community search, spam link detection, and regulatory motifs discovering [12], [24], [25]. Based on this, we also define a new temporal cohesive subgraph model as follows.

Definition 3 (γ-Quasi-Clique [12]): Given a de-temporal graph $G = (V, E)$ and a parameter γ, G is a γ-quasi-clique if $|N_u(V)| \geq \gamma \cdot (|V| - 1)$ for any $u \in V$ holds.

TABLE I
SUMMARY OF NOTATIONS

Symbol	Definition
$\mathcal{G}$	an undirected temporal graph without self-loops
V, $\mathcal{E}$, $\mathcal{T}$	the node set, the temporal edge set and time domain of $\mathcal{G}$
n, m, $\|\|\mathcal{T}\|\|$	the number of nodes, temporal edges and timestamps of $\mathcal{G}$
G, E	the de-temporal graph of $\mathcal{G}$, the edge set of G
$N_u(V)$	neighbor nodes of u in V
$\mathcal{G}_H$, $\mathcal{G}_H(T)$	the temporal subgraph of $\mathcal{G}$
$d_u^T(H)$, $a_u^T(H)$	the accumulated degree and average degree of u in $\mathcal{G}_H(T)$
δ, γ, ρ	the parameters used in our model
$co(X)$	the number of timestamps of interval set X
$\mathcal{MI}_\delta^\gamma(H)$	the maximal dense interval set of H
$CS(\delta, \gamma, H)$	the community stability of H
$(\mathcal{G}_U, S)$	a mining task where S is the chosen nodes of $\mathcal{G}_U$
$\mathcal{MCDI}(u)$	maximal candidate dense intervals of u

Definition 4 [(δ, γ)-Quasi-Clique]: Given two parameters δ and γ, the temporal subgraph $\mathcal{G}_H(T)$ is a (δ, γ)-quasi-clique if $a_u^T(H) \geq \gamma(|H| - 1)$ for any $u \in H$ holds and $|H| \geq \delta$. T is called a dense interval of H and is maximal if there is no (δ, γ)-quasi-clique $\mathcal{G}_H(T')$ such that $T \subseteq T'$.

Generally, the nodes in a temporal cohesive subgraph should have high average degrees within the subgraph and the average degrees should be related to the number of nodes of the subgraph. Thereby, by Definition 4, we know that the (δ, γ)-quasi-clique is able to capture the above intuition very well. Additionally, for a real-life temporal graph, many (δ, γ)-quasi-cliques are small in size and may not be of interest to users (see Section VI). Thus, we focus mainly on extracting (δ, γ)-quasi-cliques with size is no less than δ as defined in Definition 4. Since γ is a parameter that used to measure the compactness of a graph, we might be interested in a compact graph with $\gamma \geq 1/2$. Such a setup has been commonly used [25], [38], so we also set $\gamma \geq 1/2$ in our paper.

Example 2: Reconsider the temporal graph $\mathcal{G}$ in Fig. 2(a). Let $\delta = 3$ and $\gamma = 0.8$, and we can claim that the subgraph $\mathcal{G}_{\{a,b,c\}}([2, 6])$ is a $(3, 0.8)$-quasi-clique of $\mathcal{G}$ by Definition 4. The reason is as follows. $d_a^{[2,6]}(\{a, b, c\}) = ||[2, 2]|| + ||[4, 6]|| + ||[2, 6]|| = 9$, $a_a^{[2,6]}(\{a, b, c\}) = ([d_a^{[2,6]}(\{a, b, c\})]/|||[2, 6]||) = (9/5) \geq 0.8 \cdot (3 - 1)$, $a_b^{[2,6]}(\{a, b, c\}) = (8/5) \geq 0.8 \cdot (3 - 1)$, and $a_c^{[2,6]}(\{a, b, c\}) = (9/5) \geq 0.8 \cdot (3 - 1)$. Moreover, we can show the interval $[2, 6]$ is a maximal dense interval for $\{a, b, c\}$. In a similar way, we can derive that $\mathcal{G}_{\{a,b,c\}}([3, 7])$, $\mathcal{G}_{\{a,c,d\}}([1, 6])$, $\mathcal{G}_{\{a,c,d\}}([2, 7])$, $\mathcal{G}_{\{a,d,e\}}([1, 5])$, and $\mathcal{G}_{\{a,d,e\}}([2, 7])$ are also $(3, 0.8)$-quasi-cliques of $\mathcal{G}$.

There may exist many overlapping maximal dense intervals of H (e.g., in Example 2, there are two maximal dense intervals $[2, 6]$ and $[3, 7]$ for $\{a, b, c\}$). Given an interval set $X = \{T_1, T_2, \ldots, T_x\}$, we let co($X$) be the total number of timestamps of X, which does not contain repeated timestamps. Similar to the support of the frequent subgraph pattern [39], we define the following community stability.

Definition 5 (Community Stability): Given a vertex subset $H \subseteq V$ and two parameters δ and γ, let $\mathcal{MI}_\delta^\gamma(H)$ be the set of all maximal dense intervals of H. Then, the community stability of H, denoted by $CS(\delta, \gamma, H)$, is defined as $CS(\delta, \gamma, H) = \text{co}(\mathcal{MI}_\delta^\gamma(H))/||\mathcal{T}||$.

Based on Definition 5, we define the following model to capture stable cohesive subgraphs.

Definition 6 [ρ-Stable (δ, γ)-Quasi-Clique]: Given a temporal graph $\mathcal{G}$ and three parameters δ, γ, and ρ, a ρ-stable (δ, γ)-quasi-clique of $\mathcal{G}$ is a temporal subgraph $\mathcal{G}_H$ such that $\mathcal{G}_H$ is a connected temporal graph and $CS(\delta, \gamma, H) \geq \rho$.

In retrospect, we say that a subgraph is frequent in a collection of graphs if it appears at least f times the number of graphs [39] (i.e., its support is no less than a certain threshold). Intuitively, a temporal subgraph is more stable if its temporal cohesiveness occurs frequently. Thus, by Definition 6, we can know that a temporal subgraph is stable if it is a connected temporal graph and its community stability is no less than ρ, resulting in that ρ-stable (δ, γ)-quasi-clique is reasonable to characterize stable cohesive subgraph in temporal networks. In addition, it has the following superiorities: 1) it has many elegant computational properties to facilitate the algorithm design of the following problem, which are discussed in Sections IV and V and 2) adopting the model indeed can effectively identify some stable temporal patterns that cannot be found by the existing models as illustrated in our experimental section.

Example 3: Reconsider the temporal graph $\mathcal{G}$ in Fig. 2(a). Let $\delta = 3$, $\gamma = 0.8$, and $\rho = 0.7$, and we can derive that the temporal subgraph $\mathcal{G}_{\{a,c,d\}}$ is a 0.7-stable $(3, 0.8)$-quasi-clique of $\mathcal{G}$ by Definition 6. The reasons are as follows. First, the temporal subgraph $\mathcal{G}_{\{a,c,d\}}$ is a connected temporal graph. Second, there are two maximal dense intervals $[1, 6]$ and $[2, 7]$ for vertex set $\{a, c, d\}$ (see Example 2), thus $CS(3, 0.8, \{a, c, d\}) = ([\text{co}(\{[1, 6], [2, 7]\})]/(|||[0, 9]||)) = 0.7 \geq \rho$. In a similar way, we can obtain that the temporal subgraph $\mathcal{G}_{\{a,d,e\}}$ is also a 0.7-stable $(3, 0.8)$-quasi-clique of $\mathcal{G}$. The temporal subgraph $\mathcal{G}_{\{a,b,c\}}$ is not a 0.7-stable $(3, 0.8)$-quasi-clique of $\mathcal{G}$, because $CS(3, 0.8, \{a, b, c\}) = ([\text{co}(\{[2, 6], [3, 7]\})]/(|||[0, 9]||)) = 0.6 \leq \rho$, which contradicts with Definition 6.

Definition 7 [Maximal ρ-Stable (δ, γ)-Quasi-Clique]: A ρ-stable (δ, γ)-quasi-clique $\mathcal{G}_{\hat{H}}$ is maximal if there is no other ρ-stable (δ, γ)-quasi-clique $\mathcal{G}_H$ such that $\hat{H} \subseteq H$.

Problem Statement (MSQC): Given a temporal graph $\mathcal{G}$ and three parameters δ, γ, and ρ, our problem is to extract all maximal ρ-stable (δ, γ)-quasi-cliques from $\mathcal{G}$.

Theorem 1 (Hardness): Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ and three parameters δ, γ, and ρ, the problem of extracting *MSQC* from $\mathcal{G}$ is NP-hard.

Proof: We prove this theorem by reducing the maximal γ-quasi-clique to a special case of *MSQC*. To be specific, we consider a special temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$, where $G = \mathcal{G}_V([1, 1]) = \mathcal{G}_V([2, 2]) = \cdots = \mathcal{G}_V([||\mathcal{T}||, ||\mathcal{T}||])$. Clearly, for any $H \subseteq V$ and interval T, we have $a_u^T(H) = d_u^t(H)$ for any timestamp $t \in T$. Furthermore, we can obtain $\mathcal{MI}_\delta^\gamma(H) = \mathcal{T}$ if the cohesive subgraph H is a γ-quasi-clique in the static graph G. Consequently, mining the *MSQC* from the temporal graph $\mathcal{G}$ is equivalent to find the maximal γ-quasi-clique in G for any δ and ρ. Since the maximal γ-quasi-clique problem is NP-hard [13], our problem is also NP-hard. ■

Challenges: Although there is a close connection between our model and γ-quasi-clique [12], the existing solutions of γ-quasi-clique cannot be adopted to solve our problem.

Because (δ, γ)-quasi-clique requires that the γ-quasi-clique condition is satisfied "on average," resulting in that the solution of γ-quasi-clique is not necessarily a ρ-stable (δ, γ)-quasi-cliques by our definition. To solve our problem, a naive approach is to enumerate all (δ, γ)-quasi-cliques in each possible de-temporal graph, then aggregates all maximal dense intervals of each resulting vertex set to check which of them is a maximal ρ-stable (δ, γ)-quasi-clique. However, such an approach has the following two defects: 1) the total number of time intervals is $[||\mathcal{T}|| \cdot (||\mathcal{T}|| - 1)]/2$; thus, the total number of de-temporal graphs is $O(||\mathcal{T}||^2)$ and 2) identifying all γ-quasi-cliques in each de-temporal graph is NP-hard [12]; thereby, it is very costly to check whether a subgraph is a maximal ρ-stable (δ, γ)-quasi-clique. Thus, the challenge of our problem is how to devise efficient algorithms to prune the huge search space. In the next sections, we will devise a powerful temporal graph reduction technique, as well as an efficient enumeration algorithm with four effective pruning techniques to solve the *MSQC* problem.

IV. Temporal Graph Reduction Algorithm

It is very expensive to calculate all maximal ρ-stable (δ, γ)-quasi-cliques on original temporal graphs, because the problem is NP-hard. Therefore, we present an effective and nontrivial temporal graph reduction algorithm, called *TGRA*, to significantly reduce the original temporal graph without losing any maximal ρ-stable (δ, γ)-quasi-clique.

A. Critical Idea of Temporal Graph Reduction Algorithm

For convenience, we define a mining task as $(\mathcal{G}_U, S)$, where $S \subseteq U$ has been selected as members of any ρ-stable (δ, γ)-quasi-clique of $\mathcal{G}_U$. The task aims to find all ρ-stable (δ, γ)-quasi-cliques H from $\mathcal{G}_U$ such that $S \subseteq H$. Thus, our maximal ρ-stable (δ, γ)-quasi-cliques problem indicates the mining task $(\mathcal{G}, \emptyset)$. Let $lb(U, S)$ be the lower bound of ρ-stable (δ, γ)-quasi-clique in $(\mathcal{G}_U, S)$ and we will discuss the bound in Section V-B.

Definition 8 (Maximal Candidate Dense Interval): Given mining task $(\mathcal{G}_U, S)$, interval T is a candidate dense interval of u w.r.t. $(\mathcal{G}_U, S)$ iff $a_u^T(U) \geq \gamma \cdot (\max\{\delta, |S|, lb(U, S)\} - 1)$. T is further called a maximal candidate dense interval of u w.r.t. $(\mathcal{G}_U, S)$ if $\nexists T' \supseteq T$ such that $a_u^{T'}(U) \geq \gamma \cdot (\max\{\delta, |S|, lb(U, S)\} - 1)$.

Given an interval set $\widetilde{\mathcal{T}}$, we let $\mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$ be all maximal candidate dense intervals of u w.r.t. $(\mathcal{G}_U, S)$ in $\widetilde{\mathcal{T}}$. Namely, $\forall T \in \mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$, T is a maximal candidate dense interval of u w.r.t. $(\mathcal{G}_U, S)$ and $\exists I \in \widetilde{\mathcal{T}}$ such that $T \subseteq I$. We use $\mathcal{MCDI}(u)$ to represent $\mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$ if the context is clear. We split the time domain $\mathcal{T}_U$ into an interval set denoted by $IS(\mathcal{T}_U)$ (e.g., $IS(\{1, 2, 3, 6, 7, 8, 11, 12\}) = \{[1, 3], [6, 8], [11, 12]\}$). We next introduce an important lemma to develop the temporal graph reduction algorithm. For simplicity, we put all proofs of lemmas in the Appendix.

Lemma 1 (Stability-Based Pruning): We can prune $u \in U$ from task $(\mathcal{G}_U, S)$ without losing any maximal ρ-stable (δ, γ)-quasi-clique if $\text{co}(\mathcal{MCDI}(u, U, IS(\mathcal{T}_U), S)) < \rho \cdot ||\mathcal{T}||$.

Algorithm 1: Compute_MCDI$(u, (\mathcal{G}_U, S), \widetilde{\mathcal{T}}, \delta, \gamma)$

Input: A vertex u, a mining task $(\mathcal{G}_U, S)$, an interval set $\widetilde{\mathcal{T}}$ and parameters δ, γ
Output: The interval set $\mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$

```
1  T̃ ≜ {T_1, T_2, ..., T_l}, I ← ∅
2  for T_i ∈ T̃ do
3      d_t ← d_u^t(U) − γ · (max{δ, |S|, lb(U, S)} − 1) for all t ∈ T_i
4      S_1 ← d_{T_i.start}
5      for j = 2 to T_i.end − T_i.start + 1 do
6          S_j = S_{j−1} + d_{j+T_i.start−1}
7      position ← ∞, I_i ← ∅
8      for π(S_j) = 2 to T_i.end − T_i.start + 1 do
9          if π^−(π(S_j) − 1) < position then
10             position = π^−(π(S_j) − 1)
11         if S_j − S_position + d_{position+T_i.start−1} ≥ 0 then
12             I_i = I_i ∪ {[position + T_i.start − 1, j + T_i.start − 1]}
           else
13             I_i = I_i ∪ {[position + T_i.start, j + T_i.start − 1]}
14     I = I ∪ I_i
15 {[a_1, b_1], [a_2, b_2], ..., [a_k, b_k]} ← Sort(I)
16 I ← {[a_1, b_1]}, x = b_1
17 for j=2 to k do
18     if b_j > x then
19         I ∪ {[a_j, b_j]}
20         x = b_j
21 return I
```

There are two challenges in implementing the stability-based pruning: one is how to calculate $\mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$ effectively. A naive approach is to search all possible subintervals of $I \in \widetilde{\mathcal{T}}$ for completing the maximality check of Definition 8. Consequently, the naive approach is intractable. After pruning all the unqualified vertices of U by Lemma 1, the candidate dense intervals of the rest vertices may change, resulting in that the pruned vertices may trigger a "butterfly effect" (i.e., pruning vertices may cause qualified vertices become unqualified). Thus, the second challenge is how to efficiently update the candidate dense intervals of vertices and then iteratively prune those unqualified vertices until no vertex can be pruned by Lemma 1. The following two sections will discuss how to overcome the two challenges.

B. Identifying $\mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$ by Dynamic Programming

According to Definition 8, we have $a_u^T(U) \geq \gamma \cdot (\max\{\delta, |S|, lb(U, S)\} - 1)$ for any $T \in \mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$. Clearly, $a_u^T(U) \geq \gamma \cdot (\max\{\delta, |S|, lb(U, S)\} - 1)$ is equivalent to $\sum_{t \in T}(d_u^t(U) - \gamma \cdot (\max\{\delta, |S|, lb(U, S)\} - 1)) \geq 0$. Considering this, we propose a novel dynamic programming algorithm (Algorithm 1) to identify efficiently $\mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$, which transforms the problem into identifying all maximal intervals such that the sum of the degree of u (after being updated by line 3 in Algorithm 1) in each interval is not less than 0. Before proceeding further, we present some useful symbols: given a degree sequence $(d_1, d_2, \ldots, d_T)$, we let $S_j = \sum_{i=1}^{j} d_i$ be the prefix degree sum and π be a permutation that sorts these prefix sums in increasing order and $\pi(S_j)$ is the position of S_j in the permutation. $\pi^-(S_j)$ is the position of S_j in the prefix sum sequence (i.e., $\pi^-(S_j) = j$). $T.start$ and $T.end$ are the starting time and the ending time of T, respectively.

Algorithm 2: *TGRA*($\mathcal{G}$, δ, γ, ρ)

Input: Temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ and three parameters δ, γ and ρ
Output: The reduced graph $\mathcal{G}_{\widetilde{V}}$
1 $\mathcal{Q} \leftarrow \emptyset$, $mark(u) = True$ for all $u \in V$
2 **for** $u \in V$ **do**
3 $\quad \mathcal{I}(u) \leftarrow$ Compute_MCDI(u, $(\mathcal{G}, \emptyset)$, $IS(\mathcal{T})$, δ, γ)
4 $\quad$ **if** co($\mathcal{I}(u)$) $< \rho \cdot ||\mathcal{T}||$ **then**
5 $\quad\quad \mathcal{Q}.push(u)$ and $mark(u) = False$ {/* stability-based */}
Update
6 **while** $\mathcal{Q} \neq \emptyset$ **do**
7 $\quad u \leftarrow \mathcal{Q}.pop()$
8 $\quad$ **for** $v \in N_u(V)$ *and* $mark(v) = True$ **do**
9 $\quad\quad$ update $\mathcal{I}(v)$ by Lemma 2
10 $\quad\quad$ **if** co($\mathcal{I}(v)$) $< \rho \cdot ||\mathcal{T}||$ **then**
11 $\quad\quad\quad \mathcal{Q}.push(v)$ and $mark(v) = False$
12 $\widetilde{V} \leftarrow \{v | mark(v) = True\}$
13 return $\mathcal{G}_{\widetilde{V}}$

Definition 9 (Maximal j-Truncated Candidate Dense Interval): Given a truncated timestamp j, we let the maximal j-truncated candidate dense interval be an interval $[tru(j), j]$ such that $S_j - S_{tru(j)-1} \geq 0$ and $tru(j)$ is minimized.

By Definition 9, we can know that $\mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$ is included in all these maximal j-truncated candidate dense intervals, where j is a timestamp in $\widetilde{\mathcal{T}}$. Our dynamic programming algorithm computes all maximal j-truncated candidate dense intervals in a recursive way.

Algorithm 1 first initializes result set $\mathcal{I}$ as $\emptyset$ (line 1). For finding all maximal j-truncated candidate dense intervals, it computes the prefix sum S_j (lines 4–6). In lines 7–13, the algorithm recursively finds the maximal $(j + T_i.start - 1)$-truncated candidate dense interval if any. In line 14, $\mathcal{I}$ represents all maximal j-truncated candidate dense intervals, where j is a timestamp in $\widetilde{\mathcal{T}}$ if any. Since these intervals may contain each other, the algorithm merges these intervals to get the $\mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$ in lines 15–20. To be specific, the algorithm first sorts the above all maximal intervals in increasing order by the starting time of each interval. Let $\{[a_1, b_1], [a_2, b_2], \ldots, [a_k, b_k]\}$ be the sorted interval set (line 15). Note that it only takes the interval with the maximum ending time if there are several intervals that have the same starting time. By doing so, $a_i \neq a_j$ and $b_i \neq b_j$ if $i \neq j$. Then, the algorithm merges these intervals for identifying all maximal candidate dense intervals $\mathcal{MCDI}(u, U, \widetilde{\mathcal{T}}, S)$ (lines 16–20). The following example illustrates the procedure.

Example 4: Without loss of generality, we assume $l = 1$ and T_1.start $= 1$. There is a sequence of numbers: 13, −3, −25, 20, −3, −16, −23, 18, 20, −7, 12, −5, −22, 15, −4, 7, where each number represents the updated degree of u in line 3 of Algorithm 1. In lines 4–6, the algorithm computes the prefix sum as follows: 13, 10, −15, 5, 2, −14, −37, −19, 1, −6, 6, 1, −21, −6, −10, −3. Then, it sorts these prefix sums in ascending order and obtains the maximal candidate dense interval ending with any timestamp by running lined 8–14 of Algorithm 1. These intervals are listed as follows: [8, 13], [4, 15], [4, 10], [4, 14], [4, 16], [4, 9], [4, 12], [4, 11]. Subsequently, the algorithm merges these intervals for the final result by running lines 15–20. So, the maximal candidate dense interval of u is [4, 16].

Theorem 2: The time complexity of Algorithm 1 is $O(\sum_i^l(||T_i||))$, where $\widetilde{\mathcal{T}} = \{T_1, T_2, \ldots, T_l\}$.

Proof: $\forall T_i$, the algorithm first takes $O(||T_i||)$ time to update the degree of u (line 3), and $O(||T_i||)$ time to get the prefix sums (lines 4–6). Then, it takes $O(||T_i||)$ time to generate the maximal candidate dense interval ending with any timestamp $j + T_i.start - 1$. Note that it takes $O(||T_i||)$ time to sort S_j by utilizing the counting sort method (line 8). Thus, the algorithm takes $\sum_i^l O(||T_i||)$ in lines 2–14. In line 15, the algorithm takes $O(\text{co}(\mathcal{I}))$ time to sort $\mathcal{I}$ by utilizing the counting sort method. In lines 17–20, the algorithm takes $O(k)$ time to merge intervals, where $k \leq \text{co}(\mathcal{I})$. Therefore, the algorithm takes $O(\text{co}(\mathcal{I}))$ time in lines 15–20. Clearly, $O(\text{co}(\mathcal{I}))$ is bounded by $\sum_i^l O(||T_i||)$. Putting these together, Algorithm 1 takes $\sum_i^l O(||T_i||)$ in total. ■

C. Incrementally Update Maximal Candidate Dense Intervals

For convenience, we denote $\mathcal{T}_U(u, v)$ as the interval set of u and v connections in $\mathcal{G}_U$. Namely, $\mathcal{T}_U(u, v) = \{[t_s, t_e] | (u, v, [t_s, t_e]) \in \mathcal{E}_U\}$. Observed that we only need to update u's neighbors' partial maximal candidate dense intervals when vertex u is deleted. Considering this, we efficiently implement the update process, which can avoid some redundant computations. Below, we summarize the update process into the following lemma.

Lemma 2: After deleting vertex u from mining task $(\mathcal{G}_U, S)$, we have $\mathcal{MCDI}(v, U\backslash\{u\}, IS(\mathcal{T}_{U\backslash\{u\}}), S) = \mathcal{T}_b \cup \mathcal{MCDI}(v, U\backslash\{u\}, \mathcal{T}_a, S)$ for any $v \in N_u(U)$, where $\mathcal{T}_a \subseteq \mathcal{MCDI}(v, U, IS(\mathcal{T}_U), S)$. $\forall T_i \in \mathcal{T}_a$, $\exists [t_s, t_e] \in \mathcal{T}_U(u, v)$ such that $T_i \cap [t_s, t_e] \neq \emptyset$. $\mathcal{T}_b = \mathcal{MCDI}(v, U, IS(\mathcal{T}_U), S)\backslash\mathcal{T}_a$.

Note that the worst case for this update process is $\mathcal{T}_b = \emptyset$, i.e., all maximal candidate dense intervals of v are affected, resulting in that these maximal candidate dense intervals need to be calculated from scratch.

D. Temporal Graph Reduction Algorithm

The temporal graph reduction algorithm *TGRA* is outlined in Algorithm 2 that uses the stability-based pruning to reduce the original temporal graph. Note that in the original temporal graph, we have $lb(V, \emptyset) = \delta$ (see Section V-B). Concretely, given temporal graph $\mathcal{G}$ and three parameters δ, γ, and ρ as inputs, the algorithm outputs $\mathcal{G}_{\widetilde{V}}$ as the reduced graph. The algorithm first initializes queue $\mathcal{Q}$ as $\emptyset$ to maintain all vertices that need to be deleted and marks vertex u as true that represents u has not been deleted yet (line 1). Then, the algorithm executes the stability-based pruning to prune some unqualified vertices (lines 2–5). Subsequently, it executes the update process (Lemma 2) when the unqualified vertices are deleted (lines 6–11). Finally, the algorithm returns the reduced graph $\mathcal{G}_{\widetilde{V}}$ (lines 12 and 13). Clearly, *TGRA* can guarantee this process correctness due to the previous analysis.

Theorem 3: The time complexity of Algorithm 2 is $O((n + \bar{m}||\mathcal{T}||)||\mathcal{T}||)$, respectively.

Proof: By Theorem 2, we know that the time spent on computing maximal candidate dense intervals $\mathcal{I}(u)$ is $O(||\mathcal{T}||)$ for any $u \in V$. Thus, Algorithm 2 takes $O(n||\mathcal{T}||)$ time in lines 2–5. In lines 6–11, the algorithm executes the update

process of maximal candidate dense intervals. Specifically, the algorithm traverses each edge of the de-temporal graph G of $\mathcal{G}$ at most two times in the main loops (except for applying Lemma 2 to update the maximal candidate dense intervals). When the algorithm traverses an edge (u, v) (line 8), it applies Lemma 2 to update the $\mathcal{I}(v)$. The update process takes $\sum_i^l O(||T_i||)$ in the worst case by Algorithm 1, in which $\mathcal{I}(v) = \{T_1, T_2, \ldots, T_l\}$. Clearly, $\sum_i^l O(||T_i||)$ is bounded by $||\mathcal{T}||^2$. Thus, the algorithm takes $O(\bar{m}||\mathcal{T}||^2)$ to complete the update in lines 6–11, where $\bar{m}$ is the number of edges in de-temporal graph G of $\mathcal{G}$. Consequently, Algorithm 2 takes $O((n + \bar{m}||\mathcal{T}||)||\mathcal{T}||)$ in total. ■

V. Enumeration All Maximal ρ-Stable (δ, γ)-Quasi-Cliques

In this section, we first propose an algorithm to compute the community stability of a vertex set. Then, four powerful pruning rules are proposed. Finally, we devise an enumeration algorithm to detect all maximal ρ-stable (δ, γ)-quasi-cliques.

A. Community Stability Computation

Recall that a temporal subgraph is a ρ-stable (δ, γ)-quasi-clique if it is a connected temporal graph and its community stability is at least ρ. In this section, we design a filtering and verification algorithm to compute the community stability of C, which is sketched in Algorithm 3. Intuitively, the algorithm filters out the invalid time intervals layer by layer until the community stability is completely calculated. For convenience, we define an "intersection" operation $\bigotimes$ for two interval sets as follows: let $\mathcal{T}_k = \{[t_{s_1}, t_{e_1}], \ldots, [t_{s_k}, t_{e_k}]\}$ and $\mathcal{T}_h = \{[t_{l_1}, t_{r_1}], \ldots, [t_{l_h}, t_{r_h}]\}$, and we define $\mathcal{T}_k \bigotimes \mathcal{T}_h = \{T_1, T_2, \ldots, T_w\}$, where $T_i \subseteq [t_{s_j}, t_{e_j}]$ and $T_i \subseteq [t_{l_p}, t_{r_p}]$ for all i and some j and p, such as $\{[1, 4], [6, 10], [12, 14]\} \bigotimes \{[2, 7], [11, 15]\} = \{[2, 4], [6, 7], [12, 14]\}$.

Given vertex set C and parameters δ and γ as inputs, Algorithm 3 outputs the community stability of C. As long as $\mathcal{T}_1 \neq \emptyset$, Algorithm 3 calls Algorithm 1 to compute the maximal candidate dense intervals of each vertex in C w.r.t. $(\mathcal{G}_C, C)$ in the interval set $\mathcal{T}_1$ and updates $\mathcal{T}_2$ by "intersecting" these interval sets to filter out invalid intervals (line 3). $\mathcal{T}_1 \cap \mathcal{T}_2$ is the maximal dense intervals of C (Theorem 4); thus, the algorithm updates $\mathcal{I}$ (line 4). To avoid redundant computation, the algorithm subtracts $\mathcal{T}_1$ from $\mathcal{T}_2$ as the update of $\mathcal{T}_1$ for the next iteration (line 5).

Theorem 4: Algorithm 3 can correctly compute the community stability of C, and its time complexity is $O(|C| \cdot ||\mathcal{T}_C||^2)$.

Proof: For correctness, $\forall T \notin \bigotimes_{u \in C} \mathcal{MCDI}(u, C, \mathcal{T}_1, C)$, $\mathcal{G}_C(T)$ is not a (δ, γ)-quasi-clique (Definition 4). Thus, we can safely filter out the invalid intervals (line 3). Since all maximal dense intervals of C fall into $\bigotimes_{u \in C} \mathcal{MCDI}(u, C, \mathcal{T}_1, C)$, Algorithm 3 further iteratively calculates $\mathcal{I}$ from $\bigotimes_{u \in C} \mathcal{MCDI}(u, C, \mathcal{T}_1, C)$. Consequently, Algorithm 3 returns the community stability of C.

For time complexity, there are at most $||\mathcal{T}_C||$ loops. In each loop, the algorithm involves Algorithm 1 to compute $\mathcal{MCDI}(u, C, \mathcal{T}_1, C)$ for each vertex $u \in C$. Accordingly, the worse-case time complexity is $O(|C| \cdot ||\mathcal{T}_C||^2)$. ■

Algorithm 3: *CS*(δ, γ, C)

Input: Vertex set C and two parameters δ and γ
Output: The community stability of C
1 $\mathcal{I} \leftarrow \emptyset$, $\mathcal{T}_1 \leftarrow IS(\mathcal{T}_C)$, $\mathcal{T}_2 \leftarrow \emptyset$
2 **while** $\mathcal{T}_1 \neq \emptyset$ **do**
3 $\quad \mathcal{T}_2 \leftarrow \bigotimes_{u \in C}$ Compute_MCDI(u, $(\mathcal{G}_C, C)$, $\mathcal{T}_1$, δ, γ)
4 $\quad \mathcal{I} \leftarrow \mathcal{I} \cup (\mathcal{T}_1 \cap \mathcal{T}_2)$
5 $\quad \mathcal{T}_1 \leftarrow \mathcal{T}_2 \setminus \mathcal{T}_1$
6 return $\frac{\mathrm{co}(\mathcal{I})}{||\mathcal{T}||}$

B. Powerful Pruning Techniques

In this section, we further present four effective pruning techniques used in Section V-C. These pruning techniques can prune some unqualified vertices that are definitely not contained in any ρ-stable (δ, γ)-quasi-clique of mining task $(\mathcal{G}_U, S)$. For convenience, we let $dis(u, v, \mathcal{G})$ be the number of edges on the shortest path between u and v in the de-temporal graph G of $\mathcal{G}$. The diameter of $\mathcal{G}$ is defined as $D(\mathcal{G}) = \max_{u,v \in V}\{dis(u, v, \mathcal{G})\}$.

Lemma 3 (Distance-Based Pruning): Given temporal subgraph $\mathcal{G}_H(T)$ with $h = |H| \geq 2$, if $\mathcal{G}_H(T)$ is a (δ, γ)-quasi-clique, then

$$D(\mathcal{G}_H(T)) = \begin{cases} 1, & \frac{h-2}{h-1} < \gamma \leq 1 \\ \leq 2, & \frac{h-2}{2(h-1)} < \gamma \leq \frac{h-2}{h-1}. \end{cases} \quad (1)$$

From Lemma 3, we know that $D(\mathcal{G}_H(T))$ is not larger than $f(\gamma)$ if $\mathcal{G}_H(T)$ is a (δ, γ)-quasi-clique, where $f(\gamma) = 1$ when $(h - 2/h - 1) < \gamma \leq 1$, and $f(\gamma) = 2$ when $[(h-2)/(2(h-1))] < \gamma \leq [(h-2)/(h-1)]$. Recalling that $\gamma \geq (1/2)$; thus, we can derive Lemma 3 considering all cases of $D(\mathcal{G}_H(T))$ because $\lim_{h \to \infty}[(h-2)/(2(h-1))] = (1/2) \leq \gamma$. Consequently, we can prune these vertices that are farther away than $f(\gamma)$ from the selected vertex set S. Formally, for the mining task $(\mathcal{G}_U, S)$, we denote $P(U, S) = \cap_{u \in S}\{v \in U | dis(u, v, \mathcal{G}_U) \leq f(\gamma)\}$ and then we can safely delete any vertex that is not in $P(U, S)$.

Below, we design a bound-based pruning technique to prune the whole $(\mathcal{G}_U, S)$ if there does not exist any ρ-stable (δ, γ)-quasi-clique in $(\mathcal{G}_U, S)$. For convenience, we denote $A_S(U, v) = \max_{I \subseteq \mathcal{T}_U}\{a_v^I(S)\}$ and $A_S(U, S) = \sum_{v \in S} A_S(U, v)$ for vertex $v \in U$. We sort vertex $u \in U \setminus S$ in descending order of $A_S(U, u)$ and denote the sorted vertices $u_1, u_2, \ldots, u_{|U \setminus S|}$.

Lemma 4: Given task $(\mathcal{G}_U, S)$ and $S \neq \emptyset$. If $S \subseteq Y \subseteq U$ and Y is a ρ-stable (δ, γ)-quasi-clique, then $|Y| \leq ub(U, S)$, where $ub(U, S) = \max\{k | A_S(U, S) + \sum_{i=1}^k A_S(U, u_i) \geq |S|\gamma(|S| + k - 1)\} + |S|$.

Lemma 5: Given task $(\mathcal{G}_U, S)$ and $S \neq \emptyset$, if $S \subseteq Y \subseteq U$ and Y is a ρ-stable (δ, γ)-quasi-clique, then $lb(U, S) \leq |Y|$, where $lb(U, S) = \max\{lb_1(U, S), lb_2(U, S)\}$, $lb_1(U, S) = \min\{k | A_S(U, S) + \sum_{i=1}^k A_S(U, u_i) \geq |S|\gamma(|S| + k - 1)\} + |S|$, and $lb_2(U, S) = ([|S|^2 - \gamma|S| - A_S(U, S)/[|S|(1 - \gamma)])$.

Note that when $S = \emptyset$, we cannot use Lemmas 4 and 5 to compute the bounds. Thus, we let $ub(U, \emptyset) = |U|$ and $lb(U, \emptyset) = \delta$ for convenience.

Lemma 6 (Bound-Based Pruning): Given task $(\mathcal{G}_U, S)$, we can safely prune the whole task if it satisfies one of the following conditions: 1) $|U| < \delta$; 2) $ub(U,S) < lb(U,S)$; 3) $ub(U,S) < \delta$; and 4) $\delta \le ub(U,S) < |S|$.

Note that once $\delta \le ub((\mathcal{G}_U, S)) = |S|$, we return the pruned subgraph as $\mathcal{G}_S$, because there are no more vertices that can be added to S for task $(\mathcal{G}_U, S)$.

For task $(\mathcal{G}_C, S)$, we will divide $(\mathcal{G}_C, S)$ into two different subspaces by selecting vertex $v \in C \backslash S$ when C is not a ρ-stable (δ, γ)-quasi-clique (Section V-C): 1) the subspace of excluding v and 2) the subspace of including v. Below, we present the excluding-node pruning and including-node pruning for the above two subspaces, respectively.

Excluding-Node Pruning: Recall that the deletion of v may trigger a "butterfly effect," i.e., pruned v may cause its qualified neighbors become unqualified, deletion of v's neighbor may result in deletion of v's neighbor's neighbor, and so on. Thus, we can iteratively delete those unqualified vertices until no vertex can be pruned. The process is similar to *TGRA*, except that the search terminates once any vertex in S is deleted. The details of the pruning rules are described in Section V-C.

Lemma 7 (Including-Node Pruning): For task $(\mathcal{G}_C, S \cup \{v\})$, let $\mathcal{I} = \bigotimes_{w \in S \cup \{v\}} \mathcal{MCDI}(w, C, IS(\mathcal{T}_C), S \cup \{v\})$. If $\text{co}(\mathcal{I}) < \rho \cdot ||\mathcal{T}||$, we can safely prune the whole mining task. If $\text{co}(\mathcal{I} \bigotimes \mathcal{MCDI}(u, C, IS(\mathcal{T}_C), S \cup \{v, u\})) < \rho \cdot ||\mathcal{T}||$ for vertex $u \in C \setminus \{S \cup \{v\}\}$, then we prune u from $(\mathcal{G}_C, S \cup \{v\})$ without loss of accuracy.

C. BB&SCM Algorithm

BB&SCM (branch and bound & stable cohesive subgraph mining) is a tractable algorithm to mine all maximal ρ-stable (δ, γ)-quasi-cliques (Algorithm 4). *BB&SCM* includes two stages. First, it calls Algorithm 2 to reduce the original temporal graph. Subsequently, on the reduced temporal graph, we use all previous pruning techniques to devise the branch and bound algorithm (*B&B*) for enumerating all maximal ρ-stable (δ, γ)-quasi-cliques. Specifically, Algorithm 4 first initializes result set $\mathcal{R}$ as an empty set and calls Algorithm 2 to reduce significantly the original temporal graph (line 1). Then, it calls *B&B* procedure (Algorithm 5) with reduced temporal graph $\mathcal{G}_{\widetilde{V}}$, an empty set (of selected vertices), and parameters δ, γ, and ρ (line 2).

Algorithm 5 is a recursive procedure of mining task $(\mathcal{G}_U, S)$. Concretely, it first applies the distance-based pruning rule to prune some unqualified vertices and denotes $\widetilde{U}$ as the pruned vertex set (line 1). Then, it invokes the bound-based pruning to determine whether the task needs to continue (lines 2 and 3). If it is true, for each connected temporal graph C that contains S and $ub(C,S) \ge \delta$, the algorithm checks whether cohesive subgraph C is a maximal ρ-stable (δ, γ)-quasi-clique. If it is true, the algorithm updates the result set (lines 8–10). Otherwise, the algorithm randomly selects a node $v \in C \backslash S$ to branch task $(\mathcal{G}_C, S)$ into two subtasks (lines 11–34). The algorithm executes the process of excluding-node pruning (lines 12–27) and including-node pruning (lines 28–34) respectively. For excluding-node pruning, the algorithm initializes vertex set D as $\{v\}$ to collect the deleted vertices and queue $\mathcal{Q}$ to maintain all vertices that need to be deleted (line 12). In lines 13–26, the algorithm first checks which nodes of $C \setminus \{v\}$ should be deleted and pushes them into $\mathcal{Q}$. As long as $\mathcal{Q} \neq \emptyset$, it pops vertex u from $\mathcal{Q}$ and updates u's neighbors' maximal candidate dense intervals. Note that once any vertex in S is deleted, it jumps to another search subspace (lines 16 and 17 and lines 24 and 25). For including-node pruning, it directly applies Lemma 7 to prune those unqualified vertices or the whole search subspace (lines 28–34). As Section VI shows, the two pruning techniques can largely reduce the number of nodes in C, thus substantially improving the search efficiency of the two subspaces. Clearly, the above two subtasks contain all cases of task $(\mathcal{G}_C, S)$. Thus, *BB&SCM* performs an enumeration search and the algorithm derives correctly maximal ρ-stable (δ, γ)-quasi-cliques. Fig. 3 illustrates the procedure of Algorithm 4.

Complexity Analysis: Since the problem of identifying all maximal ρ-stable (δ, γ)-quasi-cliques is NP-hard, the worst

Algorithm 4: *BB&SCM*$(\mathcal{G}, \delta, \gamma, \rho)$

Input: Temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ and three parameters δ, γ and ρ
Output: The all maximal ρ-stable (δ, γ)-quasi-cliques of $\mathcal{G}$

1 $\mathcal{R} \leftarrow \emptyset$, $\mathcal{G}_{\widetilde{V}} \leftarrow TGRA(\mathcal{G}, \delta, \gamma, \rho)$
2 *B&B*$(\mathcal{G}_{\widetilde{V}}, \emptyset, \delta, \gamma, \rho)$
3 return $\mathcal{R}$

Algorithm 5: *B&B*$(\mathcal{G}_U, S, \delta, \gamma, \rho)$

1 $\widetilde{U} \leftarrow P(U,S)$ {/* distance-based pruning */}
2 **if** *at least one condition of Lemma 6 holds for* $(\mathcal{G}_{\widetilde{U}}, S)$ **then**
3 return {/* bound-based pruning */}
4 **if** $\delta \le ub(\widetilde{U}, S) = |S|$ **then**
5 $\mathcal{G}_{\widetilde{U}} \leftarrow \mathcal{G}_S$
6 $\mathcal{CT} \leftarrow$ all connected temporal graphs of $\mathcal{G}_{\widetilde{U}}$
7 **for** $C \in \mathcal{CT}$ *and* $S \subseteq C$ *and* $ub(C,S) \ge \delta$ **do**
8 **if** *there is a* $R \in \mathcal{R}$ *s.t.* $C \subseteq R$ **then**
  jump Line 7
9 **if** $CS(\delta, \gamma, C) \ge \rho$ **then**
10 $\mathcal{R} \leftarrow \mathcal{R} \cup \{C\}$
  else
11 randomly select vertex $v \in C \setminus S$
12 $D \leftarrow \{v\}$, $\mathcal{Q} \leftarrow \emptyset$ {/* excluding-node pruning */}
13 **for** $u \in C \setminus \{v\}$ **do**
14 $\mathcal{I}(u) \leftarrow$ Compute_MCDI$(u, (\mathcal{G}_{C \backslash v}, S), IS(\mathcal{T}_{C \backslash v}), \delta, \gamma)$
15 **if** $\text{co}(\mathcal{I}(u)) < \rho \cdot ||\mathcal{T}||$ **then**
16 **if** $u \in S$ **then**
17 jump Line 28
18 $\mathcal{Q}.push(u)$
19 **while** $\mathcal{Q} \neq \emptyset$ **do**
20 $u \leftarrow \mathcal{Q}.pop()$, $D \leftarrow D \cup \{u\}$
21 **for** $w \in N_u(C)$ *and* $w \notin D$ **do**
22 update $\mathcal{I}(w)$ by Lemma 2
23 **if** $\text{co}(\mathcal{I}(w)) < \rho \cdot ||\mathcal{T}||$ **then**
24 **if** $w \in S$ **then**
25 jump Line 28
26 $\mathcal{Q}.push(w)$
27 *B&B*$(\mathcal{G}_{C \backslash D}, S, \delta, \gamma, \rho)$
28 $\mathcal{I} \leftarrow \bigotimes_{w \in S \cup \{v\}} \mathcal{MCDI}(w, C, IS(\mathcal{T}_C), S \cup \{v\})$ {/* including-node pruning */}
29 **if** $\text{co}(\mathcal{I}) < \rho \cdot ||\mathcal{T}||$ **then**
  jump Line 7
30 $\mathcal{A} \leftarrow \emptyset$
31 **for** $u \in C \setminus \{S \cup \{v\}\}$ **do**
32 **if** $\text{co}(\mathcal{I} \bigotimes \mathcal{MCDI}(u, C, IS(\mathcal{T}_C), S \cup \{v, u\})) < \rho \cdot ||\mathcal{T}||$ **then**
33 $\mathcal{A}.push(u)$
34 *B&B*$(\mathcal{G}_{C \backslash \mathcal{A}}, S \cup \{v\}, \delta, \gamma, \rho)$

case time complexity of Algorithm 4 is exponential. Clearly, our *BB&SCM* is a powerful binary enumeration tree due to these pruning techniques in *B&B*. Specifically, let $\hat{n}$ and $\hat{\mathcal{T}}$ be the largest number of nodes and time domain of connected temporal graphs of the reduced graph, respectively. There are at most $O(2^{\hat{n}})$ subspaces in each connected temporal graph of the reduced graph (a subspace corresponds to a *B&B* algorithm). In each *B&B*, the algorithm takes $O(\hat{n}||\hat{\mathcal{T}}||^2)$ time to determine whether the current cohesive subgraph is a ρ-stable (δ, γ)-quasi-clique (Theorem 4). If not, the algorithm takes $O((\hat{n} + \hat{m}||\hat{\mathcal{T}}||)||\hat{\mathcal{T}}||)$ time to execute the excluding-node pruning (Theorem 3) and requires $O(\hat{n}||\hat{\mathcal{T}}||^2)$ to execute the including-node pruning, where $\hat{m}$ is the largest number of edges in connected subgraphs of G. Therefore, the time complexity of *BB&SCM* is $O(n_r \cdot 2^{\hat{n}}(\hat{n} + \hat{m})||\hat{\mathcal{T}}||^2 + (n + \bar{m}||\mathcal{T}||)||\mathcal{T}||)$, where $\bar{m} = |E|$ and n_r is the number of connected temporal graphs of the reduced temporal graph. Since $\hat{n}$ is usually not very large (Section IV-D) and the pruning techniques in *B&B* are very effective, the algorithm can deal with large-scale temporal graphs.

Greedy Vertex Selection: Recall that line 11 of Algorithm 5 randomly selects vertex v to branch task $(\mathcal{G}_C, S)$ into two subtasks. However, the approach may be ineffective because it may select a bad vertex that slows the pruning process. Here, we propose a heuristic approach to evaluate the quality of v. Specifically, the degree of v w.r.t. C in the time domain $\mathcal{T}_C$ should be as large as possible, the reason is that $C \setminus \{v\}$ may be more sparse such that $\mathcal{I}(u)$ in line 14 of Algorithm 5 could be even smaller, resulting in that excluding-node pruning can prune more vertices. On the other hand, the maximal candidate dense intervals of v w.r.t. $(\mathcal{G}_C, S)$ in interval set $IS(\mathcal{T}_C)$ should be as small as possible, because $\mathcal{I}$ in line 28 of Algorithm 5 could be even smaller, resulting in that including-node pruning can prune more vertices by Lemma 7. Based on these intuitions, we design function Q to evaluate the quality of v as follows: $Q(v, C, S) = d_v^{\mathcal{T}_C}(C) - \text{co}(\mathcal{MCDI}(v, C, IS(\mathcal{T}_C), S))$. Thus, we select vertex $v \in C \setminus S$ that has the largest $Q(v, C, S)$.

VI. Experimental Evaluation

Here, we evaluate the efficiency, scalability, and effectiveness of our approaches. All algorithms are implemented in C++ and compiled using g++ compiler at −O2 optimization level. All experiments are conducted on a server with a 2.4-GHz Xeon CPU and 256-GB memory running CentOS 6.1.

A. Experimental Setting

Datasets: Seven real-world temporal datasets are used in our experiments. CollegeMsg (Msg) is a private message network at the University of California, Irvine, in which edge (u, v, t) means that user u sent a message to user v at time t. Chess is a game network in which temporal edge (u, v, t) records players u and v played a chess game at time t. Facebook wall posts (Facebook) is the wall posts between users in Facebook. Linux and Enron are temporal communication networks, where the temporal edge (u, v, t) denotes that u sent v an email at time t. Edge (u, v, t) of Wikipedia simple-En (Wiki) indicates that article u and v were connected by a hyperlink at time t. DBLP is a collaboration network, where edge (u, v, t) represents author u and v coauthored a publication at time t. In our experiments, the self-loops are deleted and directed temporal graphs are converted into undirected temporal graphs. We adopt a similar method as used in [2] to transform these (u, v, t) into our interval form. Namely, temporal edge $(u, v, [t_s, t_e])$ of our model means that u and v have an interaction at time t for any $t \in [t_s, t_e]$. The statistics of datasets is shown in Table II. All datasets are downloaded from http://konect.unikoblenz.de/, except that CollegeMsg is downloaded from http://snap.stanford.edu/.

TABLE II
STATISTICS OF DATASETS. d_{max} IS THE MAXIMUM NUMBER OF TEMPORAL EDGES ASSOCIATED WITH A VERTEX. TS IS THE TIME UNIT FOR EACH SNAPSHOT

Dataset	$n = \|V\|$	$m = \|\mathcal{E}\|$	$\|\|\mathcal{T}\|\|$	$\bar{m} = \|E\|$	d_{max}	TS
Msg	1,899	59,835	193	13,066	594	day
Chess	7,301	63,689	101	55,899	233	month
Facebook	45,813	855,541	53	165,349	590	month
Linux	26,885	1,096,440	98	149,569	14,172	month
Enron	86,803	1,147,027	46	279,745	4,311	month
Wiki	100,304	1,623,808	115	782,224	18,692	month
DBLP	1,721,631	11,986,363	72	15,919,980	5,980	year

Algorithms: We compare seven different algorithms for efficiency testing: 1) *Quick* [12]; 2) *TGRA* (Section IV); 3) *BB&SCM*-BA; 4) *BB&SCM*-EX; 5) *BB&SCM*-IN; 6) *BB&SCM*; and 7) *BB&SCM*-GR. Specifically, *Quick* is a state-of-the-art algorithm that can identify all maximal γ-quasi-cliques from de-temporal graph G. *BB&SCM*-BA enumerates all maximal ρ-stable (δ, γ)-quasi-cliques using *BB&SCM* framework but only applies the distance-based pruning and bound-based pruning. *BB&SCM*-EX and *BB&SCM*-IN are *BB&SCM*-BA with the excluding-node pruning and the including-node pruning, respectively. *BB&SCM*-GR is *BB&SCM* with the greedy vertex selection. To the best of our knowledge, there is no existing work to detect stable cohesive subgraphs on temporal networks. Thus, we use *Quick* and *BB&SCM*-BA as baselines for efficiency testing. Note that *BB&SCM*-BA, *BB&SCM*-EX, *BB&SCM*-IN, and *BB&SCM* have certain randomness (line 11 of Algorithm 5), we repeat ten times and report the average results for them. To evaluate the effectiveness of the proposed model, we adopt *Quick* [12], *DClique* [2], and *PCore* [10] as baseline models. *DClique* can find diversified temporal cohesive subgraphs, which may not have the property of stability. *PCore* is also a state-of-the-art temporal cohesive subgraph model. The model involves a vertex set and an interval set $\mathcal{I}$, which persistently maintains a k-core in any θ-length subinterval of each interval of $\mathcal{I}$. Unless otherwise stated, we terminate the execution of an algorithm when its running time exceeds one day.

Parameters: In our experiments, δ varies from 6 to 14 with a default value 10. γ varies from 0.5 to 0.9 with a default value 0.7, and ρ ranges from 0.5 to 0.7 with a default value 0.6. Unless otherwise stated, we take the default values of other parameters when changing a parameter.

B. Effectiveness Evaluation

Goodness Effectiveness Metric: Most previous effectiveness metrics (e.g., conductance or modularity) only consider

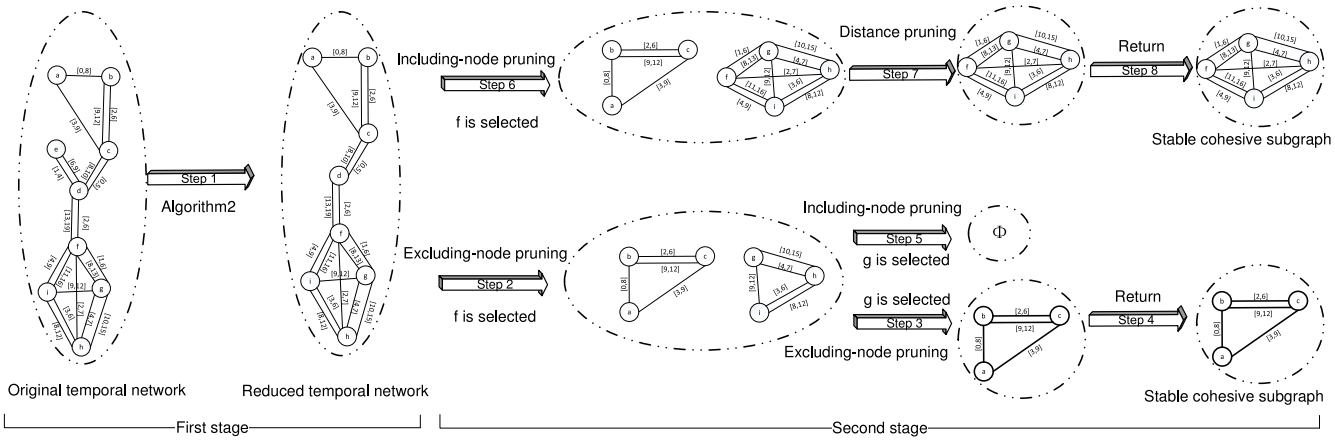


Fig. 3. Illustration of running *BB&SCM* algorithm on an example temporal network. By setting $\gamma = 0.6$, $\rho = 0.6$, and $\theta = 3$, the high-level idea of *BB&SCM* algorithm consists of two stages. At the first stage, a temporal graph reduction algorithm (i.e., Algorithm 2) is executed to reduce the size of the original temporal network. The second stage iteratively runs a branch and bound algorithm framework (i.e., Algorithm 5), which identifies final answers on the reduced temporal network. Specifically, it randomly first selects a node (e.g., f) and executes Excluding-node pruning and Including-node pruning in steps 2 and 6, respectively. For example, d and f are pruned by Excluding-node pruning at step 2. Then, node g is selected for next iteration. Until returning to a stable cohesive subgraph in step 4. Similarly, it executes Including-node pruning in step 6 and Distance pruning in step 7. The process continues until another stable cohesive subgraph is identified in step 8. Note that some pruning techniques (Section V-B) may not work during iteration, we omit the intermediate process.

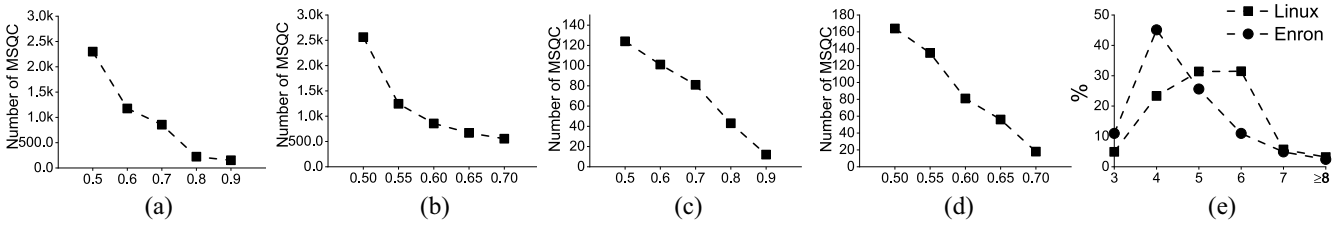


Fig. 4. Statistical characterization of our *MSQC*. (a) Linux (vary γ). (b) Linux (vary ρ). (c) Enron (vary γ). (d) Enron (vary ρ). (e) Size distributed of *MSQC*.

TABLE III
QUALITY OF *Quick*, *DClique*, *PCore*, AND *MSQC* IN TERMS OF TEMPORAL DENSITY (*TD*). EACH RESULT IS SHOWN IN AVERAGE ± STANDARD DEVIATION OVER ALL DETECTED TEMPORAL SUBGRAPHS. THE BEST RESULT IS HIGHLIGHTED IN BOLD

Model	Chess	Linux	Enron	DBLP
Quick	0.0234± 0.00	0.3408± 0.01	0.1331± 0.00	0.1089± 0.01
DClique	0.0085± 0.00	0.0038± 0.00	0.0489± 0.06	0.0012± 0.010
PCore	**0.0541**± 0.07	0.1400± 0.00	0.03379± 0.00	0.4227± 0.24
MSQC	0	**0.5110**± 0.05	**0.5367**± 0.09	**0.4847**± 0.07

structural information but temporal attributes [40]. Here, we generalize the existing temporal cohesiveness metric [17] to the entire time domain of the subgraph. Specifically, let $\mathcal{G}_C$ be a temporal subgraph, we define its temporal density $TD(C) = ([\sum_{u \in C} d_u^{\mathcal{T}_C}(C)]/[|C|(|C|-1)||\mathcal{T}_C||])$. Clearly, *TD* ranges from 0 to 1. Intuitively, the larger $TD(C)$ is, the denser C is in the whole temporal extent. Thereby, an temporal cohesive subgraph in a temporal graph should have a high temporal density.

Exp-1 (Quality of Quick, DClique, PCore, and MSQC): The experiments evaluate the quality of temporal subgraphs detected by different models with their default parameters. For PCore, different datasets have different parameter settings. Pcore did not reported parameter settings on other datasets except for the four datasets elaborated in Table III. Therefore, for a fair comparison, we choose the four datasets and report the comparison results in terms of temporal density, in which the scores are averaged over all detected temporal subgraphs by each model. Meanwhile, the standard deviations are also reported (Table III). The best scores are achieved by our model on all datasets expect for Chess. Since chess games were not held frequently, no stable quasi-cliques can be identified by our model. In addition, we also observe that *PCore* outperforms *Quick* and *DClique* in three of the four datasets. The reason is that *PCore* also considers multiple candidate intervals to measure the persistence of the community. However, *Quick* does not consider the temporal information of the graph and *DClique* only considers an interval to identify communities. In a nutshell, this experiment indicates that our model is denser and higher quality in terms of temporal feature than baselines.

Exp-2 (Statistical Characterization of Our MSQC): Fig. 4(a)–(d) reports the number of *MSQC* on Linux and Enron by varying γ or ρ with $\delta = 6$. Similar results can also be observed on other datasets. From Fig. 4(a)–(d), we can see that the number of *MSQC* decreases with an increasing γ or ρ. The reason is that with larger γ or ρ, the constraint of ρ-stable (δ, γ)-quasi-clique will be stronger, thus the number of *MSQC* decreases. Fig. 4(e) reports the distribution of the size of *MSQC* on Linux and Enron with $\gamma = 0.5$, $\rho = 0.5$ and $\delta = 3$. The similar trends can also be obtained on other datasets and with other parameter settings. We can observe

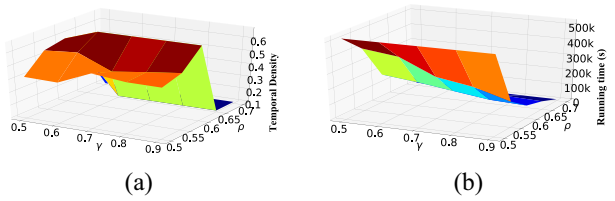


Fig. 5. Effect of different parameter distributions on the temporal density and running time of *MSQC*. (a) Average temporal density. (b) Running time (s).

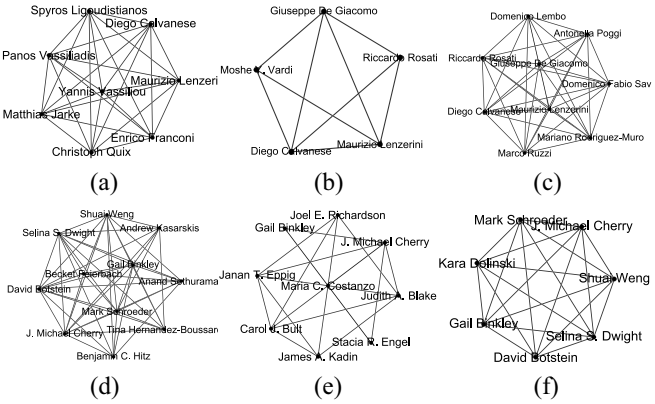


Fig. 6. Case study on DBLP. (a) *Quick*. (b) PCore. (c) Our model. (d) *Quick*. (e) PCore. (f) Our model.

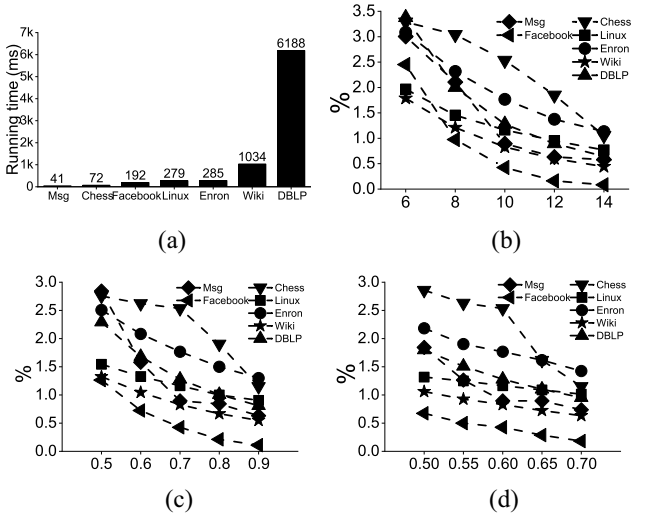


Fig. 7. Performance of temporal graph reduction technique. (a) Running time of *TGRA*. (b) Percentage of remaining nodes (vary δ). (c) Percentage of remaining nodes (vary γ). (d) Percentage of remaining nodes (vary ρ).

that the size of the *MSQC* is mainly distributed ranges from 4 to 6 on Linux and Enron. The reason can be explained as follows: it is difficult to form a stable quasi-clique at a small size and restrictions on *MSQC* in Definition 4 will become stronger as its size grows.

Exp-3 (Effect of Different Parameter Distributions on the Temporal Density and Running Time of MSQC): In this experiment, we study the impact of γ and ρ distributions for the temporal density and running time of *MSQC* using the dataset Linux and *BB&SCM* algorithm. To get more stable quasi-cliques, we fixed $\delta = 3$ and treated all output stable quasi-cliques in 6 days as candidates, and reported average temporal density and running time in Fig. 5. As shown in Fig. 5(a), we can observe that the medium values of γ and ρ are more probable to yield higher temporal density results. The running time [Fig. 5(b)] generally decreases with the growth of γ and ρ. The reason is that larger values of γ and ρ improve the pruning techniques (Section V-B); hence, the algorithm requires less time to identify stable quasi-cliques.

Exp-4 (Case Study on DBLP): Fig. 6 visualizes the cohesive subgraphs containing Prof. M. Lenzerini or J. M. Cherry obtained by *Quick*, *PCore*, and our model, respectively. In particular, Fig. 6(a)–(c) show the cohesive subgraph of M. Lenzerini by each model. As shown in Fig. 6(c), the cohesive subgraph of M. Lenzerini obtained by our model is a stable cohesive subgraph, because all authors in the subgraph collaborate closely and stably with M. Lenzerini from 1999 to 2016 (detailed information shown in Table IV). Although the subgraph obtained by *Quick* [Fig. 6(a)] is also cohesiveness in terms of structure, the authors did not work with M. Lenzerini frequently, resulting in that the cohesiveness of the subgraph was not stable over time. Namely, the subgraph obtained by *Quick* includes some unstable collaborators who did not frequently and densely collaborate with M. Lenzerini. Because *PCore* requires the k-core condition to be satisfied in any θ-length subinterval within a valid community, fewer authors are detected by *PCore* [Fig. 6(b)]. Thereby, it may not capture some potential stable cohesive subgraphs. Similar results can also be seen in the cohesive subgraph containing J. M. Cherry [Fig. 6(d)–(f)]. Looking at J. M. Cherry's homepage (https://cherrylab.stanford.edu/) and M. Lenzerini's homepage (http://www.dis.uniroma1.it/lenzerin/index.html/), we find that other authors in Fig. 6(f) are J. M. Cherry's students during 2000–2007, thus their cooperation during that period was dense and frequent. Similarly, in Fig. 6(c), those authors are M. Lenzerini's intimate parters during 1999–2016. In a nutshell, our model is more effective to identify stable cohesive subgraphs in temporal networks than the other models.

C. Efficiency Evaluation

Exp-5 (Running Time of Temporal Graph Reduction Algorithm): In this experiment, we report the running time of *TGRA* on all datasets under default parameter setting. As can be seen in Fig. 7(a), the running time of *TGRA* on the Msg, Chess, Facebook, Linux, Enron, and Wiki are less than 2 s. On the larger dataset DBLP, *TGRA* also only consumes around 6 s. These results demonstrate that our temporal graph reduction techniques are very efficient in practice, which is consistent with our theoretical analysis in Section IV.

Exp-6 (Percentage of Remaining Nodes With Varying Parameters): Fig. 7(b)–(d) shows the percentage of the remaining nodes after pruning the original graph with varying δ, γ, and ρ. *TGRA* effectively prunes lots of vertices on all datasets. For example, on Linux, Enron, and DBLP, the amount of the rest vertices obtained by *TGRA* algorithm are only 1.16%, 1.77%, and 1.28% of the original graph, respectively, with default parameter setting. As expected, the size of the reduced

TABLE IV
AUTHORS COLLABORATE WITH PROF. MAURIZIO LENZERINI (a) AND J. MICHAEL CHERRY (b). THE NUMBER 1-16 DENOTE D. CALVANESE, G. DE GIACOMO, M. LENZERINI, D. LEMBO, A. POGGI, R. ROSATI, M. RUZZI, D. F. SAVO, M. RODRIGUEZ-MURO, M. SCHROEDER, K. DOLINSKI, D. BOTSTEIN, S. WENG, S. S. DWIGHT, J. M. CHERRY, AND G. BINKLEY, RESPECTIVELY

(a)

Authors	Timestamps	Authors	Timestamps
1, 2	1997-2016	3, 7	2010-2013
1, 3	1994-1995, 1997-2013, 2015-2016	3, 8	2010-2013
1, 4	2004-2012	3, 9	2009-2011
1, 5	2006-2009	4, 5	2006-2014
1, 6	1998-1999, 2003-2012	4, 6	2002-2016
1, 7	2010-2011	4, 7	2003-2007, 2010-2015
1, 8	2015-2016	4, 8	2010-2016
1, 9	2007-2016	4, 9	2009-2011
2, 3	1994-2013, 2015-2016	5, 6	2006-2009, 2011-2013
2, 4	2004-2013	5, 7	2007-2008, 2010-2013
2, 5	2006-2009, 2011-2013	5, 8	2010-2013
2, 6	1996-1999, 2003-2013, 2015-2016	5, 9	2009-2011
2, 7	2010-2013	6, 7	2003-2006, 2010-2016
2, 8	2011-2013	6, 8	2010-2016
2, 9	2009-2011	6, 9	2009-2013
3, 4	2002-2015	7, 8	2008-2015
3, 5	2006-2016	7, 9	2010-2013
3, 6	1998-1999, 2001-2013	8, 9	2010-2013

(b)

Authors	Timestamps
10, 11	2000-2004, 2006-2007
10, 12	2000-2004, 2006-2007
10, 13	1999-2004, 2006-2007
10, 14	1998-2004, 2006-2007
10, 15	1998-2004, 2006-2007
10, 16	1998-2004, 2006-2007
11, 12	2000-2004, 2006-2007, 2009-2010
11, 13	2000-2004, 2006-2007
11, 14	2000-2004, 2006-2007
11, 15	2000-2007, 2016-2017
11, 16	2000-2004, 2006-2007, 2012-2012
12, 13	2000-2004, 2006-2007
12, 14	1999-2004, 2006-2007
12, 15	2000-2004, 2006-2007
12, 16	2000-2004, 2006-2007
13, 14	1999-2004, 2006-2007
13, 15	1999-2004, 2006-2007
13, 16	1999-2004, 2006-2007
14, 15	1998-2004, 2006-2007
14, 16	1998-2004, 2006-2007
15, 16	1998-2004, 2006-2007

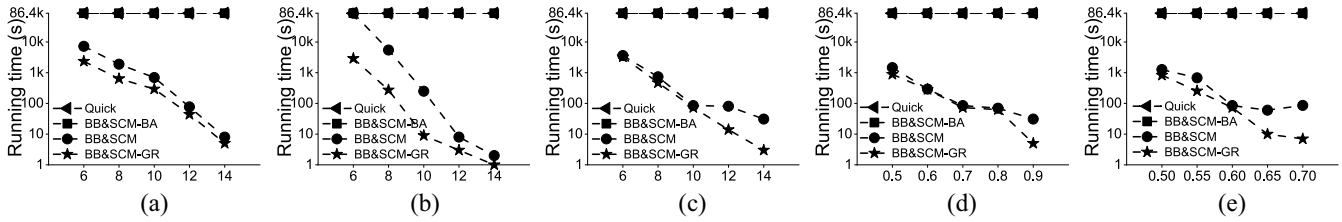


Fig. 8. Running time of different *MSQC* search algorithms with varying parameters. (a) Linux (vary δ). (b) Enron (vary δ). (c) DBLP (vary δ). (d) DBLP (vary γ). (e) DBLP (vary ρ).

graph decreases with increasing δ, γ, or ρ. This is because the power of stability-based pruning is enhanced when δ, γ, or ρ increases (Definition 8 and Lemma 1). These results demonstrate that *TGRA* is powerful for pruning the real-world graphs.

Exp-7 (Running Time of Quick, BB&SCM-BA, BB&SCM, and BB&SCM-GR With Varying Parameters): Fig. 8 illustrates the running time of different algorithms on Linux, Enron and DBLP with varying parameters. The results on the other datasets are consistent. As can be seen, BB&SCM-GR is consistently faster than all the other algorithms under all parameter settings. Instead, *Quick* and BB&SCM-BA are inefficient, which cannot terminate within one day on Linux, Enron, and DBLP in any parameter settings. Clearly, BB&SCM is also very efficient under most parameter settings. Moreover, we can see that the running time of BB&SCM is at least two orders of magnitude faster than those of *Quick* and BB&SCM-BA under most parameter settings. For example, when $\delta = 10$, $\gamma = 0.7$, and $\rho = 0.6$, BB&SCM takes 695 s to enumerate all maximal ρ-stable (δ, γ)-quasi-cliques on Linux, while *Quick* and BB&SCM-BA do not terminate within one day. These results suggest that proposed pruning techniques and the greedy vertex selection in Algorithm 5 are very effective to prune the search space. It is worth noting that although our model is a bit more complicated than traditional quasi-clique, the search is more efficient by considering temporal constraints. As expected, the running time of BB&SCM and BB&SCM-GR decrease with increasing δ, γ, or ρ. The reasons are as follows: when δ, γ, or ρ increases, *TGRA* can prune a large number of vertices, resulting in a very small search space in *B&B*. Moreover, when δ, γ, or ρ increases, the excluding-node pruning and including-node pruning are enhanced, resulting in that more unqualified vertices are pruned.

Exp-8 (Effectiveness of Different Pruning Techniques With Varying Parameters): Fig. 9 reports the results on Linux, Enron and DBLP with varying parameters. Similar results can be seen on the other datasets and parameters. As can be seen, *BB&SCM*-EX and *BB&SCM*-IN are faster than *BB&SCM*-BA, which indicates our excluding-node pruning and including-node pruning are effective in practice. Meanwhile, *BB&SCM*-EX is consistently faster than *BB&SCM*-IN under all parameter settings, which shows our excluding-node pruning is likely more effective than including-node pruning. In particular, the running time of BB&SCM-EX is 1–3 orders of magnitude faster than BB&SCM-IN. For example, when $\delta = 12$, $\gamma = 0.7$, and $\rho = 0.6$, BB&SCM-EX takes 14 s to enumerate all maximal ρ-stable (δ, γ)-quasi-cliques on Enron, while BB&SCM-IN takes 5738 s. More generally, the running time of BB&SCM-EX, BB&SCM-IN decrease with increasing δ, γ or ρ. Because

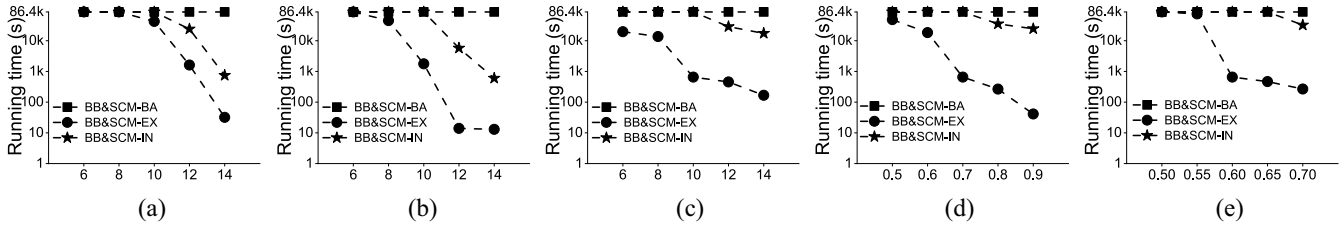


Fig. 9. Effectiveness of different pruning techniques with varying parameters. (a) Linux (vary δ). (b) Enron (vary δ). (c) DBLP (vary δ). (d) DBLP (vary γ). (e) DBLP (vary ρ).

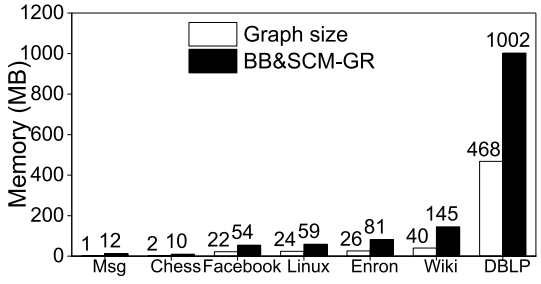


Fig. 10. Memory overhead of BB&SCM-GR.

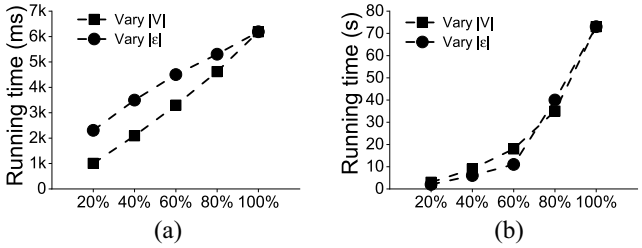


Fig. 11. Scalability (DBLP). (a) *TGRA*. (b) BB&SCM-GR.

both excluding-node pruning and including-node pruning are improved when δ, γ, or ρ increases, resulting in that more vertices are pruned. However, for larger DBLP, the two pruning techniques are still not good enough. Thereby, the experiment further illustrates that to improve the efficiency, excluding-node pruning and including-node pruning need to be combined for solving our problem [see Fig. 8(c)–(e)].

Exp-9 (Memory Overhead of BB&SCM-GR): Fig. 10 shows the memory overhead of BB&SCM-GR on all datasets with default parameter values. We can see that the memory overhead of BB&SCM-GR is higher than the size of the temporal graph, but typically lower than four times of the size of the large temporal graph. For example, on DBLP, BB&SCM-GR consumes 1002-MB memory while the temporal graphs needs 468 MB. These results demonstrate that our BB&SCM-GR can achieve near linear space complexity.

Exp-10 (Scalability Testing): We choose the largest dataset DBLP to test the scalability of *TGRA* and BB&SCM-GR under default parameter setting. Specifically, we generate four subgraphs by randomly selecting 20%–100% vertices or temporal edges from DBLP. Then, we evaluate the running time of our algorithms on these subgraphs (Fig. 11). We can see that both *TGRA* and BB&SCM-GR scales near linear with respect to the size of the temporal graph. Similar results can be also observed in other parameter settings. The results indicate that our algorithms are scalable.

VII. Conclusion and Future Work

In this article, we systematically formalized the stable cohesive subgraphs on temporal graphs and present solutions to extract it. Specifically, we proposed a novel cohesive subgraph model, called maximal ρ-stable (δ, γ)-quasi-clique, to characterize both the cohesiveness and the stability of a subgraph. We showed that the problem of mining all maximal ρ-stable (δ, γ)-quasi-cliques is NP-hard. To efficiently solve the problem, we first proposed *TGRA* to significantly reduce the original temporal graph. Then, on the reduced temporal graph, a powerful branch and bound enumeration algorithm, named *BB&SCM*, with four carefully designed pruning techniques was devised to efficiently identify all maximal ρ-stable (δ, γ)-quasi-cliques. Comprehensive experiments on seven real-world temporal networks demonstrated the efficiency, scalability, and effectiveness of our approach. There are some interesting directions for future exploration.

1) Adopting other cohesive subgrph models (e.g., clique, k-truss) to model the stable cohesive subgraph on temporal networks.
2) Considering query-biased community search problem from massive temporal networks.

Appendix

Proof of Lemma 1: The lemma can be proved by contradiction. Assume that $\exists$ ρ-stable (δ, γ)-quasi-clique H such that $S \cup \{u\} \subseteq H \subseteq U$. By Definition 6, we can know $\text{co}(\mathcal{MI}_\delta^\gamma(H)) \geq \rho||\mathcal{T}||$. $\forall\ T \subseteq \mathcal{MI}_\delta^\gamma(H)$, we have $a_u^T(H) \geq \gamma(|H| - 1)$, resulting in $a_u^T(H) \geq \gamma(\max\{\delta, |S|, lb(H, S)\} - 1)$ due to $|H| \geq \max\{\delta, |S|, lb(H, S)\}$. Thus, T is a candidate dense interval of u w.r.t. $(\mathcal{G}_H, S)$, and then we can derive $\text{co}(\mathcal{MCDI}(u, H, IS(\mathcal{T}_H), S)) \geq \text{co}(\mathcal{MI}_\delta^\gamma(H))$. In a similar way, $\forall I \subseteq \mathcal{MCDI}(u, H, IS(\mathcal{T}_H), S)$, we have $a_u^I(U) \geq a_u^I(H) \geq \gamma(\max\{\delta, |S|, lb(H, S)\} - 1)$. Since $lb(U, S)$ and $lb(H, S)$ represent the lower bound of ρ-stable (δ, γ)-quasi-clique in $(\mathcal{G}_U, S)$ and $(\mathcal{G}_H, S)$, respectively, we have $lb(H, S) \geq lb(U, S)$ due to $H \subseteq U$. Consequently, $a_u^I(U) \geq \gamma(\max\{\delta, |S|, lb(H, S)\} - 1) \geq \gamma(\max\{\delta, |S|, lb(U, S)\} - 1)$. Namely, I is a candidate dense interval of u w.r.t. $(\mathcal{G}_U, S)$. Therefore, $\text{co}(\mathcal{MCDI}(u, U, IS(\mathcal{T}_U), S)) \geq \text{co}(\mathcal{MCDI}(u, H, IS(\mathcal{T}_H), S)) \geq \text{co}(\mathcal{MI}_\delta^\gamma(H)) \geq \rho||\mathcal{T}||$, which contradicts with $\text{co}(\mathcal{MCDI}(u, U, IS(\mathcal{T}_U), S)) < \rho \cdot$

$||\mathcal{T}||$. Thus, vertex u can be removed from task $(\mathcal{G}_U, S)$ without loss of accuracy.

Proof of Lemma 2: Clearly, $d_v^t(U)$ decreases by 1 when timestamp t is shared by $\mathcal{MCDI}(v, U, IS(\mathcal{T}_U), S)$ and $\mathcal{T}_U(u, v)$, i.e., there are interval $[t_{s_i}, t_{e_i}] \in \mathcal{MCDI}(v, U, IS(\mathcal{T}_U), S)$ and $[t_{l_j}, t_{r_j}] \in \mathcal{T}_U(u, v)$ such that $t \in [t_{s_i}, t_{e_i}] \cap [t_{l_j}, t_{r_j}]$, while $d_v^t(U)$ keeps unchanged for other timestamps. Thus, t falls into interval set $\mathcal{T}_a$ when $d_v^t(U)$ decreases by 1. Moreover, by Definition 8, we can know that the maximal candidate dense intervals of v w.r.t. $(\mathcal{G}_U, S)$ in interval set $\mathcal{T}_a$ may be shrunk because the degree of v is reduced in $\mathcal{T}_a$. However, the maximal candidate dense intervals of v w.r.t. $(\mathcal{G}_U, S)$ in interval set $\mathcal{T}_b$ are not affected since $d_v^t(U \setminus \{u\})$ does not change for any timestamp t in $\mathcal{T}_b$. Thus, the deletion of u only affects the maximal candidate dense interval T of v, in which $T \in \mathcal{T}_a$. Consequently, when vertex u is deleted from $(\mathcal{G}_U, S)$, $\mathcal{MCDI}(v, U \setminus \{u\}, IS(\mathcal{T}_{U\setminus\{u\}}), S) = \mathcal{T}_b \cup \mathcal{MCDI}(v, U \setminus \{u\}, \mathcal{T}_a, S)$ for any $v \in N_u(U)$ holds.

Proof of Lemma 3: When $(h - 2/h - 1) < \gamma \leq 1$. $\forall u \in H$, we can derive $a_u^T(H) \geq \gamma \cdot (h - 1) > h - 2$ by Definition 4. Moreover, $a_u^T(H) \leq h-1$. Thus, $a_u^T(H) = h-1$. Consequently, the de-temporal graph of $\mathcal{G}_H(T)$ is a complete graph and $D(\mathcal{G}_H(T)) = 1$.

When $(h - 2/2(h - 1)) < \gamma \leq (h - 2/h - 1)$. On the one hand, $\forall w \in H$ that has $a_w^T(H) \geq \gamma \cdot (h - 1) > (h - 2/2)$ by Definition 4, so $d_w^T(H) > (h - 2/2) \cdot ||T||$. On the other hand, we assume that $D(\mathcal{G}_H(T)) > 2$, i.e., $\exists u, v \in H$ such that $N_u^T(H) \cap N_v^T(H) = \emptyset$, where $N_u^T(H) = \{w \in H | \exists (u, w, [t_s, t_e]) \in \mathcal{E}_H(T)\}$. So, $\forall t \in T$, we have $d_u^t(H) \leq h - 1 - 1 - d_v^t(H)$, $\sum_{t\in T} d_u^t(H) \leq (h-2) \cdot ||T|| - \sum_{t\in T} d_v^t(H)$. So, $d_u^T(H) < (h - 2/2) \cdot ||T||$, which contradicts with previous results. Thus, $D(\mathcal{G}_H(T)) \leq 2$.

Proof of Lemma 4: $\forall T \in \mathcal{MI}_\delta^\gamma(Y)$, $u \in S$, we have $a_u^T(Y) \geq \gamma(|Y| - 1)$ (Definition 4). $a_u^T(Y) = a_u^T(S) + a_u^T(Y \setminus S) = a_u^T(S) + a_{Y\setminus S}^T(u)$, that is $|S|\gamma(|Y| - 1)| \leq \sum_{u\in S}\{a_u^T(S) + a_{Y\setminus S}^T(u)\} \leq A_S(U, S) + \sum_{i=1}^{|Y\setminus S|} A_S(U, u_i)$, where $a_u^T(S) \leq A_S(U, u)$ and $\sum_{u\in S} a_{Y\setminus S}^T(u) = \sum_{w\in Y\setminus S} a_w^T(S) \leq \sum_{i=1}^{|Y\setminus S|} A_S(U, u_i)$ due to $T \subseteq \mathcal{T}_U$. Let $|Y \setminus S| = k$, we have $|S|\gamma(|S| + k - 1) \leq A_S(U, S) + \sum_{i=1}^{k} A_S(U, u_i)$; thus, $|Y| \leq ub(U, S)$.

Proof of Lemma 5: Clearly, we know that $|Y| \geq lb_1(U, S)$ by Lemma 4. On the other hand, $\forall T \in \mathcal{MI}_\delta^\gamma(Y)$ and $u \in S$, we have $a_u^T(Y) \geq \gamma(|Y| - 1)$ by Definition 4. $a_u^T(Y) = ((\sum_{t\in T} d_u^t(Y))/||T||) \leq ([\sum_{t\in T}(d_u^t(S) + |Y| - |S|)]/||T||) = a_u^T(S) + |Y| - |S|$, thus, $\gamma(|Y| - 1) \leq a_u^T(Y) \leq a_u^T(S) + |Y| - |S|$. That is, $|S|\gamma(|Y| - 1) \leq \sum_{u\in S} a_u^T(S) + |S|(|Y| - |S|) \leq A_S(U, S) + |S|(|Y| - |S|)$ due to $T \subseteq \mathcal{T}_U$. Thus, we have $|Y| \geq ([|S|^2 - \gamma|S| - A_S(U, S)]/[|S|(1 - \gamma)])$. So, this lemma holds.

Proof of Lemma 6: The first three conditions are obvious. For condition (4), there does not exist a ρ-stable (δ, γ)-quasi-clique that contains set S, which contradicts the fact that set S must be included in certain ρ-stable (δ, γ)-quasi-clique in $(\mathcal{G}_U, S)$.

Proof of Lemma 7: According to Definition 8, all maximal dense intervals of any ρ-stable (δ, γ)-quasi-clique in $(\mathcal{G}_C, S \cup \{v\})$ fall into $\mathcal{I}$. Thus, $\forall\rho$-stable (δ, γ)-quasi-clique H of $(\mathcal{G}_C, S\cup\{v\})$, we have $\text{co}(\mathcal{MI}_\delta^\gamma(H)) \leq \text{co}(\mathcal{I})$. $\text{co}(\mathcal{MI}_\delta^\gamma(H)) < \rho \cdot ||\mathcal{T}||$ when $\text{co}(\mathcal{I}) < \rho \cdot ||\mathcal{T}||$, i.e., there is no ρ-stable (δ, γ)-quasi-clique in $(\mathcal{G}_C, S\cup\{v\})$ due to Definition 6. On the other hand, we assume that there is a ρ-stable (δ, γ)-quasi-clique H such that $S \cup \{v, u\} \subseteq H \subseteq C$. By Definition 8, we have $\text{co}(\mathcal{MI}_\delta^\gamma(H)) \leq \text{co}(\mathcal{I} \bigotimes \mathcal{MCDI}(u, C, IS(\mathcal{T}_C), S \cup \{v, u\})) < \rho \cdot ||\mathcal{T}||$, i.e., $\text{co}(\mathcal{MI}_\delta^\gamma(H)) < \rho \cdot ||\mathcal{T}||$, which contradicts with Definition 6.

REFERENCES

[1] Y.-Q. Zhang, X. Li, J. Xu, and A. V. Vasilakos, "Human interactive patterns in temporal networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 2, pp. 214–222, Feb. 2015.

[2] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. S. Lui, "Diversified temporal subgraph pattern mining," in *Proc. KDD*, 2016, pp. 1965–1974.

[3] T. Viard, M. Latapy, and C. Magnien, "Computing maximal cliques in link streams," *Theor. Comput. Sci.*, vol. 609, pp. 245–252, Jan. 2016.

[4] P. Rozenshtein and A. Gionis, "Mining temporal networks," in *Proc. KDD*, 2019, pp. 3225–3226.

[5] K. Semertzidis and E. Pitoura, "Top-k durable graph pattern queries on temporal graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 1, pp. 181–194, Jan. 2019.

[6] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *Proc. VLDB Endowm.*, vol. 7, no. 9, pp. 721–732, 2014.

[7] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in *Proc. ICDE*, 2016, pp. 145–156.

[8] S. Huang, A. W. Fu, and R. Liu, "Minimum spanning trees in temporal graphs," in *Proc. SIGMOD*, 2015, pp. 419–430.

[9] L. Chang and L. Qin, "Cohesive subgraph computation over large sparse graphs," in *Proc. ICDE*, 2019, pp. 2068–2071.

[10] R. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *Proc. ICDE*, 2018, pp. 797–808.

[11] Z. Borbora and J. Srivastava, "User behavior modelling approach for churn prediction in online games," in *Proc. PASSAT*, 2012, pp. 51–60.

[12] G. Liu and L. Wong, "Effective pruning techniques for mining quasi-cliques," in *Proc. PKDD*, 2008, pp. 33–49.

[13] S. Sanei-Mehri, A. Das, and S. Tirthapura, "Enumerating top-k quasi-cliques," in *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 1107–1112.

[14] R. Kumar and T. Calders, "2SCENT: An efficient algorithm to enumerate all simple temporal cycles," *Proc. VLDB Endowm.*, vol. 11, no. 11, pp. 1441–1453, 2018.

[15] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai, "Fast computation of dense temporal subgraphs," in *Proc. ICDE*, 2017, pp. 361–372.

[16] E. Galimberti, A. Barrat, F. Bonchi, C. Cattuto, and F. Gullo, "Mining (maximal) span-cores from temporal networks," in *Proc. CIKM*, 2018, pp. 107–116.

[17] L. Chu, Y. Zhang, Y. Yang, L. Wang, and J. Pei, "Online density bursting subgraph detection from temporal graphs," *Proc. VLDB Endowm.*, vol. 12, no. 13, pp. 2353–2365, 2019.

[18] L. Lin, P. Yuan, R. Li, and H. Jin, "Mining diversified top-r lasting cohesive subgraphs on temporal networks," *IEEE Trans. Big Data*, early access, Feb. 9, 2021, doi: 10.1109/TBDATA.2021.3058294.

[19] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified top-k clique search," in *Proc. ICDE*, 2015, pp. 387–398.

[20] C. Lu, J. X. Yu, H. Wei, and Y. Zhang, "Finding the maximum clique in massive graphs," *Proc. VLDB Endowm.*, vol. 10, no. 11, pp. 1538–1549, 2017.

[21] S. Khuller and B. Saha, "On finding dense subgraphs," in *Proc. ICALP*, 2009, pp. 597–608.

[22] C. E. Tsourakakis, "The k-clique densest subgraph problem," in *Proc. WWW*, 2015, pp. 1122–1132.

[23] 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 *Proc. KDD*, 2013, pp. 104–112.

[24] P. Lee and L. V. S. Lakshmanan, "Query-driven maximum quasi-clique search," in *Proc. SDM*, 2016, pp. 522–530.

[25] B. Boden, S. Günnemann, H. Hoffmann, and T. Seidl, "MiMAG: Mining coherent subgraphs in multi-layer graphs with edge labels," *Knowl. Inf. Syst.*, vol. 50, no. 2, pp. 417–446, 2017.

[26] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *Proc. ICDE*, 2011, pp. 51–62.

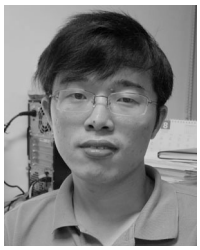
[27] R. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *Proc. VLDB Endowm.*, vol. 8, no. 5, pp. 509–520, 2015.
[28] L. Cui, L. Yue, D. Wen, and L. Qin, "K-connected cores computation in large dual networks," *Data Sci. Eng.*, vol. 3, no. 4, pp. 293–306, 2018.
[29] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proc. VLDB Endowm.*, vol. 5, no. 9, pp. 812–823, 2012.
[30] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *Proc. KDD*, 2010, pp. 939–948.
[31] J. Cheng, X. Wu, M. Zhou, S. Gao, Z. Huang, and C. Liu, "A novel method for detecting new overlapping community in complex evolving networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 9, pp. 1832–1844, Sep. 2019.
[32] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proc. KDD*, 2013, pp. 349–360.
[33] A. Angel, N. Koudas, N. Sarkas, D. Srivastava, M. Svendsen, and S. Tirthapura, "Dense subgraph maintenance under streaming edge weight updates for real-time story identification," *VLDB J.*, vol. 23, no. 2, pp. 175–199, 2014.
[34] G. Rossetti and R. Cazabet, "Community discovery in dynamic networks: A survey," *ACM Comput. Survey*, vol. 51, no. 2, p. 35, 2018.
[35] A. Epasto, S. Lattanzi, and M. Sozio, "Efficient densest subgraph computation in evolving graphs," in *Proc. WWW*, 2015, pp. 300–310.
[36] R.-H. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2453–2465, Oct. 2014.
[37] S. Qiao *et al.* "Dynamic community evolution analysis framework for large-scale complex networks based on strong and weak events," *IEEE Trans. Syst., Man, Cybern., Syst.*, early access, Jan. 7, 2020, doi: 10.1109/TSMC.2019.2960085.
[38] Z. Zeng, J. Wang, L. Zhou, and G. Karypis, "Coherent closed quasi-clique discovery from large dense graph databases," in *Proc. KDD*, 2006, pp. 797–802.
[39] X. Yan and J. Han, "gSPAN: Graph-based substructure pattern mining," in *Proc. ICDM*, 2002, pp. 721–724.
[40] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowl. Inf. Syst.*, vol. 42, no. 1, pp. 181–213, 2015.

Longlong Lin is currently pursuing the Ph.D. degree with the Huazhong University of Science and Technology, Wuhan, China.

His current research interests include temporal community detection and search, cohesive sub-graph mining, and graph representation learning.

Pingpeng Yuan (Member, IEEE) received the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2002.

He is a Professor with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. During exploring his research, he implements systems and innovative applications in addition to investigating theoretical solutions and algorithmic design. Thus, he is the principle developer in multiple system prototypes, including TripleBit, PathGraph, and SemreX. His research interests include databases, knowledge representation and reasoning, and NLP, with a focus on high-performance computing.

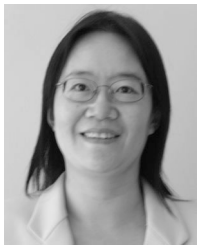


Rong-Hua Li received the Ph.D. degree in computer science from the Chinese University of Hong Kong, Hong Kong, in 2013.

He is currently an Associate Professor with the Beijing Institute of Technology, Beijing, China. His research interests include cohesive subgraph mining, graph computation systems, and graph representation learning.

Jifei Wang is currently pursuing the M.D. degree with the Huazhong University of Science and Technique, Wuhan, China.

His current research interests include temporal community detection and cohesive subgraph mining.



Ling Liu (Fellow, IEEE) received the Ph.D. degree in computer science from Tilburg University, Tilburg, The Netherlands, in 1993.

She is currently a Professor with the School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA. She directs the research programs with the Distributed Data Intensive Systems Lab, examining various aspects of large-scale big data-powered artificial intelligence systems, and machine learning algorithms and analytics, including performance, availability, privacy, security, and trust. Her current research is primarily supported by USA National Science Foundation under CISE programs.

Prof. Liu is a recipient of the IEEE Computer Society Technical Achievement Award in 2012 and the Best Paper Award from numerous top venues, including IEEE ICDCS, WWW, ACM/IEEE CCGrid, IEEE Cloud, and IEEE ICWS. She has served on editorial board for over a dozen international journals, including the Editor in Chief for IEEE TRANSACTIONS ON SERVICE COMPUTING from 2013 to 2016, and currently the Editor in Chief for *ACM Transactions on Internet Computing*.

Hai Jin (Fellow, IEEE) received the Ph.D. degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1994.

He is a Cheung Kung Scholars Chair Professor of Computer Science and Engineering with the Huazhong University of Science and Technology. He worked with The University of Hong Kong, Hong Kong, from 1998 to 2000, and was a Visiting Scholar with the University of Southern California, Los Angeles, CA, USA, from 1999 to 2000. He has coauthored 22 books and published over 700 research papers. He is the Chief Scientist of ChinaGrid, the largest grid computing project in China, and the chief scientists of National 973 Basic Research Program Project of Virtualization Technology of Computing System, and Cloud Security. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

Dr. Jin was awarded the Excellent Youth Award from the National Science Foundation of China in 2001. In 1996, he was awarded the German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz in Germany. He is a CCF Fellow and a member of the ACM.