Mining Bursting Core in Large Temporal Graphs

Hongchao Qin*, Rong-Hua Li*, Ye Yuan*, Guoren Wang*, Lu Qin#, Zhiwei Zhang*
*Beijing Institute of Technology, China; #University of Technology Sydney, Australia
{hcqin; rhli; yuan-ye; wanggr}@bit.edu.cn; Lu.Qin@uts.edu.au; cszwzhang@outlook.com

ABSTRACT

Temporal graphs are ubiquitous. Mining communities that are bursting in a period of time is essential for seeking real emergency events in temporal graphs. Unfortunately, most previous studies on community mining in temporal networks ignore the bursting patterns of communities. In this paper, we study the problem of seeking bursting communities in a temporal graph. We propose a novel model, called the (l, δ)-maximal bursting core, to represent a bursting community in a temporal graph. Specifically, an (l, δ)-maximal bursting core is a temporal subgraph in which each node has an average degree no less than δ in a time segment with length no less than l. To compute the (l, δ)-maximal bursting core, we first develop a novel dynamic programming algorithm that can reduce time complexity of calculating the segment density from $O(|\mathcal{T}|)^2$ to $O(|\mathcal{T}|)$. Then, we propose an efficient updating algorithm which can update the segment density in $O(l)$ time. In addition, we develop an efficient algorithm to enumerate all (l, δ)-maximal bursting cores that are not dominated by the others in terms of l and δ. The results of extensive experiments on 9 real-life datasets demonstrate the effectiveness, efficiency and scalability of our algorithms.

PVLDB Reference Format:
Hongchao Qin*, Rong-Hua Li*, Ye Yuan*, Guoren Wang*, Lu Qin#, Zhiwei Zhang*. Mining Bursting Core in Large Temporal Graphs. PVLDB, 14(1): XXX-XXX, 2022.

1 INTRODUCTION

In temporal graphs, each edge can be represented as a triple (u, v, t), where u, v are two end nodes of the edge and t denotes the interaction time between u and v [3, 19]. The interaction patterns in a temporal graph are often known to be bursty, e.g., human communication events last for a short time [3, 19]. Here, the bursty patterns denote a number of events occurring in and lasting for a short time. In this paper, we study a particular bursty pattern on temporal networks, called the bursting core, which is a dense subgraph pattern that occurs in a short time. In other words, we aim to identify densely-connected subgraphs from a temporal graph in which each node rapidly accumulates its adjacent edges. Specifically, a bursting core is a temporal subgraph in which each node has an average degree no less than a given constant in a lasting time segment.

There are evidences that the timing of many human activities, ranging from communication to entertainment and work patterns, follow non-Poisson statistics, characterized by bursts of rapidly occurring events separated by long periods of inactivity [3]. Therefore, the popular topics in temporal networks are changing over time, but every popular topic will last for a period of time. By mining the bursting communities in such temporal social networks, we can identify a group of users that densely interact with each other in a lasting and bursting time. The common topics discussed among the users in a bursting community may represent an activity that recently spreads over the networks. Our case studies are presented in Section 5 (see Figs. 9 and 13), which gives two examples of such bursting communities in a communication network Enron and a collaboration network DBLP, revealing real activities and communities in the temporal networks. Therefore, identifying bursting communities is useful for finding such emerging activities in a temporal network.

In the literature, there exist a few studies on mining cohesive subgraphs in temporal graphs. For example, Wu et al. [38] proposed a temporal core model to find cohesive subgraphs in a temporal graph; Ma et al. [27] identified the densest subgraphs in a weighted temporal graph; Qin et al. [29] devised an efficient algorithm to seek periodic cliques in a temporal graph. The above studies did not consider the bursting patterns of the community; thus, their techniques cannot be applied to solve our problem. Recently, Chu et al. [11] studied the problem of mining the densest and bursting subgraphs in temporal graphs. However, to search the bursting communities, the model of the densest and bursting subgraph has three limitations: ***(i)*** The original global densest subgraphs may contain outliers that are not dense regions of the graph, so they cannot be directly treated as communities [30, 34]. ***(ii)*** Mining the densest and bursting subgraph is NP-hard [11]; thus, it is difficult to handle large temporal graphs. ***(iii)*** The densest and bursting subgraph model can only return the subgraph with the highest density; therefore, it is difficult to find other dense and bursting subgraphs in the temporal graph [11].

To the best of our knowledge, we are the first to study the bursting core mining problem, i.e., the problem of finding a cohesive temporal subgraph in which each node *bursts out* in a short time.

Contributions. In this paper, we formulate and provide efficient solutions for finding bursting cores in a temporal graph. In particular, we make the following main contributions.

***(i)* Novel Model.** We propose a novel concept, called the (l, δ)-maximal bursting core, to characterize the bursting community in temporal graphs. Each node in the (l, δ)-maximal bursting core has an average degree no less than δ in a time segment with length no less than l. We also define a new concept called the pareto-optimal (l, δ)-maximal bursting core, which denotes the set of (l, δ)-maximal bursting cores that are not dominated by other (l, δ)-maximal bursting cores in terms of the parameters l and δ. The pareto-optimal (l, δ)-maximal bursting cores can provide a good summary of all the bursting communities in a temporal graph over the entire parameter space.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

***(ii)* New Algorithms.** We have proved that the (l, δ)-maximal bursting core satisfies the properties of uniqueness, containment and reduction, thus we can apply the peeling-based core decomposition framework to seek the (l, δ)-maximal bursting core. However, when invoking the peeling algorithm, the bursty status of the node must be checked once an edge is deleted. Therefore, the main technical challenge is to determine whether a node u has an average degree no less than δ in a time segment with length no less than l, i.e. to compute the maximum average sequential numbers of the degree sequence. We show that the naive algorithm for solving this issue requires $O(|\mathcal{T}|^2)$ time, where $|\mathcal{T}|$ is the number of timestamps in the temporal network. To improve the efficiency, we first propose a dynamic programming algorithm which takes $O(|\mathcal{T}|)$ to solve this issue. Then, we develop a more efficient updating algorithm which can update the maximum average sequential numbers in $O(l)$ time. In addition, we also propose an efficient algorithm to find the pareto-optimal (l, δ)-maximal bursting cores.

***(iii)* Extensive Experiments.** We conduct comprehensive experiments using 9 real-life temporal graphs to evaluate the proposed algorithm under different parameter settings. The results indicate that our algorithms significantly outperform the baselines in terms of community quality. We also perform a case study on the Enron dataset. The results demonstrate that our approach can identify many meaningful and interesting bursting communities that cannot be found by the other methods. In addition, we also evaluate the efficiency of the proposed algorithms, and the results demonstrate the high efficiency of our algorithms. For example, on a large-scale temporal graph with more than 1M nodes and 10M edges, our algorithm can find a bursting community in 26.95 seconds.

Organization. Section 2 introduces the model and formulates our problem. The algorithms to efficiently mining bursting communities are proposed in Section 3 and 4. Experimental studies are presented in Section 5, and the related work is discussed in Section 6. Section 7 draws the conclusion of this paper.

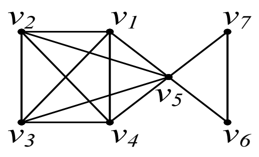
2 PRELIMINARIES

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be an undirected ***temporal graph***, where $\mathcal{V}$ and $\mathcal{E}$ denote the set of nodes and temporal edges, an arithmetic time sequence $\mathcal{T} = \{t_1, t_2 ... t_{|\mathcal{T}|}\}$ denotes the set of all timestamps in which $t_i - t_{i-1}$ is a constant for each integer i. Each temporal edge $e \in \mathcal{E}$ is a triplet (u, v, t), where u, v are nodes in $\mathcal{V}$, and $t \in \mathcal{T}$ is the interaction time between u and v. The ***de-temporal graph*** of $\mathcal{G}$ denoted by $G = (V, E)$ is a simple graph that ignores all the timestamps associated with the temporal edges. More formally, for the de-temporal graph G of $\mathcal{G}$, we have $V = \mathcal{V}$ and $E = \{(u, v)|(u, v, t) \in \mathcal{E}\}$.

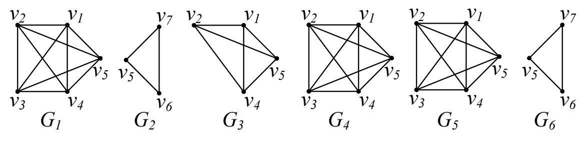
By sorting the temporal edges in a chronological order, the temporal graph can be represented as a link stream [22]. The widely-used approach to extract interesting patterns from a temporal graph relies on series of snapshots [2, 22]. The i-th ***snapshot*** of $\mathcal{G}$ is a de-temporal graph $G_i = (V_i, E_i)$ where $V_i = \{u|(u, v, i) \in \mathcal{E}\}$ and $E_i = \{(u, v)|(u, v, i) \in \mathcal{E}\}$. Each timestamp is an integer, because the *UNIX* timestamps are integers in practice. For convenience, we use $\mathcal{T} = \{1, 2, ...|\mathcal{T}|\}$ to represent timestamps $\{t_1, t_2 ... t_{|\mathcal{T}|}\}$ in the rest of this paper. Fig.1 (a) illustrates a temporal graph $\mathcal{G}$ with 42 temporal edges and $\mathcal{T} = [1 : 6]$. Fig.1 (b) illustrates the de-temporal

t	(u, v)
1	$(v_1, v_2)(v_1, v_3)(v_1, v_4)(v_1, v_5)(v_2, v_3)$ $(v_2, v_4)(v_2, v_5)(v_3, v_4)(v_3, v_5)(v_4, v_5)$
2	$(v_5, v_6)(v_5, v_7)(v_6, v_7)$
3	$(v_1, v_2)(v_1, v_4)(v_1, v_5)$ $(v_2, v_4)(v_2, v_5)(v_4, v_5)$
4	$(v_1, v_2)(v_1, v_3)(v_1, v_4)(v_1, v_5)(v_2, v_3)$ $(v_2, v_4)(v_2, v_5)(v_3, v_4)(v_3, v_5)(v_4, v_5)$
5	$(v_1, v_2)(v_1, v_3)(v_1, v_4)(v_1, v_5)(v_2, v_3)$ $(v_2, v_4)(v_2, v_5)(v_3, v_4)(v_3, v_5)(v_4, v_5)$
6	$(v_5, v_6)(v_5, v_7)(v_6, v_7)$

(a) Temporal edges in $\mathcal{G}$



(b) The de-temporal graph G



(c) The six snapshots of $\mathcal{G}$

Figure 1: Basic concepts of the temporal graph

graph G of $\mathcal{G}$ in Fig.1 (a). Fig.1 (c) illustrates all the six snapshots of $\mathcal{G}$ in Fig.1 (a).

Here, we introduce some necessary concepts. Let $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$ be the number of nodes and temporal edges, $N_u(G) = \{v|(u, v) \in E\}$ be the set of neighbor nodes of u, and $deg_G[u] = |N_u(G)|$ be the degree of u in G. For a given set of nodes $S \subseteq V$, a subgraph $G_S = (V_S, E_S)$ is referred to as an induced subgraph of G from S if $V_S = S$ and $E_S = \{(u, v)|u, v \in V_S, (u, v) \in E\}$.

The nodes in bursting cores have a feature in common that they have high degrees in the induced subgraphs of some continuous time periods. We introduce some definitions below to describe the properties.

Definition 1 (temporal subgraph). *Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, a continuous time interval $T = [t_s : t_e] \subseteq [1 : |\mathcal{T}|]$ and a given set of nodes $S \subseteq \mathcal{V}$, a temporal subgraph can be denoted by $\mathcal{G}_S(T) = (S, \mathcal{E}_S(T), T)$, and it is an induced temporal graph of $\mathcal{G}$ from temporal edges $\mathcal{E}_S(T) = \{(u, v, t)|u, v \in S, t \in T, (u, v, t) \in \mathcal{E}\}$.*

Based on Definition 1, a temporal subgraph $\mathcal{G}_S(T)$ is an induced graph from nodes set S in time interval T, and it can also extract a series of snapshots. The snapshot of temporal subgraph in time i is the induced subgraph of $V_i \cap S$, thus it can be denoted by $G_{V_i \cap S}$. For each node $u \in S$, $deg_{G_{V_i \cap S}}[u] = |N_u(G_{V_i \cap S})| = |N_u(G_i) \cap S|$.

Definition 2 (degree sequence). *Given a temporal subgraph $\mathcal{G}_S(T)$, for node $u \in S$, the degree sequence of u in $\mathcal{G}_S(T)$, abbreviated as $\mathcal{DS}(u, \mathcal{G}_S(T))$, is a sequence of u's degree in each snapshot of $\mathcal{G}_S(T)$. Each item in the degree sequence can be denoted by $\mathcal{DS}(u, \mathcal{G}_S(T))[i] = |N_u(G_i) \cap S|$.*

Definition 3 (l-segment density). *Given an integer l, a time interval $T = [t_s : t_e]$ and $\mathcal{DS}(u, \mathcal{G}_S(T))$, the l-segment density of u in this degree sequence, abbreviated as $\mathcal{SD}(u, \mathcal{G}_S(T))$, is the average degree of u in $\mathcal{DS}(u, \mathcal{G}_S(T))$ while the length of the segment is no less than l, which can be denoted by*

$$\mathcal{SD}(u, \mathcal{G}_S(T)) = \frac{\sum_{i=t_s}^{t_e} |N_u(G_i) \cap S|}{t_e - t_s + 1}, \textit{ in which } t_e - t_s + 1 \geq l$$

Based on Definition 3, the maximum l-segment density of u in $\mathcal{G}_S$ (abbreviated as $\mathsf{MSD}(u, \mathcal{G}_S)$), is the l-segment density $\mathcal{SD}(u, \mathcal{G}_S(T))$ such that there are no $S' \subseteq \mathcal{V}, T' \subseteq [1 : |\mathcal{T}|]$ satisfying $\mathcal{SD}(u, \mathcal{G}_{S'}(T')) > \mathcal{SD}(u, \mathcal{G}_S(T))$.

Below, we give a definition to describe the node which has average degree no less than δ in a time segment of length no less than l in a given temporal subgraph.

Table 1: Main symbols

Symbols	Definitions
$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$	the undirected temporal graph
$G = (V, E)$ and $G_i = (V_i, E_i)$	the de-temporal graph and the i-th snapshot of $\mathcal{G}$
$N_u(G) = \{v \mid (u, v) \in E\}$	the set of neighbor nodes of u in G
$\mathcal{G}_S(T) = (S, \mathcal{E}_S(T), T)$	the temporal subgraph induced by nodes' set S, time interval T
$\mathcal{DS}(u, \mathcal{G}_S(T))(\mathsf{DS}[u])$	the degree sequence of u in $\mathcal{G}_S(T)$
$\mathcal{SD}(u, \mathcal{G}_S(T))$	the l-segment density of u in $\mathcal{G}_S(T)$
$\mathsf{MSD}(u, \mathcal{G}_S)(\mathsf{MSD}[u])$	the maximum l-segment density of u in $\mathcal{G}_S$
(l, δ)-MBC	the (l, δ)-maximal bursting core
$\mathcal{CSC}$ of u	the cumulative sum curve of $\mathcal{DS}(u, \mathcal{G}_C)$
$\mathsf{MTS}[u][t]$	u's maximum slope ended at time t with segment length $\geq l$
$\mathsf{MTS}_{2l}[u][t]$	u's maximum slope ended at t with $l \leq$ segment length $\leq 2l$
POMBC	the Pareto Optimal (l, δ)-MBC in terms of l and δ

Definition 4 ((l, δ)-bursting node). *Given a temporal graph $\mathcal{G}$, an integer l and a real value δ, node u is an (l, δ)-bursting node in $\mathcal{G}$ if $\mathsf{MSD}(u, \mathcal{G}) \geq \delta$.*

Note that, we set a constraint that the length of the considering segment is no less than l. This is because that we want to find the node which has high degree in a continuous time. If we loosen the constraint, some nodes may be highly connected in just one timestamp and we may obtain outliers that are not located in the dense regions of the graph (as shown in Figure 7(c) of Section 5).

According to Definition 4, we introduce a structure which can cluster the (l, δ)-bursting nodes.

Definition 5 ((l, δ)-maximal bursting core). *Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, an integer $l \geq 2$ and a real value $\delta > 0$, an (l, δ)-maximal bursting core (abbreviated as (l, δ)-MBC) is an induced temporal graph $\mathcal{G}_C$ in which $C \subseteq \mathcal{V}$, satisfying*
(i) each node in C is an (l, δ)-bursting node in $\mathcal{G}_C$, which means that $\forall u \in C$, $\mathsf{MSD}(u, \mathcal{G}_C) \geq \delta$ holds.
(ii) there is no subset of nodes $C' \supseteq C$ that satisfies each node in C' is an (l, δ)-bursting node in $\mathcal{G}_{C'}$.

The main symbols in Section 2 and 3 can be seen in Table 1. Below, we use an example to illustrate the above definitions.

Example 1. *Consider the temporal graph in Fig. 1. Given $l = 3, \delta = 3$. As shown in Fig. 1(c), we can get that $\mathcal{DS}(v_5, \mathcal{G}) = [4, 2, 3, 4, 4, 2]$. As $l = 3$, the maximum l-segment density $\mathsf{MSD}(v_5, \mathcal{G}) = (3 + 4 + 4)/3 = 3.66$. Given $S = \{v_1, v_2, v_3, v_4, v_5\}$, we can get that $\mathcal{DS}(v_5, \mathcal{G}_S) = [4, 0, 3, 4, 4, 0]$, $\mathsf{MSD}(v_5, \mathcal{G}_S) = (3 + 4 + 4)/3 = 3.66$. Thus, v_5 is a $(3, 3)$-bursting node in $\mathcal{G}_S$. Considering v_3 in S, we can get that $\mathcal{DS}(v_3, \mathcal{G}_S) = [4, 0, 0, 4, 4, 0]$, $\mathsf{MSD}(v_3, \mathcal{G}_S) = (0+4+4)/3 = 2.66$. So, v_3 is a not $(3, 3)$-bursting node in $\mathcal{G}_S$. Therefore, $\mathcal{G}_S$ is not a $(3, 3)$-MBC. However, given $C = \{v_1, v_2, v_4, v_5\}$, we can find that all the nodes in C are $(3, 3)$-bursting nodes, because all the nodes have the maximum l-segment density of 3 considering $T = [3 : 5]$. So, $\mathcal{G}_C$ is a $(3, 3)$-MBC with $C = \{v_1, v_2, v_4, v_5\}$.* □

Problem 1 (Bursting Core). Given a temporal graph $\mathcal{G}$, an integer $l \geq 2$ and a real value $\delta > 0$, the goal of mining one bursting core is to compute the (l, δ)-MBC in $\mathcal{G}$.

Based on Definition 5, (l, δ)-MBC is a bursting community which can identify important events in the temporal graph, but it may be not easy to find proper parameters of l and δ for practical applications. Intuitively, a good bursting community will have large l and δ values. But large l and δ values may result in losing answers. However, based on the theory of Pareto Optimality, we are able to compute the bursting cores that are not dominated by the other cores in terms of parameters l and δ. Below, we introduce a new concept, POMBC, to define those bursting cores.

Definition 6 (Pareto Optimal (l, δ)-MBC). *Given a temporal graph $\mathcal{G}$, an (l, δ)-MBC in $\mathcal{G}$ is a Pareto Optimal (l, δ)-MBC (abbreviated as POMBC), if there does not exist an (l', δ')-MBC in $\mathcal{G}$ such that $l' > l, \delta' \geq \delta$ or $l' \geq l, \delta' > \delta$.*

Based on Definition 6, each (l, δ)-MBC will be contained in one of the POMBCs since they are maximal. Finding all the POMBCs in a temporal graph helps to set the value of parameters (l, δ) for the (l, δ)-MBC model. The parameter l in our model is indeed as small as possible since the proposed model aims to find a subgraph in which each node has a high average degree of length l. However, in real-world applications, we do not know how to set the proper parameters (l, δ) Because if (l, δ) are large we may get empty results, and if (l, δ) are small we will get too many unnecessary nodes in (l', δ')-MBC. Thus, we introduce the second studied problem below.

Problem 2 (Pareto-Optimal Bursting Core). Given a temporal graph $\mathcal{G}$, the goal of mining Pareto-optimal bursting cores is to enumerate all the POMBCs in $\mathcal{G}$.

Challenges. The problem of mining one bursting core is similar to mining traditional k-core. However, it is not sufficient by adopting the traditional core decomposition method directly, since we need to find the maximum average sequential numbers when peeling the (l, δ)-MBCs. The peeling method iteratively removes the nodes which are not (l, δ)-bursting nodes, and then determines whether the remaining nodes are (l, δ)-bursting nodes until no nodes can be deleted. Therefore, many nodes will be checked whether are (l, δ)-bursting nodes again and again. The time complexity of the naive method to check whether the node is (l, δ)-bursting node for one time is $O(|\mathcal{T}|^2)$ since we need to check all segments with length no less than l. Thus, the status of the node must be checked while one edge is deleted, the times of the checking steps are $O(m)$. So, the whole time complexity is $O(m|\mathcal{T}|^2)$. Clearly, this approach may involve numerous redundant computations for checking some nodes which are definitely not contained in an (l, δ)-MBC.

To list all the POMBCs, the naive method is to enumerate parameter pairs (l, δ) and outputs the one which cannot be dominated. This way is difficult since it is hard to set the proper δ which is a real value. However, another possible way is to consider one dimension, such as l first, and then find the maximal δ. Next, we keep δ unchanged and find the maximal l. So, the challenge is how to acquire the answers with less redundant computations.

3 ALGORITHMS FOR MINING MBC

In this section, we first introduce a basic decomposition framework to mine the (l, δ)-MBC. Next, we develop a dynamic programming algorithm which can compute the segment density efficiently, and then propose an improved algorithm with several novel pruning techniques. Due to the limit of space, all the proofs of the lemmas, corollaries and complexity analysis are described in the supplementary materials.

3.1 The MBC Algorithm

We can observe that (l, δ)-MBC has the following three properties.

Property 1 (Uniqueness). *Given parameters $l > 1$ and $\delta > 0$, the (l, δ)-MBC of the temporal graph $\mathcal{G}$ is unique.*

Proof. We prove this lemma by a contradiction. Suppose that there exist two different (l, δ)-maximal bursting cores in $\mathcal{G}$, denoted by C_1 and C_2 ($C_1 \neq C_2$). Let us consider the node set $C' = C_1 \cup C_2$. Following Definition 5, every node in C' is a (l, δ)-bursting node in $\mathcal{G}(C')$, because it is a (l, δ)-bursting node in $\mathcal{G}_{C_1} \cup \mathcal{G}_{C_2}$. Since $C_1 \neq C_2$, we have $C_1 \subset C'$ and $C_2 \subset C'$ which contradicts to the fact that C_1 (or C_2) satisfies the maximal property. □

Property 2 (Containment). *Given an (l, δ)-*MBC *of the temporal graph $\mathcal{G}$, the (l', δ')-*MBC *with $\delta' \geq \delta$, $l' \geq l$ is a temporal subgraph of (l, δ)-*MBC.

Proof. According to Definition 5, an (l, δ)-MBC C is a maximal temporal subgraph, and any node in C has segment density at least δ with length no less than l. For $\delta' \geq \delta$, $l' \geq l$, each node in (l', δ')-maximal bursting core will also have segment density at least δ with length no less than l. Since the C is a maximal temporal subgraph, (l', δ')-maximal bursting core must be contained in C. □

We first give the definition of k-core, and then show the third property. The **k-core of the de-temporal graph of $\mathcal{G}$** can be denoted by $G_c = (V_c, E_c)$, which is a maximal subgraph such that $\forall u \in G_c : deg_{G_c}[u] \geq k$.

Property 3 (Reduction). *Given an (l, δ)-*MBC *of the temporal graph $\mathcal{G}$, the nodes in (l, δ)-*MBC *must be contained in the k-core $(k = \delta)$ of the de-temporal graph G.*

Proof. According to Definition 5, any node u in an (l, δ)-MBC $\mathcal{G}_C$ has segment density at least δ with length no less than l ($l \geq 2$). So, u must have degree at least δ in at least one snapshot G^*. As each $G^* \subseteq G$, each u in C must have degree no less than δ. Since the k-core $(k = \delta)$ of the de-temporal graph G is the maximal subgraph such that each node has degree no less than δ, C must be contained in the k-core $(k = \delta)$ of G. □

Core Decomposition Framework. Following Property 3, we first compute the k-core $(k = \delta)$ of the de-temporal graph of $\mathcal{G}$, denoted by G_c. Given the properties of *Uniqueness* and *Containment*, we can apply the core decomposition framework to compute the (l, δ)-MBC. Next, we check whether node u is an (l, δ)-bursting node, as defined in Definition 4. Specifically, we compute the k-core G_c in G first, and then check whether node u is an (l, δ)-bursting node for all $u \in G_c$. If u is not an (l, δ)-bursting node, we delete u from the results. Since the deletion of u may result in u's neighbors no longer being the (l, δ)-bursting node, we need to iteratively process u's neighbors. The process terminates if no node can be deleted. The details are provided in Algorithm 1.

Algorithm 1 first computes the k-core $(k = \delta)$ of the de-temporal graph G (lines 1-2), denoted by $G_c = (V_c, E_c)$. Then, it initializes a queue Q to store the nodes to be deleted, a set D to store the deleted node, a collection MSD to store maximum l-segment density for each node (line 3) and deg to store the degree of nodes in G_c (line 5). Next, for each u in V_c, it invokes Algorithm 2 to check whether u is an (l, δ)-bursting node or not (line 6). If u's maximum l-segment density $\mathsf{MSD}[u]$ is less than δ, u is not an (l, δ)-bursting node and it will be pushed into a queue Q (lines 6-7). Subsequently, the algorithm iteratively processes the nodes in Q. In each iteration, the algorithm pops a node v from Q and uses D to maintain all the deleted nodes (line 9). For each neighbor node w of v, the algorithm updates $deg[w]$ (line 11). If the revised $deg[w]$ is smaller than δ, w is

Algorithm 1: $\mathsf{MBC}(\mathcal{G}, l, \delta)$

```
   Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, parameters $l$ and $\delta$
   Output: $(l, \delta)$-MBC in $\mathcal{G}$
1  Let $G = (V, E)$ be the de-temporal graph of $\mathcal{G}$;
2  Let $G_c = (V_c, E_c)$ be the $k$-core $(k = \delta)$ of $G$;
3  $Q \leftarrow [\emptyset]; D \leftarrow [\emptyset]; \mathsf{MSD} \leftarrow [\emptyset]$;
4  for $u \in V_c$ do
5      $deg[u] \leftarrow |N_u(G_c)|$;        /* compute the degree of all nodes in $G_c$ */
6      $\mathsf{MSD}[u] \leftarrow \mathsf{ComputeMSD}(\mathcal{G}, l, u, V_c)$;
7      if $\mathsf{MSD}[u] < \delta$ then $Q.push(u)$;
8  while $Q \neq [\emptyset]$ do
9      $v \leftarrow Q.pop()$; $D \leftarrow D \cup \{v\}$;
10     for $w \in N_v(G_c)$, s.t. $deg[w] \geq \delta$ and $\mathsf{MSD}(w) \geq \delta$ do
11         $deg[w] \leftarrow deg[w] - 1$;
12         if $deg[w] < \delta$ then $Q.push(w)$;
13         else
14             $\mathsf{MSD}[w] \leftarrow \mathsf{ComputeMSD}(\mathcal{G}, l, w, V_c \setminus D)$;
15             if $\mathsf{MSD}[w] < \delta$ then $Q.push(w)$;
16 return $\mathcal{G}_{V_c \setminus D}$ ;
```

clearly not an (l, δ)-bursting node. As a consequence, the algorithm pushes w into Q which will be deleted in the next iterations (line 12). Otherwise, the algorithm invokes Algorithm 2 to determine whether w is an (l, δ)-bursting node (lines 14-15). The algorithm terminates when Q is $\emptyset$. At this moment, the remaining nodes $V_c \setminus D$ is the (l, δ)-bursting nodes of $\mathcal{G}$, and the algorithm returns temporal subgraph $\mathcal{G}_{V_c \setminus D}$ (line 16).

Example 2. *Recall the temporal graph in Fig. 1. Given $l = 3$, $\delta = 3$. Algorithm 1 first computes the k-core $(k = \delta)$ of de-temporal graph G. So, $V_c = \{v_1, v_2, v_3, v_4, v_5\}$. Then, for each node u in V_c, it checks whether u is an (l, δ)-bursting node. Consider v_3, we can get $\mathcal{DS}(v_3, \mathcal{G}_{V_c}) = [4, 0, 0, 4, 4, 0]$, and cannot find a segment of at least 3 length in which the density is no less than 3. Next, v_3 will be pushed into Q. In line 9, v_3 is added into set D and all of its neighbors will be checked in line 10. Now the remained nodes are $\{v_1, v_2, v_4, v_5\}$, and we can find that the deg and* MSD *of them are no less than 3. Therefore, Algorithm 1 returns $\mathcal{G}_{V_c \setminus D}$ with $V_c \setminus D = \{v_1, v_2, v_4, v_5\}$.* □

Complexity of Algorithm 1. The time and space complexity of Algorithm 1 by invoking Algorithm 2 to compute MSD is $O(m|\mathcal{T}|)$ and $O(m)$ respectively.

Different from the traditional core decomposition algorithm, Algorithm 1 needs to check whether the node is an (l, δ)-bursting node in each iteration. Below, the implementation details of ComputeMSD are described.

3.2 Dynamic Programming of Computing MSD

Recall that by Definition 4 and 5, if $\mathsf{MSD}(u, \mathcal{G}_C) \geq \delta$, then u is an (l, δ)-bursting node. To compute MSD, we first get u's degree sequence $\mathcal{DS}(u, \mathcal{G}_C)$ inside the candidate (l, δ)-MBC for $i \in [1 : |\mathcal{T}|]$. For convenience, we denote $\mathcal{DS}(u, \mathcal{G}_C)$ by $\mathsf{DS}[u] = \{|N_u(G_i) \cap C|, i \in [1 : |\mathcal{T}|]\}$, $\mathsf{MSD}(u, \mathcal{G}_C)$ by $\mathsf{MSD}[u]$. To get $\mathsf{MSD}[u]$, the naive method is to consider all the segments of longer than l, but the time complexity of such a naive method is $O(|\mathcal{T}|^2)$. Below, we propose a **dynamic programming** algorithm that transforms the problem into finding the maximum slope in a cumulative sum curve, which can reduce the computational overhead to linear complexity.

Definition 7 (cumulative sum curve). *Given node u's degree sequence $\mathcal{DS}(u, \mathcal{G}_C)$ (abbreviated as* $\mathsf{DS}[u]$*), the i-th item in the* *cumulative sum curve of u (abbreviated as $CSC[i]$) is the sum of front i-th position of* $\mathsf{DS}[u]$*, i.e.* $CSC[i] = \sum_1^i \mathsf{DS}[u][i], i \in [1 : |\mathcal{T}|]$.

Without loss of generality, we set $CSC[0] = 0$. Then, the points $\{(0, CSC[0]), (1, CSC[1]) \ldots (|\mathcal{T}|, CSC[|\mathcal{T}|])\}$ can be drawn as a curve in the Cartesian Coordinate System, and we denote this curve by CSC. For example in Fig. 2(a), we can see $\mathsf{DS}[u] = [4, 2, 3, 4, 4, 2, 2, 6, 1]$ and the CSC sequence is $[0, 4, 6, 9, 13, 17, 19, 21, 27, 28]$. Below, we define the *slope* in the curve by considering two points in CSC.

Definition 8 (slope). *Given integers* $i, j \in [1 : |\mathcal{T}|], i < j$*, the slope of curve CSC from i to j can be denoted by* $\mathsf{slope}(i, j, CSC) = \frac{CSC[j]-CSC[i-1]}{j-i+1}$*, where i, j are the start and end of the slope.*

For convenience, we abbreviate $\mathsf{slope}(i, j, CSC)$ as $\mathsf{slope}(i, j)$ in the following paper while the symbol CSC can not be confused.

Lemma 1. *For a degree sequence* $\mathsf{DS}[u]$*, a time interval* $[t_s : t_e]$*, the segment density of the subsequence in* $[t_s : t_e]$ *equals the slope of curve CSC from t_e to t_s. Formally,* $\frac{\sum_{i=t_s}^{t_e} \mathsf{DS}[u][i]}{t_e - t_s + 1} = \mathsf{slope}(t_s, t_e)$.

Definition 9 (maximum t-truncated l-slope). *Given a curve CSC of node u, a truncated time* $t \in [l : |\mathcal{T}|]$*, the maximum t-truncated l-slope* $\mathsf{MTS}[u][t] = \{\max(\mathsf{slope}(i, t)) | i \in [0 : t - l]\}$.

According to Lemma 1 and Definition 9, $\mathsf{MTS}[u][t]$ is the maximum slope which ended at time t and the length of the corresponding segment is no less than l. For convenience, $\mathsf{MTS}[u]$ is the collection of $\{\mathsf{MTS}[u][t] | t \in [l : |\mathcal{T}|]\}$.

Corollary 1. *The problem of finding the* $\mathsf{MSD}[u]$*, can be transformed to computing* $\max(\mathsf{MTS}[u])$ *in CSC of u.*

Next, the problem is how to compute all the $\mathsf{MTS}[u][t]$ with $t = [1 : |\mathcal{T}|]$. One idea is to maintain $\mathsf{MTS}[u][t+1]$ by the computed $\mathsf{MTS}[u][t]$ and the changes of the curve from time t to $t + 1$. Below, considering the computed $\mathsf{MTS}[u][t]$ and the newly joined point $(t+1, CSC[t + 1])$, we can maintain $\mathsf{MTS}[u]$ based on the following observations.

Observation 1. *We can compute a lower convex hull (abbreviated as* CH*) in CSC of u which ended at time $t - l$, the slope of the tangent from point $(t, CSC[t])$ to the* CH *is the maximum l-segment density of node u ended at time t.*

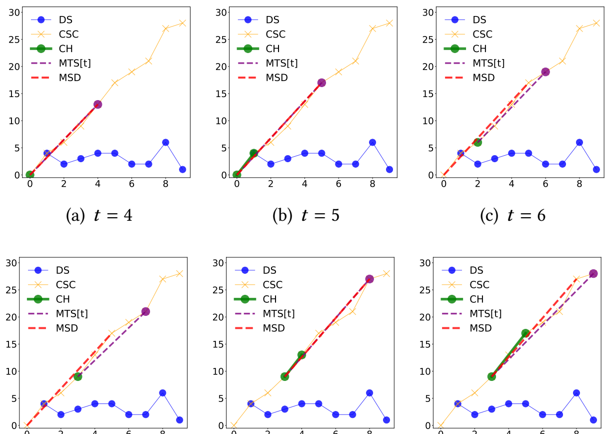
Observation 2. *If the point $(a, CSC[a])$ and $(b, CSC[b])$ are on the maintained lower convex hull, suppose that $a < b < c$,* CH *will add node $(c, CSC[c])$ and remove node $(b, CSC[b])$ if $(CSC[c] - CSC[b])/(c - b) \le (CSC[b] - CSC[a])/(b - a)$.*

Observation 3. *For one ended time t, if* $\frac{CSC[t]-CSC[b]}{t-b} \ge \frac{CSC[b]-CSC[a]}{b-a}$*, then the slope of $CSC[t]$ to $CSC[a]$ will not be maximum, and node $(a, CSC[a])$ should be removed from $\mathcal{CH}$.*

Following the observations above, we devise an algorithm to maintain the lower convex hull CH ended at time $t - l$, and the $\mathsf{MTS}[u][t]$ can be computed in a recursive way as the following algorithm shows.

Algorithm of Computing MSD**.** Algorithm 2 first initializes $CSC[t]$ of u for all timestamps (lines 1-5). As the nodes set C may be changed in Algorithm 1, the degree of u can be computed in line 4. Next, it maintains an array CH to record the indexes of each points in the lower convex hull, i_s to record the *start* index of CH, i_e to record the *end* index of CH and $\mathsf{MTS}[u]$ to record MTS (line 6). For time t from l to $|\mathcal{T}|$, it dynamically computes $\mathsf{MTS}[u][t]$ of u (lines 7-13). In lines 8-9, i_e reduces by 1 if the $\mathsf{slope}(\mathsf{CH}[i_e], t - l)$ is no larger than $\mathsf{slope}(\mathsf{CH}[i_e - 1], \mathsf{CH}[i_e])$, because the rear node point will be above the convex hull CH by the end of $t - l$ following Observation 2. If there is no such point in the end, the *end* index i_e increases by 1 and $\mathsf{CH}[i_e]$ is assigned by $t - l$. In lines 11-12, the head index adds up by 1 if $\mathsf{slope}(\mathsf{CH}[i_s], t)$ is no larger than $\mathsf{slope}(\mathsf{CH}[i_s], \mathsf{CH}[i_s + 1])$, because it will have an upper convex hull in the curve of CH at the start of $\mathsf{CH}[i_s]$ according to Observation 3. We will get a array MTS of $\mathsf{MTS}[u][t]$ with t ranges from l to $|\mathcal{T}|$. Finally, it returns $\max(\mathsf{MTS}[u])$ after all the iterations (line 14).

Example 3. *Fig. 2 shows the toy example of computing maximum l-segment density for u's degree sequence of* $[4, 2, 3, 4, 4, 2, 2, 6, 1]$ *with* $l = 4$*. Clearly,* $\mathcal{T} = [1 : 9]$, $CSC = [0, 4, 6, 9, 13, 17, 19, 21, 27, 28]$*. According to Corollary 1, the procedure starts at $t = 4$ because we need satisfy that the length of the segment is no less than l. At this time,*



(a) $t = 4$ (b) $t = 5$ (c) $t = 6$
(d) $t = 7$ (e) $t = 8$ (f) $t = 9$

Figure 2: Running example of computing maximum l-segment density for a degree sequence of $[4, 2, 3, 4, 4, 2, 2, 6, 1]$ **with $l = 4$. The data on x-axis and y-axis represent the time t and the cumulative sum.**

Algorithm 2: $\mathsf{ComputeMSD}(\mathcal{G}, l, u, C)$

```
1  $CSC \leftarrow [\emptyset]$; $CSC[0] \leftarrow 0$; $\mathcal{DS}[u] \leftarrow [\emptyset]$ ;
2  for $t \leftarrow 1 : |\mathcal{T}|$ do
3      Let $G_t$ be the snapshot of $\mathcal{G}$ at timestamp $t$;
4      $\mathsf{DS}[u][t] \leftarrow |N_u(G_t) \cap C|$;
5      $CSC[t] = CSC[t-1] + \mathsf{DS}[u][t]$;
6  $\mathsf{CH} \leftarrow [\emptyset]$, $i_s \leftarrow 0$, $i_e \leftarrow -1$, $\mathsf{MTS}[u] \leftarrow [\emptyset]$;
7  for $t \leftarrow l : |\mathcal{T}|$ do
8      while $i_s < i_e$ and $\mathsf{slope}(\mathsf{CH}[i_e], t-l, CSC) \le$
         $\mathsf{slope}(\mathsf{CH}[i_e - 1], \mathsf{CH}[i_e], CSC)$ do
9          $i_e \leftarrow i_e - 1$;
10     $i_e \leftarrow i_e + 1$; $\mathsf{CH}[i_e] \leftarrow t - l$;
11     while $i_s < i_e$ and $\mathsf{slope}(\mathsf{CH}[i_s], t, CSC) \ge$
         $\mathsf{slope}(\mathsf{CH}[i_s], \mathsf{CH}[i_s + 1], CSC)$ do
12         $i_s \leftarrow i_s + 1$;
13     $\mathsf{MTS}[u] \leftarrow \mathsf{MTS}[u] \cup \{\mathsf{slope}(\mathsf{CH}[i_s], t, CSC)\}$;
14 return $\max(\mathsf{MTS}[u])$;

15 Procedure $\mathsf{slope}(i, j, CSC)$
16 return $(CSC[j] - CSC[i])/(j - i)$
```

there is only one item in CH. *When* $t = 5$, *the* i_e *index of* CH *adds up by* 1 *(line 10), but the* i_s *index is remained* 0 *because* $\mathsf{slope}(0,5) = (17-0)/(5-0) = 3.4$ *is no larger than* $\mathsf{slope}(0,1) = (4-0)/(1-0) = 4.0$ *(lines 12). And* $\max(\mathsf{MTS}[u][t])$ *is currently* $\mathsf{MTS}[u][5] = 3.4$. *Next,* $t = 6$, *according to Observation 2, the* i_e *index of* CH *reduces by* 1 *because* $\mathsf{slope}(1,2) = 2.0$ *is no larger than* $\mathsf{slope}(0,1) = 4.0$ *(lines 8-9). Then, the newly* i_e *is* 1 *and* $\mathsf{CH}[i_e]$ *is assigned by* $t - l = 2$ *(line 10). Now* CH *is* $[0,2]$, $i_s = 0, i_e = 1$. *In the next step, the* i_s *index adds up by* 1 *because* $\mathsf{slope}(0,6) = 19/6 > \mathsf{slope}(0,2) = 6/2$ *(line 12). So, the final* CH *and* $\mathsf{MTS}[u][6]$ *can be shown at Fig. 2(c). Likewise, when* $t = 7$ *to* 9, *the* CH *will be maintained by the similar processes. It should be noted that when* $t = 7$, $\mathsf{slope}(3,8) = 3.6$, *which is larger than* $\mathsf{MTS}[u][5]$. *Finally,* $\mathsf{MSD}[u] = \max(\mathsf{MTS}[u][t]) = 3.6$, *which is the density of the* 4*th to* 8*th items* $[4,4,2,2,6]$. □

Complexity of Algorithm 2. For a temporal graph $\mathcal{G}$ with $|\mathcal{T}|$ timestamps, the time and space complexity of Algorithm 2 are $O(|\mathcal{T}|)$ and $O(|\mathcal{T}|)$ respectively.

3.3 An improved MBC+ algorithm

Although Algorithm 1 is efficient in practice, it still has two limitations. ***(i)*** It still needs to call ComputeMSD procedure for all nodes in $\overline{V_c}$ (line 6 in Algorithm 1). In the worst case, the time complexity of this process can be near to $|\mathcal{T}|m$. We can observe that if we delete a certain node u, the $deg[v]$ of u's neighbor v will reduce, and we can monitor it at once to check whether $deg[v] < \delta$. Once $deg[v] < \delta$, we do not need to call the procedure ComputeMSD for v any more. ***(ii)*** It still needs to compute all the maximum l-segment density dynamically for each deletion of the edges. We can observe that in each call of ComputeMSD, the degree of u reduces only one and $\mathsf{MSD}[u]$ may not change. So, the ComputeMSD algorithm clearly results in significant amounts of redundant computations for the iterations for all t from l to $|\mathcal{T}|$.

To overcome this limitation, we propose an improved algorithm called MBC+. The striking features of MBC+ are twofold. On one hand, it needs not to call procedure ComputeMSD for each node in advance. Instead, it calculates SD of the candidate node on-demand. On the other hand, when deleting a node u, MBC+ does not re-compute MSD for a neighbor node w of u. Instead, MBC+ dynamically updates the computed MSD for each node w, thus substantially avoiding redundant computations. The detailed description of MBC+ is shown in Algorithm 3.

To Overcome Limitation *(i)*. Algorithm 3 first computes the k-core ($k = \delta$) G_c in the de-temporal graph (line 2). Next, it explores the nodes in V_c based on an increasing order by the degrees in G_c (line 5). When processing a node u, the algorithm first checks whether u has been deleted or not (line 6). If u has not been removed, MBC+ invokes Algorithm 2 to compute $\mathsf{MSD}[u]$ (lines 7-8). It should be noted that ComputeMSD* is all the same to ComputeMSD except that it returns $(\mathsf{MTS}[w], \mathcal{DS}[u])$ (replace line 14 of Algorithm 2). Next, if $\mathsf{MSD}[u]$ is no larger than δ, u is not an (l,δ)-bursting node. Thus, the algorithm pushes u into the queue Q (line 9). Subsequently, the algorithm iteratively deletes the nodes in Q (lines 10-19). When removing a node v, MBC+ explores all v's neighbors (line 12). For a neighbor node w, MBC+ first updates the degree of w (line 13), i.e., $deg[w]$. If the updated degree is less than δ, w is not an (l,δ)-bursting node (line 14). In this case, the algorithm pushes it into Q and continues to process

Algorithm 3: $\mathsf{MBC+}(\mathcal{G}, l, \delta)$

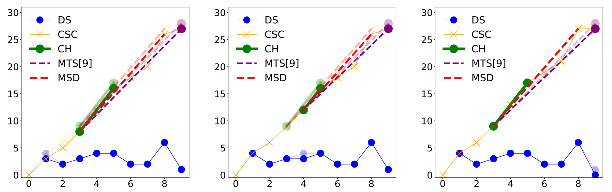
```
   Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, parameters $l$ and $k$
   Output: $(l,\delta)$-MBC in $\mathcal{G}$
1  Let $G = (V, E)$ be the de-temporal graph of $\mathcal{G}$;
2  Let $G_c = (V_c, E_c)$ be the $k$-core ($k = \delta$) of $G$;
3  Let $deg[u]$ be the degree of $u$ in $G_c$;
4  $Q \leftarrow [\emptyset]; D \leftarrow [\emptyset]; \mathsf{MSD} \leftarrow [\emptyset]; \mathsf{MTS} \leftarrow [\emptyset]; \mathcal{DS} \leftarrow [\emptyset]$;
5  for $u \in V_c$ in an increasing order by $deg[u]$ do
6      if $u \in D$ then continue;
7      $(\mathsf{MTS}[u], \mathcal{DS}[u]) \leftarrow \mathsf{ComputeMSD}^*(\mathcal{G}, l, u, V_c \setminus D)$;        /*
         all the same to Alg. 2 except that it returns $(\mathsf{MTS}[u], \mathcal{DS}[u])$ */
8      $\mathsf{MSD}[u] \leftarrow \max(\mathsf{MTS}[u])$;
9      if $\mathsf{MSD}[u] < \delta$ then {$Q.push(u); deg[u] \leftarrow 0$;}
10     while $Q \neq \emptyset$ do
11         $v \leftarrow Q.pop(); D \leftarrow D \cup \{v\}$;
12         for $w \in N_v(G_c) \setminus D$, s.t. $deg[w] \geq \delta$ do
13             $deg[w] \leftarrow deg[w] - 1$;
14             if $deg[w] < \delta$ then {$Q.push(w)$; continue;}
15             if $\mathsf{MSD}[w]$ is not existed then continue;
16             for $t$, s.t. $(v, w, t) \in \mathcal{E}$ do
17                 $\mathcal{DS}[w][t] \leftarrow \mathcal{DS}[w][t] - 1$;
18                 $\mathsf{MSD}[w] \leftarrow \mathsf{UpdateMSD}(w, t, l, \mathcal{DS}, \mathsf{MTS})$;
19             if $\mathsf{MSD}[w] < \delta$ then {$Q.push(w); deg[w] \leftarrow 0$;}
20 return $\mathcal{G}_{V_c \setminus D}$;

21 Procedure UpdateMSD$(w, t, l, \mathcal{DS}, \mathsf{MTS})$
22 $CSC \leftarrow [\emptyset]; t_s \leftarrow \max(0, t - 2l); t_e \leftarrow \min(t + 2l, |\mathcal{T}|); CSC[0] \leftarrow 0$;
23 for $i \leftarrow 0 : t_e - t_s$ do
24     $CSC[i+1] = CSC[i] + \mathcal{DS}[w][t_s + i]$;
25 $\mathsf{CH} \leftarrow [\emptyset], i_s \leftarrow 0, i_e \leftarrow -1$;
26 for $j \leftarrow l : t_e - t_s + 1$ do
27     while $i_s < i_e$ and $\mathsf{slope}(\mathsf{CH}[i_e], j - l, CSC) \leq$
         $\mathsf{slope}(\mathsf{CH}[i_e - 1], \mathsf{CH}[i_e], CSC)$ do
28         $i_e \leftarrow i_e - 1$;
29     $i_e \leftarrow i_e + 1; \mathsf{CH}[++i_e] \leftarrow t - l$;
30     while $i_s < i_e$ and $\mathsf{slope}(\mathsf{CH}[i_s], j, CSC) \geq$
         $\mathsf{slope}(\mathsf{CH}[i_s], \mathsf{CH}[i_s + 1], CSC)$ do
31         $i_s \leftarrow i_s + 1$;
32     if $j \geq t - t_s$ then
33         $\mathsf{MTS}[w][j + t_s - l] \leftarrow \mathsf{slope}(\mathsf{CH}[i_s], j, CSC)$;
34 return $\max(\mathsf{MTS}[w])$;
```

the next node in Q (the degree pruning rule). Otherwise, if $\mathsf{MSD}[w]$ has already been computed, the algorithm invokes UpdateMSD to update $\mathsf{MSD}[w]$ (line 19). If the updated $\mathsf{MSD}[w]$ is less than δ, w is not an (l,δ)-bursting node and the algorithm pushes w into Q (line 19). We can see that if $\mathsf{MSD}[w]$ has not been computed yet, the algorithm does not need to update $\mathsf{MSD}[w]$. In this case, $\mathsf{MSD}(w)$ will be calculated in the next iterations of line 7. It also should be noted that the $\mathcal{DS}$ is always updated, because if the nodes have been deleted by the degree constraint, $\mathcal{DS}$ will be newest in $\mathcal{G}_{V_c \setminus D}$ (line 7), otherwise if the nodes have been deleted by the (l,δ)-dense constraint, $\mathcal{DS}$ will be updated in line 17. Finally, MBC+ outputs $\mathcal{G}_{V_c \setminus D}$ as the result.

To Overcome Limitation *(ii)*. In the following, we introduce the UpdateMSD procedure. Suppose that before updating, the maximum l-segment density of w exists from time t_s to t_e. At this time, if $\mathcal{DS}[w][t']$ reduces by 1, then there exist three situations: (i) $t' < t_s$; (ii) $t_s \leq t' \leq t_e$; (iii) $t' > t_e$.

Example 4. *Fig. 3 shows the three situations of Fig. 2(f) after* $\mathcal{DS}[w][t']$ *reduces by 1. We can see that the current maximum* l*-segment density of* w *exists from* $t_s = 3$ *to* $t_e = 8$. *As shown in Fig. 3(a) in which* $t' < t_s$ *and Fig. 3(c) in which* $t' > t_s$, *we can see that the* $\mathsf{MSD}[w]$ *will not change. We can find that the parts of curve with the maximum slop are all moved down. Also, it can be proved easily from the definition of* l*-segment density that* $\mathsf{MSD}[w]$ *will not*



(a) $t' = 1$ (b) $t' = 4$ (c) $t' = 9$

Figure 3: Updated situations of Fig. 2(f) after $\mathcal{DS}[w][t']$ reduces by 1. The data on x-axis and y-axis represent the time t and the cumulative sum.

change. Howerver, in Fig. 3(b), $\mathcal{DS}[w][4]$ reduces by 1 and the new sequence is $[3, 2, 3, 3, 4, 2, 2, 6, 1]$. The maximum l-segment density is 3.5, which is the density of the 5th to 8th items $[4, 2, 2, 6]$. So, only when $t_s \le t' \le t_e$ should we update the MSD. □

Below we will introduce that it only needs to consider $\mathcal{DS}$ from time $t - 2l$ to time $t + 2l$ to update MSD. We first define a concept, $\text{MTS}_{2l}[u][j]$, which is a maximum j-truncated l-slope of considering only $2l$ length of the curve $\mathcal{CSC}$ of node u.

Definition 10 (maximum t-truncated $(l$-$2l)$-slope). *Given a curve $\mathcal{CSC}$ of node u by Definition 7, a truncated time $t \in [l : |\mathcal{T}|]$, the maximum i-lower t-truncated $(l$-$2l)$-slope $\text{MTS}_{2l}[u][t] = \{\max(\text{slope}(i, t))|i = [j - 2l : j - l]\}$.*

Based on Definition 10, $\text{MTS}_{2l}[u][t]$ is the maximum slope which only considers $\text{MTS}[u][j]$ with the slope ends at j and starts in $[j - 2l : j - l]$. For convenience, $\text{MTS}_{2l}[u]$ is the collection of $\{\text{MTS}_{2l}[u][t]|t \in [l : |\mathcal{T}|]\}$. Furthermore, it holds the property below.

Lemma 2. $\text{MSD}(u, \mathcal{G}_C) = \max(\text{MTS}[u]) = \max(\text{MTS}_{2l}[u])$ *holds for a given curve $\mathcal{CSC}$ of u.*

Corollary 2. *According to Lemma 2,* $\text{MSD}(u, \mathcal{G}_C) = \max(\text{MTS}[u])$. *If $\mathcal{DS}[u]$ reduces by 1 at time t, we only need to update* $\text{MTS}[u][t'] = \text{MSD}[u][t']$ *with $t' \in [t : t + 2l]$ to get the updated* $\text{MSD}(u, \mathcal{G}_C)$.

Corollary 3. *If $\mathcal{DS}[u]$ reduces by 1 at time t, we only need to use $\mathcal{DS}[u][t']$ with $t' \in [\max(0, t-2l) : \min(t+2l, |\mathcal{T}|)]$ to update* $\text{MTS}[u]$, *and then get the updated* $\text{MSD}(u, \mathcal{G}_C)$.

According to the above corollaries, the UpdateMSD procedure first initializes t_s as the left side of the considered time interval, t_e as the right side and $\mathcal{CSC}$ based on Definition 7 (lines 22-24). The following step is aimed at computing all the $\text{MTS}_{2l}[w][j]$ which ends at time j and starts from time $j - 2l$ to $j - l$. The following process is much same as that in Algorithm 2 (lines 27-31). Note that, we use $\text{MTS}[w][j]$ to record $\text{MTS}_{2l}[w][j]$ of node w and it should be updated only when $j \ge t - t_s$ (line 32). After all the $\text{MTS}[w][j]$ with j from t to t_s have been maintained, the procedure returns $\max(\text{MTS}[w])$ as the updated $\text{MSD}[w]$ (line 34).

Lemma 3. *For a temporal graph G with $|\mathcal{T}|$ timestamps, procedure* UpdateMSD *need $O(l)$ to maintain the maximum l-segment density.*

Complexity of Algorithm 3. The time and space complexity of Algorithm 3 are $O(\alpha|\mathcal{T}| + \beta l)$ and $O(\alpha|\mathcal{T}| + m)$ respectively, where $\alpha = |V_c|, \beta = |E_c|$ are number of nodes and edges in k-core ($k = \delta$) of G.

4 ALGORITHMS FOR MINING POMBCS

Finding all POMBCs in a temporal network helps us determine whether there exists an (l', δ')-MBC by arbitrarily given l', δ'. The parameter l in our model should be as small as possible, since a good bursting community is often with a large δ and small l. However, in real-world applications, we do not know how to set proper parameters l', δ' to mine such good bursting communities. Because if l', δ' are too large we may get empty results, and if l', δ' are too small, we then will obtain too many unnecessary nodes in the (l, δ)-MBC. Therefore, it is meaningful to finding all POMBCs efficiently, since they can show whether there exists an (l', δ')-MBC by arbitrarily given l', δ'.

In this section, we develop an efficient algorithm to record all POMBCs. The basic idea of our algorithm is as follows. The algorithm first only considers the l dimension, and computes the maximal $\widehat{\delta}$, among all the (l, δ)-maximal bursting cores. Then, the algorithm considers the δ dimension with $\delta = \widehat{\delta}$ to compute the currently maximal l' value. Using the above method, we can find one POMBC which has the maximal (l, δ) value of all the skyline cores. The challenge is how to find the other POMBC iteratively. We can tackle this challenge based on the following results.

Lemma 4. *Let $(l', \widehat{\delta})$-MBC be a POMBC which has the largest $\widehat{\delta}$ among all the POMBCs, if the node is not a (l, δ)-bursting node with $l > l', \delta > 0$, it can not be contained in another POMBC.*

Lemma 5. *Let (l', δ')-MBC be a POMBC. If $l^* > l'$ and (l^*, δ^*)-MBC is another POMBC, (l^*, δ^*)-MBC must be contained in an induced temporal subgraph from k-core of G in which $k = \frac{\delta' \times l'}{l^*}$.*

Based on Lemma 4 and 5, after computing one POMBC (l, δ)-MBC, as l is integer, we can initialize $l' = l + 1$ to get the next POMBC. Furthermore, we can reduce the considering graph by the following corollary.

Corollary 4. *Let (l, δ)-MBC and (l', δ')-MBC be two POMBCs. If $l' > l$, then nodes in (l', δ')-MBC must be contained in a k-core of G in which $k = \frac{\delta \times l}{l+1}$.*

The detail of the POMBC algorithm is shown as follows. First, Algorithm 4 initializes $l = 2, \delta = 0$ to be default, R to store the result and C to be the nodes of the considered bursting nodes (line 2). Then, the algorithm considers the l dimension and grows l to find all the POMBCs. Next, it computes $\text{MSD}[u]$ and $deg[u]$ in the induced graph from nodes C (lines 4-7). By the given l, the MaxDelta algorithm finds the maximal δ and the corresponding core nodes (line 8). Next, given one maximal δ, the MaxL algorithm finds the maximal l and the final C (line 9). The induced temporal subgraph of C from $\mathcal{G}$ is a POMBC and $(l, \delta, \mathcal{G}_C)$ is recorded as a result (line 10). Based on Corollary 4, in the iteration of $l \longleftarrow l+1$, the new POMBC must be contained in a induced temporal subgraph from k-core of G in which $k = \frac{\delta \times l}{l+1}$, so C is updated as V_c for next loop(lines 10-11). The iterations will terminate when l is increased to $|\mathcal{T}|$ (line 3).

Procedure MaxDelta describes the process of finding the largest δ by parameter l. It is a loop until all the nodes have been deleted (line 15). The algorithm maintains Q to be the deleting queue and D to be the deleted nodes. Specifically, $\overline{\delta}$ and δ are assigned to the minimal and second minimal value of $\text{MSD}[u]$ for $u \in V^*$ (line 16). Then, the nodes are deleted if $deg[w] < \delta$ or $\text{MSD}[w] < \delta$,

Algorithm 4: POMBC($\mathcal{G}$)

```
    Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$
    Output: POMBCs in $\mathcal{G}$
1   Let $G = (V, E)$ be the de-temporal graph of $\mathcal{G}$;
2   $l \leftarrow 2; \delta \leftarrow 0; R \leftarrow [\emptyset]; C \leftarrow V$;
3   while $l \le |\mathcal{T}|$ do
4       for $u \in C$ do
5           $(\mathsf{MTS}[u], \mathcal{DS}[u]) \leftarrow \mathsf{ComputeMSD}^*(\mathcal{G}, l, u, C)$;
6           $\mathsf{MSD}[u] \leftarrow \max(\mathsf{MTS}[u])$;
7           $deg[u] \leftarrow |N_u(G) \cap C|$;
8       $(\delta, C) \leftarrow \mathsf{MaxDelta}(\mathcal{G}, l, C, \mathcal{DS}, \mathsf{MTS}, \mathsf{MSD}, deg)$;
9       $(l, C) \leftarrow \mathsf{MaxL}(\mathcal{G}, l+1, \delta, C, deg)$;
10      $R \leftarrow R \cup (l, \delta, \mathcal{G}_C)$;
11      Let $G_c = (V_c, E_c)$ be the $k$-core ($k = \frac{\delta \times l}{l+1}$) of $G$;
12      $C \leftarrow V_c; l \leftarrow l + 1$;
13  return $R$;

14  Procedure $\mathsf{MaxDelta}(\mathcal{G}, l, V^*, \mathcal{DS}, \mathsf{MTS}, \mathsf{MSD}, deg)$
15  while True do
16      $Q \leftarrow [\emptyset]; D \leftarrow [\emptyset]; \overline{\delta} \leftarrow \min(\mathsf{MSD}); \delta \leftarrow 2nd \min(\mathsf{MSD})$;
17      for $u \in V^*$ do
18          Lines 6-19 in Algorithm 3;
19      if $D \neq V^*$ then
20          $V^* \leftarrow V^* \setminus D$; for $u \in D$ do $\mathsf{MSD}[u] \leftarrow \emptyset$ ;
21      else return $(\overline{\delta}, V^*)$;

22  Procedure $\mathsf{MaxL}(\mathcal{G}, l, \delta, V^*, deg)$
23  while $l \le |\mathcal{T}|$ do
24      $Q \leftarrow [\emptyset]; D \leftarrow [\emptyset]; \mathsf{MSD} \leftarrow [\emptyset]; \mathsf{MTS} \leftarrow [\emptyset]$;
25      for $u \in V^*$ do
26          Lines 6-19 in Algorithm 3.
27      if $D \neq V^*$ then
28          $V^* \leftarrow V^* \setminus D$;
29          if $l = |\mathcal{T}|$ then return $(l, V^*)$;
30          $l \leftarrow l + 1$;
31      else return $(l, V^*)$;
```

which is similar to the process in Algorithm 3 (lines 17-18). Next, if the deleted nodes set D is not equal to the remained nodes set V^*, the remained V^* is updated by $V^* \setminus D$ and MSD will pop all the $\mathsf{MSD}[u]$ for u in the deleted nodes' set D (lines 19-20). Else, if $D = V^*$, then the remained nodes V^* will have maximal $\overline{\delta}$ (lines 21). Furthermore, procedure MaxL can use the remained nodes set of MaxDelta and the known maximal δ to find the maximal l. It grows l to find the largest l and it will terminate if l increases to $|\mathcal{T}|$ (line 23). The unsatisfying nodes are deleted same as that in Algorithm 3 (lines 25-26). MaxL ends at the first time when all the V^* will be deleted or $l = |\mathcal{T}|$, it returns the maximal l and the remained nodes set V^* (lines 27-31).

Speed-up strategies. Although the core reduction pruning in line 11 of Algorithm 4 reduces the temporal graph into a smaller size, the algorithm is still not efficient. It is because when l is small and δ is large, we need to try to compute the $\min(\mathsf{MSD})$ and $2nd \min(\mathsf{MSD})$ in line 16 for lots of time. When l is large but δ is small, the pruning strategy in line 26 is not powerful. To solve these problems, we propose two speed-up strategies, which are shown below.

(i) We use a binary search to try the δ when l is small. For small l, the first considered $\delta = m/2$. If such (l, δ)-MBC exists, we set δ to be $\frac{m/2+m}{2}$, otherwise we set δ to be $\frac{m}{4}$.

(ii) We use an early termination to try δ when l is large. For large l, we set $\delta' = \frac{\delta_{l-1} \times (l-1)+1}{l}$ in which δ_{l-1} is the optimal δ of $l-1$. If the (l, δ')-MBC does not exist, then the algorithm performs an early termination and the optimal δ_l is $\frac{\delta_{l-1} \times (l-1)}{l}$.

Table 2: Statistics of datasets

Dataset	$\lvert V \rvert = n$	$\lvert E \rvert$	$\lvert \mathcal{E} \rvert = m$	$d_{\max}$	$\lvert \mathcal{T} \rvert$	Time scale
Chess	7,301	55,899	63,689	233	101	month
Lkml	26,885	159,996	328,092	14,172	96	month
Enron	86,836	296,952	501,510	2,156	87	month
DBLP	1,729,816	8,546,306	12,007,380	5,980	78	year
YTB	3,223,589	9,376,594	12,218,755	129,819	225	day
FLK	2,302,925	22,838,276	24,690,648	28,276	197	day
MO	24,759	187,986	294,293	5,556	2,351	day
AU	157,222	455,691	549,914	7,325	2,614	day
WT	1,094,018	2,787,967	4,010,611	214,518	2,321	day

The above two strategies can handle the situations when l is very small or very large. The experimental results in Section 5 show that the pruning strategies can speed up the computations in practice.

Complexity of Algorithm 4. The worst time and space complexity of Algorithm 4 are $O(m|\mathcal{T}|^2)$ and $O(n|\mathcal{T}| + m)$ respectively. However, the pruning rule based on Corollary 4 can reduces the computation time greatly. We will show the running time in practice at Section 5.

5 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the proposed algorithms. We implement eight different algorithms for comparison: *(i)* KC [38] is a baseline that computes the k-core ($k = \delta$) in the temporal graph. *(ii)* DS [33] is a baseline algorithm that searches the Densest Subgraph in the temporal graph. *(iii)* DBS [11] is a related algorithm that can find the Densest and Bursting Subgraph in the temporal graph. *(iv)* MBC-B is a baseline that computes (l, δ)-MBC using the framework shown in Algorithm 1, but it enumerates all subsequences to compute maximum l-segment density. *(v)* MBC is the implementation of Algorithm 1, which uses Algorithm 2 to compute MSD. *(vi)* MBC+ is the implementation of Algorithm 3 to compute (l, δ)-MBC. *(vii)* POMBC can output all the POMBCs by Definition 6, and it is an implementation of Algorithm 4. *(viii)* POMBC-B is a basic implementation of POMBC without integrating the pruning rules developed in Corollary 4.

All algorithms are implemented in Python (available at https://github.com/VeryLargeGraph/MBC), and conducted on a Linux kernel 4.4 server with an Intel Core(TM) i5-8400@3.80GHz processor and 32 GB memory. When quantity measures are evaluated, the test was repeated over 5 times and the average is reported here.

Datasets. We use nine different real-world temporal networks in the experiments. The detailed statistics of our datasets are summarized in Table 2, where $d_{\max}$ denotes the maximum number of temporal edges associated with a node, and $|\mathcal{T}|$ denotes the number of snapshots. All snapshots are simple, undirected and unweighted graphs. Chess[1] is a network that represents two chess players playing games together from 1998 to 2006. Lkml[1] is a communication network of the Linux kernel mailing list from 2001 to 2011. Enron[1] is an email communication network between employees of Enron from 1999 to 2003. DBLP[2] is a collaboration network of authors in DBLP from 1940 to Feb. 2018. Youtube[3] (YTB for short) and Flickr[1] (FLK) are friendship networks of users in Youtube and Flickr, respectively. In addition, MathOverflow[3] (MO), AskUbuntu[3] (AU) are temporal networks of interactions on

[1]http://www.konect.cc/
[2]https://dblp.uni-trier.de/xml/
[3]http://snap.stanford.edu/data/index.html

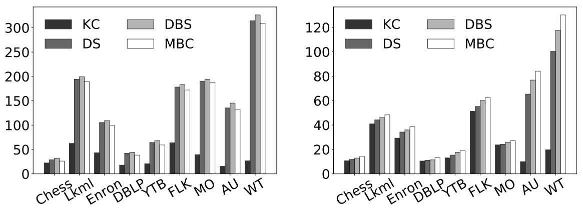

(a) AD (b) AS

Figure 4: Effectiveness results of KC, DS and MBC

the stack exchange web site mathoverflow.net and askubuntu.com, respectively. WikiTalk[3] (WT) is a temporal network representing the interactions among Wikipedia users.

Parameter settings. There are two parameters l and δ in the (l,δ)-MBC model. We vary l from 3 to 11 with a default value of 3 in the testing, and vary δ from 3.0 to 11.0 with a default value of 3.0. Unless otherwise specified, the values of other parameters are set to their default values when varying a parameter.

Goodness metrics. Since most existing metrics (e.g., modularity) for measuring community quality are tailored for traditional graphs, we introduce two goodness metrics evaluating communities for temporal graphs, which are motivated by *density* and *separability* [39]. Let C be a community computed by different algorithms.

Average Density (AD) builds on the idea that good communities are well connected. It measures the fraction of the temporal edges that appear between the nodes in C: AD $= [\frac{\sum_{v_i \in C} deg_{\mathcal{G}_C}(v_i)}{|C|}]$, where $deg_{\mathcal{G}_C}(v_i)$ denotes the number of temporal edges that are associated with v_i in the community C.

Average Separability (AS) captures the intuition that good communities are well-separated from the rest of the network, meaning that they have relatively few across edges between C and the rest of the network: AS $= [\frac{|\{(u,v,t)\in\mathcal{E}:u\in C,v\in C\}|/|C|}{|S=\{(u,v,t)\in\mathcal{E}:u\in C,v\notin C\}|/|S|}]$, which measures the ratio between the internal average density and external average density.

However, in traditional graphs, the density of a given sub-graph S is the ratio between the number of static edges and the nodes inside S, and the separability of S is the ratio between the internal density and external density of S. Obviously, our proposed AD and AS can capture the notion of cohesion within a time frame, since we consider the temporal edges. To the best of our knowledge, we are the first to extend the definition of density and separability from static graphs into temporal graphs.

5.1 Effectiveness Evaluation

Exp-1. Effectiveness of KC, DS, DBS and MBC. Fig. 4 shows the qualities of the communities computed by different algorithms under the default parameter setting. Similar results can also be observed using the other parameter settings. As seen in Fig. 4(a), DBS and DS outperform the others in terms of the AD metric (but DBS and DS are very time consuming). Considering AD, we can also observe that DS is slightly better than DBS. This is because that DS obtains the subgraph with the largest density and DBS obtains the densest subgraph considering the bursting property. MBC is slightly weaker than DBS, but much better than KC in terms of AD. Furthermore, the AD values of both DS, DBS and MBC in WT are much higher than those in the other datasets. The reason is that the maximum degree in WT is largest among all datasets; thus, there must exist a community with higher density. In Fig. 4(b), the MBC community we proposed have a higher AS than DBS and DS on all datasets. Compared to the other datasets, the AD on MO is high , but the AS is low. The reason is that AS captures the ratio between the internal average density and external average density. Clearly, each node in MBC has a high internal average density. In conclusion, DBS searches the densest bursting subgraph in the temporal graph so it has the best AD; MBC has the best AS and slightly lower AD; KC performs poorly in all the effectiveness tests.

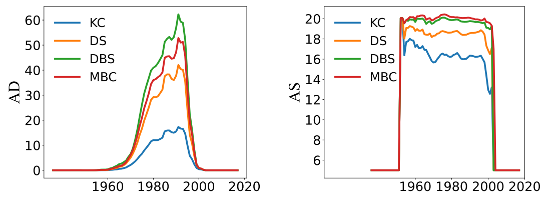

(a) AD (b) AS

Figure 5: Results of burstiness testings on DBLP

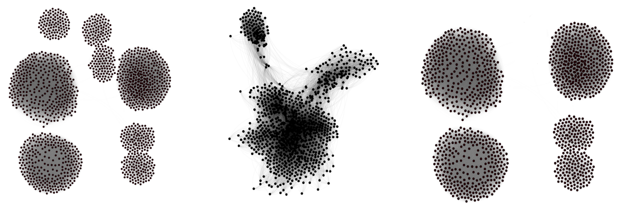
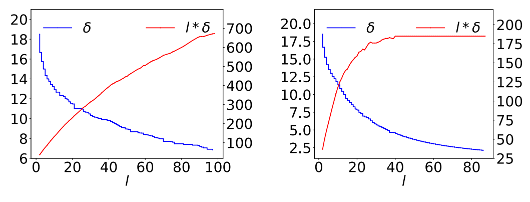

(a) MBC on DBLP (b) MBC on Enron (c) Bursty truss on DBLP

Figure 6: Results of the bursty subgraphs

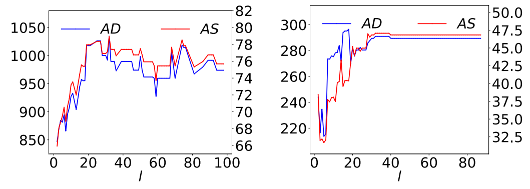
Fig. 5 shows the distributions of average density (AD) and average separability (AS) for the algorithms while varying the ends of the time intervals. More specifically, we show the AD and AS of the subgraph KC, DS, DBS, MBC in (1945,1950], (1946,1951] ... (2015,2020] on DBLP by observing segments of each 5 years. In Fig. 5(a), we can see that the curves of the AD for all algorithms increase from 1960 to 2000, and then decrease from 2000 to 2020. These results indicate that there indeed exist bursty subgraphs which are densely-connected rapidly in the real datasets. Note that, MBC is slightly weaker than DBS in terms of AD, and better than KC and DS, which is consistent with the results in Fig. 4(a). In Fig. 5(b), we can observe that MBC is better than the other algorithms in terms of AS, because MBC can seek more accurate communities which has low external average density. Interestingly, the AD of all the algorithms decrease from 1960 to 2020. This is because in early years the publications and the collaborations are limited but in recent years the collaborations of the authors are active, resulting in that there are more external collaborations in recent years.

Exp-2. Results of the bursty subgraph. Fig. 6 shows the results of MBC and the bursty k-truss model on DBLP and Enron. Similar results can also be observed on the other datasets. In Fig. 6(a), we can observe that MBC ($l = 10, \delta = 10$) on DBLP contains many connected components, since the subgraphs may be split into several parts by the definition of the k-core model. In Fig. 6(b), we can see MBC ($l = 5, \delta = 10$) on Enron are connected, but it can also be grouped by three parts. Fig. 6(c) shows the bursty k-truss which is obtained by a simple modification of Definition 5. The bursty k-truss has almost the same structure as the bursty k-core, except that the bursty k-truss removes some small connected components



(a) Lkml (b) Enron

Figure 7: l, δ values of POMBCs **on different datasets**



(a) Lkml (b) Enron

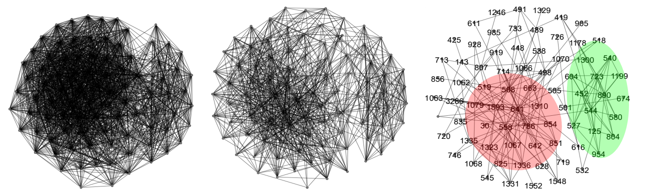
Figure 8: Effectiveness results of POMBCs

in k-core. This is because in k-truss, the cross-domain edges will be removed and then the small components in the subgraph will also be deleted.

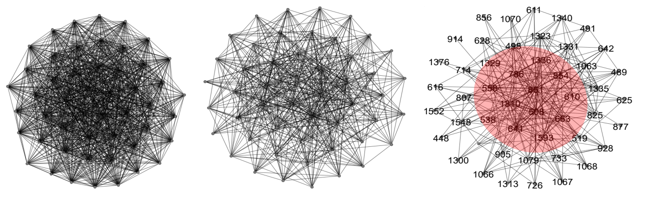
Exp-3. Results of POMBC**.** Fig. 7 shows the l, δ values for each POMBC on Lkml and Enron. Again, similar results can also be observed on the other datasets. From Fig. 7(a), we observe that when $l = 2$, an (l, δ)-MBC in Lkml achieves the maximum l-segment density which is equal to 18.5. The δ values drop dramatically when $l = 20$. As desired, the δ values in both Fig. 7(a) and Fig. 7(b) exhibit a staircase shape because of the parato-optimal property. In Fig. 7(a), we can see that l increases from $l = 21$ to $l = 25$ when δ is unchanged, which shows that there is a POMBC which can dominate others in terms of parameters l, δ. In Fig. 7(b), we can also observe that the values of $l * \delta$ on Enron grow rapidly from $l = 0$ to $l = 40$, and then the values tend to be balanced. This is because when $l > 40$, the POMBC in Enron is the same subgraph with changing the lowest l constraint. However, the values of $l * \delta$ on Lkml continue increasing from $l = 0$ to $l = 96$. This is the reason that the average degree inside Lkml is much larger than that inside Enron, so it has different communities with high degrees.

Fig. 8 shows the AS and AD values of POMBCs on Lkml and Enron. The results on the other datasets are consistent. The AS and AD values increase as l increases from 0 to 30. We can also see that the curves of the AS and AD values match the curves of $l * \delta$ values in Fig. 7. This is because while $l * \delta$ values increase, the inside degree of the subgraphs increase, which results in the growth of AS and AD. We can observe that in Fig. 8(b), when $l = 15$ to $l = 20$, the AD values increase but the AS values decline. It shows that AS and AD are two metrics that are not all increasing at the same time. When $l > 30$, AS and AD values change slightly. This is the reason that when $l > 30$ (i) the POMBC in Enron is the same subgraph; (ii) $l * \delta$ values of the POMBC on Lkml increases, but the quantity of communities may not enlarge.

Exp-4. Communities results on Enron**.** We present the communities results on Enron. The Enron dataset consists of emails sent between employees of Enron from 1999 to 2003. Nodes in the network are individual employees, and edges are individual emails. As the results of KC and DS contain too many edges and cannot be applied to finding the bursting patterns, we compare the results of DBS and MBC in this section. Fig. 9(a) shows one DBS on Enron



(a) DBS (b) DBS (each edge $\#t \geq 5$) (c) DBS (each edge $\#t \geq 10$)



(d) MBC (e) MBC (each edge $\#t \geq 5$) (f) MBC (each edge $\#t \geq 10$)

Figure 9: Communities results on Enron

with all the temporal edges (u, v, t) inside it. Figs. 9(b-c) show the DBS on Enron keeping only the temporal edges that exist at no fewer than 5 and 10 timestamps, respectively. Similarly, Figs. 9(d-f) show the MBC and the MBC keeping only the temporal edges which exist at no fewer than 5 and 10 timestamps, respectively. We can obtain the following observations: ***(i)*** **The subgraph obtained by** DBS **is slightly denser than** MBC**.** Specifically, the AD values in DBS and MBC are 296.51 and 289.48, respectively. However, the MBC has higher AS than DBS, as the AS values in 9(a) and 9(d) are 42.95 and 46.83, respectively. The results indicate the fact that DBS has higher AD and lower AS than MBC, which is consistent with the result of Exp-1. ***(ii)*** **The** MBC **contains more temporal edges that are frequently connected with each other than** DBS**.** In DBS, there exist 90 nodes, 2151 edges in Fig. 9(a), 90 nodes, 1027 edges in Fig. 9(b) and 75 nodes, 195 edges in Fig. 9(c) (isolated nodes are removed). Furthermore, considering the MBC, there are 50 nodes, 1040 edges in Fig. 9(d), 50 nodes, 569 edges in Fig. 9(e) and 49 nodes, 272 edges in Fig. 9(f). We can see that although DBS has more edges than MBC $(2151 > 1040)$, after cutting some infrequent temporal edges whose $\#t < 10$, DBS has fewer edges than MBC $(195 < 272)$. ***(iii)*** **In Figs. 9(a-c), we observe a similar result as ref [30] that** DBS **obtains outliers which belong to the other dense regions of the graph.** However, our models in Figs. 9(d-f) show clearer communities than DBS. Moreover, we can separate the two dense regions of Fig. 9(c) into a red region and a green region. We also observe that the red region in Fig. 9(c) is similar to the red region in Fig. 9(f), which consists of the same kernel nodes in the community (such as: 851, 1310, 508, 1336, 786, 854...). These results indicate that our proposed MBC model can identify higher edge-connected and more accurate bursting communities than DBS.

5.2 Efficiency Evaluation

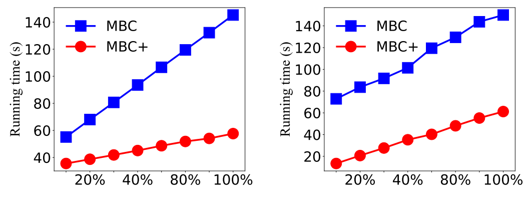
Exp-5. Running time of the algorithms. Table. 3 evaluates the running time of KC, DS, DBS, MBC-B, MBC and MBC+ with parameters $l = 3$ and $\delta = 3$. Similar results can also be observed with the other parameter settings. From Table. 3, we can see that MBC+ is much faster than DS, DBS, MBC-B and MBC on all datasets. Note that KC is the fastest algorithm, as it has a linear time complexity of [38]. However, KC is ineffective in finding bursting communities,

Table 3: Running time (s) of different algorithms

Dataset	KC	DS	DBS	MBC-B	MBC	MBC+
Chess	0.05	13.45	8.32	1.32	0.78	**0.50**
Lkml	0.06	35.23	20.32	2.4	1.02	**0.36**
Enron	0.19	134.2	82.32	13.41	3.54	**1.25**
DBLP	6.84	1602.32	542.54	187.32	53.90	**26.95**
YTB	30.53	6653.23	3123.13	759.52	126.92	**68.23**
FLK	17.53	5234.23	3123.32	876.4	122.87	**34.52**
MO	0.11	5602.21	2213.21	1200.23	30.15	**3.71**
AU	0.52	10232.23	3121.31	2599.78	66.89	**13.36**
WT	2.15	23123.23	8021.31	11865.87	145.23	**57.65**

Table 4: Running time (s) of POMBC-B V.S. POMBC

	POMBC-B (t_1)	POMBC (t_2)	t_2/t_1
Chess	245.23	53.24	21.7%
Lkml	682.32	175.32	25.6%
Enron	953.42	280.43	29.4%
DBLP	10232.32	2407.13	23.5%
YTB	24563.23	6153.52	25.1%
FLK	14245.23	3698.13	26.1%
MO	17232.42	3424.12	19.9%
AU	43231.45	11678.23	27.0%
WT	>1 day	>1 day	N/A



(a) percents of $|\mathcal{T}|$ (b) percents of edges

Figure 10: Scalability testings on WT

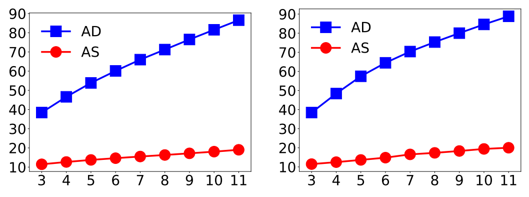
as shown in Exp-1. Recall that the theoretical running time of MBC+ is $O(\alpha|\mathcal{T}| + \beta l)$ where $\alpha = |V_c|, \beta = |E_c|$ denotes the number of nodes and edges in k-core ($k = \delta$) of G. Thus, on DBLP, KC takes 6.8 seconds and our proposed MBC+ only consumes 26.9 seconds. On WT, we can see that DS takes 23123 seconds, DBS takes 8021 seconds, MBC-B takes 11865 seconds and MBC+ only takes 57 seconds. These results confirm that our proposed algorithms are indeed very efficient on large real-life temporal networks.

Exp-6. Running time of computing all POMBCs. Table 4 shows the running time of POMBC-B and POMBC with the default parameter setting. We can see that POMBC requires approximately 20%-30% of the time of POMBC-B on all the datasets. For example, POMBC-B needs approximately 17,232 seconds and 43,231 seconds to compute all the POMBCs in MO and AU datasets but POMBC only needs 19.9% and 27.0% times, respectively. This is because the core reduction pruning in line 11 of Algorithm 4 reduces the temporal graph into a much smaller size, and the speed up strategies can avoid computing (l, δ)-MBC with small l and δ. Note that both POMBC-B and POMBC cannot obtain results on WT in 1 day. The results above indicate that the pruning rule in Section 4 is indeed very powerful in practice.

Exp-7. Scalability. Fig. 10 shows the scalability of MBC and MBC+ on the WT dataset. Similar results can also be observed on the other datasets. We generate ten temporal subgraphs of the temporal edges by 10%-100% of the timestamps, sample the temporal edges forward in time by 10%-100%, and then evaluate the running times of MBC and MBC+ on those subgraphs. As shown in Fig. 10, the running time increases smoothly with increasing numbers of edges or increasing sizes of $|\mathcal{T}|$. These results suggest that our proposed algorithms are scalable when handling large temporal networks.

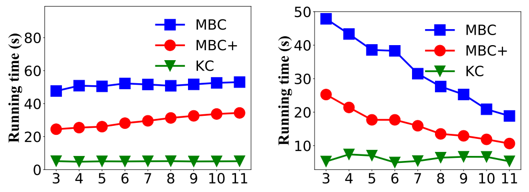
Table 5: Memory overhead of MBC and MBC+

	Graph in Memory	Memory of MBC	Memory of MBC+
Chess	3.5MB	9.2MB	44.2MB
Lkml	20.1MB	44.4MB	121.2MB
Enron	53.3MB	107.6Mb	303.2MB
DBLP	1,089.5MB	2,328.2MB	3,934.3MB
YTB	698.5MB	1,452.8MB	3,318.1MB
FLK	1,375.5MB	3,198.2MB	5,647.2MB
MO	13.23MB	45.23MB	92.75MB
AU	50.23MB	140.32MB	459.2MB
WT	324.5MB	1023.23MB	3,163.2MB



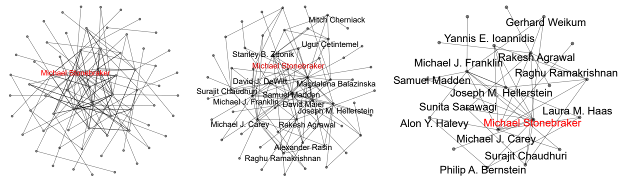
(a) vary l (DBLP) (b) vary δ (DBLP)

Figure 11: Effectiveness of MBC with varying l, δ on DBLP



(a) vary l (DBLP) (b) vary δ (DBLP)

Figure 12: Running time (s) with varying l, δ on DBLP



(a) $l = 3, \delta = 3$ (b) $l = 5, \delta = 5$ (c) $l = 7, \delta = 7$

Figure 13: Case study of (l, δ)-MBCs on DBLP

Exp-8. Memory overhead. Table 5 shows the memory usage of MBC and MBC+ on different datasets. We can see that the memory usage of MBC and MBC+ is higher than the size of the temporal graph, because MBC only needs to store $deg[u]$ (for each node u) but MBC+ needs to store $\mathcal{MSD}[u]$, $\mathcal{MTS}[u]$, $\mathcal{DS}[u]$ (for each node u). In practice, we can free the memory of $\mathcal{MSD}[u]$, $\mathcal{MTS}[u]$ and $\mathcal{DS}[u]$ once u has been added into the deleting queue Q. Therefore, on large datasets, the memory usage of MBC+ is typically lower than ten times of the size of the temporal graph. For instance, on WT, MBC+ consumes 3,163.2MB memory while the graph needs 324.5MB. These results indicate that MBC and MBC+ achieve near linear space complexity, which confirms our theoretical analysis in Section 3.

Exp-9. Effectiveness results with varying parameters. Here we study how the parameters affect the effective performance of our algorithm. Fig. 11 shows the results of MBC with varying parameters on DBLP. Similar results can also be observed on the other datasets. As seen, AD increases with growing l and δ. This is because AD measures the inside degrees of MBC, and when l or δ values increase, $l * \delta$ will be larger and AD will definitely increase. We also observe that the AS changes slightly in Fig. 11. The reason may be that when l or δ values increase, both the numerator and denominator of AS will increase. Furthermore, as l and δ control the lower bound of the segment density and directly affect the internal average density, the AS slightly increases.

Exp-10. Running time with varying parameters. Fig. 12 shows the running time of KC, MBC and MBC+ with varying parameters on DBLP. Similar results can also be observed on the other datasets. As seen, MBC+ is faster than MBC under all parameter settings. In Fig. 12(a), the running times of KC and MBC remain unchanged, but the running time of MBC+ increases slowly with increasing l. These results confirm that the time complexity of KC and MBC is independent of l, and the time complexity of MBC+ is linear w.r.t. l. We also see that the running time of MBC+ and MBC decrease with increasing δ, because all of them need to reduce the graph by the k-core based on Property 3, and the size of the k-core decreases as δ increases.

Exp-11. Case study on DBLP with varying parameters. We conduct a case study using DBLP to show the (l, δ)-MBCs with varying parameters l and δ. Figs. 13(a-c) show the three communities of Prof. Michael Stonebraker obtained by MBC with $l = \delta = 3$, 5 and 7. Note that, to be more visualized, we only keep the temporal edges that exist at more than 5 timestamps, which means that the researchers at two ends of each edge in Fig. 13 have cooperated for no fewer than 5 years. In Fig. 13(c), we can see that the $(7, 7)$-MBC comprises close collaborators of Prof. Stonebraker. Interestingly, some researchers are not cooperated with Prof. Stonebraker most frequently, such as Sunita Sarawagi, Surajit Chaudhuri and Michael J. Franklin, but all of them are top researchers and have published more than 100 papers in their research areas. This is the reason that MBC can find the bursting cores which are actually kernels in the communities. From Figs. 13(a-b), we can see that the $(3, 3)$-MBC and the $(5, 5)$-MBC not only contain $(7, 7)$-MBC in Fig. 13(c), but also include some other close collaborators of Prof. Stonebraker, such as Stanley B. Zdonik, Ugur Çetintemel and so on. In conclusion, we can use l, δ to control the MBC depending on the application requirements, and the larger l, δ is, the more likely MBC is the core of the temporal graph.

6 RELATED WORK

K-core [7, 9, 15, 16, 37] is an important cohesive subgraph model that can represent communities in a graph. Such a concept of k-core was first proposed by Seidman [35]. Recently, Malliaros et al. perform an in-depth discussion of core decomposition [28]. Other notable cohesive subgraph models include k-truss [21], k-plex [4, 12], maximal clique [10] and quasi-clique [36]. However, bursting core mining in temporal graph is a novel task that has not been well studied before. Below, we review the recent studies on temporal subgraph analysis and dynamic community mining.

Temporal Subgraph Analysis. Temporal subgraph analysis has attracted much attention in recent year, including *(i) Temporal Core Model:* Galimberti et al. [14, 17] proposed temporal span-cores, in which each node has minimum degree in a specific time interval; Wu et al. [38] studied the core decomposition problem in temporal networks; Yu et al. [42] computed the historical k-cores in the graph snapshots over the time window; Li et al. [23] developed an algorithm to detect persistent cores in a temporal graph. *(ii) Temporal Clique Model:* Qin et al. [29] proposed a model for seeking periodic cliques in a temporal graph. Yang et al. [40] studied a problem of finding a set of diversified quasi-cliques from a temporal graph. *(iii) Temporal Subgraph Model:* Yang et al. [41] proposed an algorithm to detect frequent changing components in temporal graph; Huang et al. [20] investigated the minimum spanning tree problem in temporal graphs; Gurukar et al. [18] presented a model to identify the recurring subgraphs that have similar sequence of information flow. *(iv) Temporal Densest Subgraph Model:* Ma et al. [27] and Bogdanov et al. [6] investigated the densest subgraph problem in weighted temporal graphs. Rozenshtein et al. [32] studied the problem of mining densest subgraphs at different time intervals, and a problem of finding the densest subgraph in a temporal network [33]. Liu et al. [26] proposed a novel stochastic approach to find the densest lasting subgraph. Some other works [5, 13] maintained the average-degree densest-subgraph in a graph streaming scenario. However, the above works do not study the problem of mining bursting communities in temporal graphs.

Recently, Chu et al. [11] studied the problem of mining the densest and bursting subgraphs in temporal graphs. However, to search the bursting communities, the model of the densest and bursting subgraph has three limitations compared to MBC we proposed, as described in Section 1. Furthermore, in Section 5, the experiments show our algorithm MBC+ performs better than DBS in the testing of efficiency and effectiveness.

Dynamic Community Mining. In dynamic networks, each edge is associated with a created timestamp [31]. Different from the temporal subgraph analysis, the studies on dynamic community mining aim to maintain communities that evolve over time. For example, Lin et al. [25] proposed a probabilistic generative model for analyzing communities and their evolutions; Chen et al. [8] tracked community dynamics by introducing graph representatives; Agarwal et al. [1] studied how to find dense clusters efficiently for dynamic graphs despite rapid changes to microblog streams. Li et al. [24] devised an algorithm that can maintain the k-core in large dynamic graphs. Unlike these studies, our work mainly aims to detect bursting communities in evolving graphs.

7 CONCLUSION

In this work, we study a problem of mining bursting cores in a temporal graph. We propose a novel model, called (l, δ)-MBC, to characterize the bursting core in the temporal graph. To find all (l, δ)-MBCs, we first develop an dynamic programming algorithm which can compute the segment density efficiently. Then, we propose an improved algorithm with several novel pruning techniques to improve the efficiency. Subsequently, we develop an algorithm which can compute the pareto-optimal bursting communities w.r.t. the parameters l and δ. Finally, we conduct comprehensive experiments using 9 real-life temporal networks, and the results demonstrate the efficiency, scalability and effectiveness of our algorithms. In the future, we plan to extend our model to capture the evolution of the community by considering the joins and exits of vertices. Furthermore, we can also consider the graph with bursting labels by first embedding the topic labels of each timestamp into a list of popular index, and then modifying the proposed algorithm to calculate the most bursty parts.

REFERENCES

[1] Manoj K. Agarwal, Krithi Ramamritham, and Manish Bhide. 2012. Real Time Discovery of Dense Clusters in Highly Dynamic Graphs: Identifying Real World Events in Highly Dynamic Environments. *Proc. VLDB Endow.* 5, 10 (2012), 980–991.

[2] Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. 2007. An event-based framework for characterizing the evolutionary behavior of interaction graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007.* 913–921.

[3] Albert-Lászlo Barabási. 2005. The origin of bursts and heavy tails in human dynamics. *Nature* 435, 7039 (2005), 207–211.

[4] Devora Berlowitz, Sara Cohen, and Benny Kimelfeld. 2015. Efficient Enumeration of Maximal k-Plexes. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015.* 431–444.

[5] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. 2015. Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015.* 173–182.

[6] Petko Bogdanov, Misael Mongiovì, and Ambuj K. Singh. 2011. Mining Heavy Subgraphs in Time-Evolving Networks. In *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011.* 81–90.

[7] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. 2019. Distance-generalized Core Decomposition. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019.* 1006–1023.

[8] Zhengzhang Chen, Kevin A. Wilson, Ye Jin, William Hendrix, and Nagiza F. Samatova. 2010. Detecting and Tracking Community Dynamics in Evolutionary Networks. In *ICDMW 2010, The 10th IEEE International Conference on Data Mining Workshops, Sydney, Australia, 13 December 2010.* 318–327.

[9] James Cheng, Yiping Ke, Shumo Chu, and M. Tamer Özsu. 2011. Efficient core decomposition in massive networks. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany.* 51–62.

[10] James Cheng, Yiping Ke, Ada Wai-Chee Fu, Jeffrey Xu Yu, and Linhong Zhu. 2011. Finding Maximal Cliques in Massive Networks. *ACM Trans. Database Syst.* 36, 4 (2011), 21:1–21:34.

[11] Lingyang Chu, Yanyan Zhang, Yu Yang, Lanjun Wang, and Jian Pei. 2019. Online Density Bursting Subgraph Detection from Temporal Graphs. *Proc. VLDB Endow.* 12, 13 (2019), 2353–2365.

[12] Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. 2017. Fast Enumeration of Large k-Plexes. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017.* 115–124.

[13] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. 2015. Efficient Densest Subgraph Computation in Evolving Graphs. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015.* 300–310.

[14] Edoardo Galimberti, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2018. Mining (maximal) Span-cores from Temporal Networks. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018.* 107–116.

[15] Edoardo Galimberti, Francesco Bonchi, and Francesco Gullo. 2017. Core Decomposition and Densest Subgraph in Multilayer Networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017.* 1807–1816.

[16] Edoardo Galimberti, Francesco Bonchi, Francesco Gullo, and Tommaso Lanciano. 2020. Core Decomposition in Multilayer Networks: Theory, Algorithms, and Applications. *ACM Trans. Knowl. Discov. Data* 14, 1 (2020), 11:1–11:40.

[17] Edoardo Galimberti, Martino Ciaperoni, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2020. Span-Core Decomposition for Temporal Networks: Algorithms and Applications. *ACM Trans. Knowl. Discov. Data* 15, 1, Article 2 (dec 2020), 44 pages.

[18] Saket Gurukar, Sayan Ranu, and Balaraman Ravindran. 2015. COMMIT: A Scalable Approach to Mining Communication Motifs from Dynamic Networks. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015.* 475–489.

[19] Petter Holme and Jari Saramaki. 2012. Temporal networks. *Physics Reports* 519 (2012), 97–125.

[20] Silu Huang, Ada Wai-Chee Fu, and Ruifeng Liu. 2015. Minimum Spanning Trees in Temporal Graphs. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015.* 419–430.

[21] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014.* 1311–1322.

[22] Rohit Kumar, Toon Calders, Aristides Gionis, and Nikolaj Tatti. 2015. Maintaining Sliding-Window Neighborhood Profiles in Interaction Networks. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II.* 719–735.

[23] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent Community Search in Temporal Networks. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018.* 797–808.

[24] R. H. Li, J. X. Yu, and R. Mao. 2014. Efficient Core Maintenance in Large Dynamic Graphs. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2014), 2453–2465.

[25] Yu-Ru Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle L. Tseng. 2008. Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008.* 685–694.

[26] Xuanming Liu, Tingjian Ge, and Yinghui Wu. 2019. Finding Densest Lasting Subgraphs in Dynamic Graphs: A Stochastic Approach. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019.* 782–793.

[27] Shuai Ma, Renjun Hu, Luoshu Wang, Xuelian Lin, and Jinpeng Huai. 2017. Fast Computation of Dense Temporal Subgraphs. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017.* 361–372.

[28] Fragkiskos D. Malliaros, Christos Giatsidis, Apostolos N. Papadopoulos, and Michalis Vazirgiannis. 2020. The core decomposition of networks: theory, algorithms and applications. *VLDB J.* 29, 1 (2020), 61–92.

[29] Hongchao Qin, Rong-Hua Li, Guoren Wang, Lu Qin, Yurong Cheng, and Ye Yuan. 2019. Mining Periodic Cliques in Temporal Networks. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019.* 1130–1141.

[30] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. 2015. Locally Densest Subgraph Discovery. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015.* 965–974.

[31] Giulio Rossetti and Rémy Cazabet. 2018. Community Discovery in Dynamic Networks: A Survey. *ACM Comput. Surv.* 51, 2 (2018), 35:1–35:37.

[32] Polina Rozenshtein, Francesco Bonchi, Aristides Gionis, Mauro Sozio, and Nikolaj Tatti. 2018. Finding Events in Temporal Networks: Segmentation Meets Densest-Subgraph Discovery. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018.* 397–406.

[33] Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. 2017. Finding Dynamic Dense Subgraphs. *ACM Transactions on Knowledge Discovery from Data* 11, 3 (2017), 27:1–27:30.

[34] Ahmet Erdem Sariyüce, C. Seshadhri, and Ali Pinar. 2018. Local Algorithms for Hierarchical Dense Subgraph Discovery. *Proc. VLDB Endow.* 12, 1 (2018), 43–56.

[35] Stephen B. Seidman. 1983. Network structure and minimum degree. *Social Networks* 5, 3 (1983), 269–287.

[36] Charalampos E. Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria A. Tsiarli. 2013. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013.* 104–112.

[37] Dong Wen, Lu Qin, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. 2016. I/O efficient Core Graph Decomposition at web scale. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016.* 133–144.

[38] Huanhuan Wu, James Cheng, Yi Lu, Yiping Ke, Yuzhen Huang, Da Yan, and Hejun Wu. 2015. Core decomposition in large temporal graphs. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015.* 649–658.

[39] Jaewon Yang and Jure Leskovec. 2012. Defining and Evaluating Network Communities Based on Ground-Truth. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012.* 745–754.

[40] Yi Yang, Da Yan, Huanhuan Wu, James Cheng, Shuigeng Zhou, and John C. S. Lui. 2016. Diversified Temporal Subgraph Pattern Mining. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016.* 1965–1974.

[41] Yajun Yang, Jeffrey Xu Yu, Hong Gao, Jian Pei, and Jianzhong Li. 2014. Mining most frequently changing component in evolving graphs. *World Wide Web* 17, 3 (2014), 351–376.

[42] Michael Yu, Dong Wen, Lu Qin, Ying Zhang, Wenjie Zhang, and Xuemin Lin. 2021. On Querying Historical K-Cores. *Proc. VLDB Endow.* 14, 11 (2021), 2033–2045.