Online Density Bursting Subgraph Detection from Temporal Graphs *

Lingyang Chu
Huawei Technologies Canada
Burnaby, Canada
lingyang.chu1@huawei.com

Yanyan Zhang
Fortinet Technology (Canada)
Burnaby, Canada
yzhang01@fortinet.com

Yu Yang
City University of Hong Kong
Hong Kong, China
yuyang@cityu.edu.hk

Lanjun Wang
Huawei Technologies Canada
Burnaby, Canada
lanjun.wang@huawei.com

Jian Pei
Simon Fraser University
Burnaby, Canada
jpei@cs.sfu.ca

ABSTRACT

Given a temporal weighted graph that consists of a potentially endless stream of updates, we are interested in finding density bursting subgraphs (DBS for short), where a DBS is a subgraph that accumulates its density at the fastest speed. Online DBS detection enjoys many novel applications. At the same time, it is challenging since the time duration of a DBS can be arbitrarily long but a limited size storage can buffer only up to a certain number of updates. To tackle this problem, we observe the critical decomposability of DBSs and show that a DBS with a long time duration can be decomposed into a set of indecomposable DBSs with equal or larger burstiness. We further prove that the time duration of an indecomposable DBS is upper bounded and propose an efficient method $TopkDBSOL$ to detect indecomposable DBSs in an online manner. Extensive experiments demonstrate the effectiveness, efficiency and scalability of $TopkDBSOL$ in detecting significant DBSs from temporal graphs in real applications.

PVLDB Reference Format:
Lingyang Chu, Yanyan Zhang, Yu Yang, Lanjun Wang, Jian Pei. Online Density Bursting Subgraph Detection from Temporal Graphs. *PVLDB*, 12(13): 2353-2365, 2019.
DOI: https://doi.org/10.14778/3358701.3358704

1. INTRODUCTION

General Eric Shinseki said, "If you don't like change, you're going to like irrelevance even less". Finding the most and fastest changing parts is a central task in analyzing temporal data. For example, in a stream of snapshots of a business collaboration network, where each vertex is a person or a company and the weight of an edge represents the collaboration strength between two parties in the time duration of a snapshot, a density bursting subgraph is a group of parties whose collaboration strengths in between increase dramatically fast. Such a density bursting subgraph may indicate that a new business consortium is forming, for example, due to new business opportunities.

As another concrete example, taxi trips in a city naturally form a temporal network, where each vertex is a location in the city and the weight of an edge is the number of taxi trips between two locations during a specific time period. In such a network, a density bursting subgraph indeed reveals a burst of taxi trips among a group of locations. Our case study in Figure 2 and Section 8.1 gives two examples of such bursts of taxi trips, which reveal interesting travel patterns of people on weekdays and weekends, respectively.

As long as one wishes, the list of possible applications of finding density bursting subgraphs can keep growing easily. Surprisingly, although finding density bursting subgraphs is interesting and has many applications, this problem has not been touched systematically in literature. As reviewed in Section 2, the existing works on finding dense subgraphs [1, 7, 28, 32, 34] only focus on density but do not consider the speed of density changes. The previous works on dense temporal subgraph detection [2, 3, 6, 24, 29, 30, 35] maintain dense subgraphs against incremental or streaming updates, but again do not account the change speed of density. Since a dense subgraph may slowly accumulate a large density in a long time, it may not necessarily be a density bursting subgraph. Therefore, existing dense (temporal) subgraph detection methods cannot be straightforwardly extended to detect density bursting subgraphs.

As will be investigated in Section 5, a closer look discloses that the problem of finding density bursting subgraphs is far from trivial. Due to the nature of weighted graphs, a burst can last for a long and potentially indefinite time. A static method has to buffer all the snapshots involved in a burst, thus cannot handle a large number of updates. This leaves us no choice but to design an efficient online algorithm for density bursting subgraph detection.

In this paper, we tackle the novel problem of online density bursting subgraph detection. Specifically, given a stream of snapshots of a temporal graph, a *density bursting subgraph* (**DBS** for short) is a subgraph that accumulates its density at the fastest speed during a time interval. Here, the density of a subgraph is measured by *cohesiveness* [20, 27], which is the average connection strength between all vertices in the subgraph; the speed of density accumulation is measured by *burstiness*, that is, the ratio between the density gain of a subgraph and the time to accumulate the gain. We make the following contributions.

*This research is partly based on Yanyan Zhang's Master's thesis done at Simon Fraser University, Burnaby, Canada.

Proceedings of the VLDB Endowment, Vol. 12, No. 13
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3358701.3358704

First, we consider the static version of the top-k DBS finding (**TDF** for short) problem, that is, all snapshots of a temporal graph are available and we find the set of DBSs with the top-k largest burstiness. We model the static TDF problem as a mixed integer programming problem and show that it is NP-hard. We also propose a baseline method $SlideWin$ to find a good solution to the TDF problem by iteratively solving a constrained quadratic programming (CQP) problem and a maximum density segment (MDS) problem.

Second, considering the general DBS detection problem, we show that the time duration of a DBS can be arbitrarily long, and thus a straightforward extension of $SlideWin$ or alike does not work. To tackle the problem systematically, we follow a principled approach – we try to identify the "atomic" components of DBSs. Critically, we observe that a DBS with a large time duration can be decomposed into a set of indecomposable DBSs with equal or larger burstiness. Most importantly, we show that the time duration of an indecomposable DBS has a non-trivial upper bound, which makes it possible to detect indecomposable DBSs in an online manner.

Last, we formulate the online top-k DBS finding (**OTDF** for short) problem, which is to detect top-k indecomposable DBSs online. The OTDF problem is also NP-hard. We develop an efficient algorithm $TopkDBSOL$ to achieve a 2-approximation of the top-1 indecomposable DBS. Our extensive experiments and two interesting case studies clearly show that our method is effective and efficient.

The rest of the paper is organized as follows. We review related works in Section 2, formulate the static TDF problem in Section 3, and investigate how to find a single DBS in Section 4. We present $SlideWin$ and discuss the major challenge of online DBS detection in Section 5. We explore the decomposition properties of DBSs in Section 6. In Section 7, we present the OTDF problem and develop $TopkDBSOL$. We report a systematic empirical study in Section 8 and conclude the paper in Section 9. By default all mathematical proofs are provided in Appendix.

2. RELATED WORKS

Online density bursting subgraph detection is a novel task that has not been touched in literature. Our work is related to the maximum density segment problem, dense subgraph detection and dense temporal subgraph detection.

The maximum density segment (MDS for short) problem [9] is to select a subsequence of at least L numbers from a sequence of numbers $Q = \langle q_1, \ldots, q_n \rangle$, such that the average sum of the selected numbers is maximized.

The MDS problem has many efficient solutions. Huang *et al.* [16] solved it in $O(nL)$ time. Lin *et al.* [19] proposed a right-skew method with time complexity $O(n \log L)$. Kim *et al.* [18] solved it in $O(n)$ time by finding the line segment with maximum slope. Goldwasser *et al.* [15] applied locally optimal segments to solve it in $O(n)$ time. Chung *et al.* [12] extended Goldwasser's method [15] to a linear online method. Curtis *et al.* [13] proposed an online method based on sliding window.

In our work, we employ MDS methods as a building block. However, as proved in Section 3.2, finding density bursting subgraphs in temporal graphs is NP-hard, dramatically different from the polynomial-time solvable MDS problem.

Dense subgraph detection is a well investigated task that aims to detect dense subgraphs from static graphs [1, 7, 28, 32, 34]. Our work is most related to the cohesiveness based methods [8, 10, 20, 21, 27], which measure the density of a subgraph by cohesiveness, that is, a quadratic function $\mathbf{x}^\top A\mathbf{x}$ of a subgraph embedding $\mathbf{x} \in \triangle^n$.

For unweighted graphs, Motzkin *et al.* [26] proved that maximizing the cohesiveness is equivalent to finding the maximum clique in the graph. For weighted graphs, Pavan *et al.* [27] proposed DS to detect clique-like dense subgraphs by finding local maximum points of cohesiveness in $O(n^2)$ time. Bulò *et al.* [8] proposed IID to find local maximum points of cohesiveness in $O(n)$ time. Since most dense subgraphs exist in local regions, Liu *et al.* [20, 21] proposed SEA to efficiently search local maximum points of cohesiveness in small local subgraphs. All these cohesiveness based methods cannot process the temporal information inherent in a potentially endless stream of snapshots.

Dense temporal subgraph detection aims to detect dense temporal subgraphs from temporal graphs [2, 3, 6, 24, 29, 30, 35]. Conventional methods detect subgraphs with large accumulated density using buffered snapshots or streaming updates, but do not consider the change speed of the density of a subgraph.

Bogdanov *et al.* [6] proposed MEDEN to detect dense temporal subgraphs with a large sum of edge weights. Ma *et al.* proposed FIDES [24] and FIDES$^+$ [25], which are three orders of magnitudes faster than MEDEN. Yang *et al.* [35] used γ-quasi-clique to find the set of most diversified γ-dense subgraphs. Boden *et al.* [5] employed γ-quasi-clique to find vertices densely connected by edges with similar labels. These methods buffer temporal graphs in a static manner, and have to compute from scratch when new updates arrive. Thus, they cannot handle a temporal graph with an endless stream of updates.

Several methods [2, 23, 3, 14, 4, 30] are dedicated to handle streaming updates. For example, Liu *et al.* [23] proposed a stochastic framework to find dense temporal subgraphs by maximizing a utility function that favors high accumulated density and large time duration. Shin *et al.* [30] proposed DenseStream to maintain the densest subgraph with the largest accumulated average degree. Most of these methods find subgraphs with large accumulated density instead of considering how fast a temporal subgraph accumulates density. Since a subgraph can slowly accumulate a large density during an arbitrarily long time, a subgraph with large density is not necessarily a density bursting subgraph. Thus, the above methods cannot accurately find DBSs.

The DenseAlert method proposed by Shin *et al.* [30] can be used to maintain the densest subgraph that has the largest average degree within a fixed length time window. However, DenseAlert cannot effectively find top-k DBSs, because a dense subgraph with a large average degree usually contains a large number of vertices that are not strongly connected with each other [32], however, a DBS is a small clique-like subgraph with fast growing edge connection strength between vertices.

3. PROBLEM DEFINITION

In this section, we first introduce several essential notions, then formalize the top-k density bursting subgraph finding problem and investigate its computational complexity.

3.1 Temporal Graph and Temporal Subgraph

In this paper, we consider general weighted graphs where each edge carries a weight. Two vertices are not connected if the weight of the edge between them is zero.

A **temporal graph**, denoted by $\mathcal{G}(t_0, t_c) = \langle G(t_0), G(t_1), \ldots, G(t_c) \rangle$, is a sequence of snapshots that arrive at times $t_0, t_1 \ldots, t_c$, respectively. Each snapshot is a static graph, and we assume that all snapshots share the same set of vertices V. The temporal graph $\mathcal{G}(t_0, t_c)$ is initialized as an empty graph at time t_0, that is, snapshot $G(t_0)$ contains only the set of vertices V, but

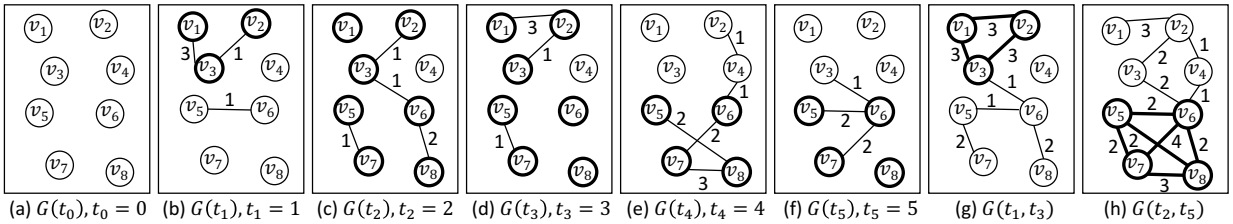


Figure 1: An example of temporal graph, accumulated graph, temporal subgraph and DBS.

no edges. The changes over time happen on the weights of edges. The snapshots $G(t_1), \ldots, G(t_c)$ carry the updates on the weights of edges at time instants $t_1, \ldots, t_c$, respectively.

The **snapshot** that arrives at time $t_h \in \{t_1, \ldots, t_c\}$ is a static graph denoted by $G(t_h) = (V, A(t_h))$, where t_h is the arrival time, and $A(t_h)$ is the affinity matrix that defines the updates on the weights of edges at time t_h.

Denote by $n = |V|$ the number of vertices in V. For each snapshot $G(t_h)$, we represent the affinity matrix $A(t_h)$ by an n-by-n non-negative matrix, where the entry $A_{ij}(t_h)$ at the i-th row and the j-th column of $A(t_h)$ is the update on the weight of the edge between the i-th vertex $v_i \in V$ and the j-th vertex $v_j \in V$. We say there is an edge between v_i and v_j in snapshot $G(t_h)$ if and only if $A_{ij}(t_h) > 0$.

EXAMPLE 1. *Figures 1(a)-1(f) show the snapshots of a temporal graph $\mathcal{G}(t_0, t_5) = \langle G(t_0), \ldots, G(t_5)\rangle$. $G(t_0)$ is initialized as an empty graph that only contains the set of vertices $V = \{v_1, v_2, \ldots, v_8\}$. The weights of all edges in $G(t_0)$ are zero. $G(t_1), \ldots, G(t_5)$ are the snapshots that carry the updates on the edge weights at time $t_1, \ldots, t_5$, respectively.*

A **time interval**, denoted by $T = (t_b, t_e] = \{t_{b+1}, t_{b+2}, \ldots, t_e\}$, is a set of time instants between **begin time** t_b and **end time** t_e, excluding t_b. The **duration** of $T = (t_b, t_e]$ is $t_e - t_b$.

An **accumulated graph** during $T = (t_b, t_e]$ is a static graph $G(t_{b+1}, t_e) = (V, A(t_{b+1}, t_e))$, where $A(t_{b+1}, t_e) = \sum_{h=b+1}^{e} A(t_h)$ is the accumulated affinity matrix. Denote by $A_{ij}(t_{b+1}, t_e)$ the element at the i-th row and the j-th column of $A(t_{b+1}, t_e)$, we write the set of vertices that are connected by at least one edge in $G(t_{b+1}, t_e)$ as $V(t_{b+1}, t_e) = \{v_i \in V \mid \exists A_{ij}(t_{b+1}, t_e) > 0\}$.

EXAMPLE 2. *Figures 1(g) and 1(h) show the accumulated graphs $G(t_1, t_3)$ and $G(t_2, t_5)$ during time intervals $(t_0, t_3]$ and $(t_1, t_5]$, respectively. The corresponding affinity matrices are $A(t_1, t_3) = A(t_1) + A(t_2) + A(t_3)$ and $A(t_2, t_5) = A(t_2) + A(t_3) + A(t_4) + A(t_5)$, respectively. Vertex v_4 is not connected by any edge in $G(t_1, t_3)$, thus v_4 is not contained in $V(t_1, t_3)$.*

Denote by $\mathcal{G}(t_{b+1}, t_e) = \langle G(t_{b+1}), \ldots, G(t_e)\rangle$ the sequence of snapshots that arrive during $T = (t_b, t_e]$. A **temporal subgraph** is a sequence of subgraphs that are induced by a set of weighted vertices $S \subseteq V$ on each of the snapshots in $\mathcal{G}(t_{b+1}, t_e)$. Each vertex v_i in S is assigned a positive weight $\mathbf{x}_i$ that indicates the importance of v_i in S. The weights of all vertices in $V \backslash S$ are set to 0's. In this way, we can induce S by an n-dimensional vector $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^\top$, such that $V_{\mathbf{x}} = S = \{v_i \in V \mid \mathbf{x}_i > 0\}$. Following the conventional dense subgraph detection settings [21], we enforce $\mathbf{x}$ to be in the standard simplex, that is $\mathbf{x} \in \triangle^n = \{\mathbf{x} \mid \sum_i \mathbf{x}_i = 1, \mathbf{x}_i \geq 0\}$. For the rest of the paper, we write a temporal subgraph as a tuple $(\mathbf{x}, T)$.

The **duration** of a temporal subgraph $(\mathbf{x}, T)$ is exactly the duration of T. For any vertex $v_i \in V$, if $v_i \in V_{\mathbf{x}}$, we say v_i is **contained** in $(\mathbf{x}, T)$ and write $v_i \in (\mathbf{x}, T)$.

EXAMPLE 3. *In Figure 1, for $T = (t_0, t_3]$ and $\mathbf{x} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0, 0, 0, 0]$, the temporal subgraph $(\mathbf{x}, T)$ is the sequence of subgraphs induced by the set of vertices $S = \{v_1, v_2, v_3\}$ on the snapshots $G(t_1)$, $G(t_2)$ and $G(t_3)$.*

3.2 Density Bursting Subgraph

For a temporal subgraph $(\mathbf{x}, T)$ and time $t_h \in T$, denote by $G_{\mathbf{x}}(t_h)$ the subgraph induced by $V_{\mathbf{x}}$ from the snapshot $G(t_h)$. We measure the density of $G_{\mathbf{x}}(t_h)$ by the **cohesiveness** [27] defined as follows.

$$q_{\mathbf{x}}(t_h) = \mathbf{x}^\top A(t_h)\mathbf{x} \tag{1}$$

The **burstiness** of $(\mathbf{x}, T)$ measures how fast it accumulates cohesiveness during time interval $T = (t_b, t_e]$, that is,

$$g(\mathbf{x}, T) = \frac{\sum_{h=b+1}^{e} q_{\mathbf{x}}(t_h)}{t_e - t_b} = \frac{\mathbf{x}^\top A(t_{b+1}, t_e)\mathbf{x}}{t_e - t_b} \tag{2}$$

Next, we define **density bursting subgraph** (DBS).

DEFINITION 1. *Given a temporal graph $\mathcal{G}(t_0, t_c)$ and a minimum duration threshold θ, a density bursting subgraph, denoted by $(\mathbf{x}^*, T^*)$ where $T^* = (t_{b^*}, t_{e^*}]$, is a temporal subgraph in $\mathcal{G}(t_0, t_c)$, such that (1) $\mathbf{x}^* \in \triangle^n$ is a local maximum point of $g(\mathbf{x}, T^*)$; (2) T^* is a global maximum point of $g(\mathbf{x}^*, T)$; and (3) $t_{e^*} - t_{b^*} \geq \theta$.*

Denote by $G_{\mathbf{x}^*}(t_{b^*+1}, t_{e^*})$ the subgraph induced by $V_{\mathbf{x}^*}$ from the accumulated graph $G(t_{b^*+1}, t_{e^*})$. As illustrated later in Section 4, condition (1) of Definition 1 requires $G_{\mathbf{x}^*}(t_{b^*+1}, t_{e^*})$ to be a small clique-like dense subgraph in $G(t_{b^*+1}, t_{e^*})$ [8, 22, 27].

Given $\mathbf{x}^*$, condition (2) of Definition 1 requires T^* to be the optimal time interval, such that the temporal subgraph $(\mathbf{x}^*, T^*)$ achieves the largest burstiness.

In condition (3) of Definition 1, the minimum duration threshold θ effectively disqualifies trivial temporal subgraphs that have large burstiness but very short durations. According to Equation 2, a trivial temporal subgraph with a very small duration can have a large burstiness even if its cohesiveness is very small. More often than not, such trivial temporal subgraphs consist of a single edge in snapshots with very small durations, which are not of much interest in real world applications.

EXAMPLE 4. *In Figure 1, for $\theta = 2$, $T = (t_0, t_3]$ and $\mathbf{x} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0, 0, 0, 0]$, the temporal subgraph $(\mathbf{x}, T)$ is a DBS satisfying all the conditions in Definition 1. The duration of $(\mathbf{x}, T)$ is $t_3 - t_0 = 3$. The burstiness of $(\mathbf{x}, T)$ is $g(\mathbf{x}, T) = \frac{2}{3}$. For $\theta = 2$, $T' = (t_1, t_5]$ and $\mathbf{x}' = [0, 0, 0, 0, \frac{1}{2}, \frac{1}{3}, \frac{1}{12}, \frac{1}{12}]$, the temporal subgraph $(\mathbf{x}', T')$ is not a DBS, because $\mathbf{x}'$ is not a local maximum point of $g(\mathbf{x}', T')$, which violates condition (1) of Definition 1. If $\theta \leq 1$, then $\mathbf{x}'' = [0, 0, 0, 0, 0, 0, \frac{1}{2}, \frac{1}{2}]$ and $T'' = (t_3, t_4]$ will induce a DBS with a large burstiness $g(\mathbf{x}'', T'') = 1.5$. However, $(\mathbf{x}'', T'')$ trivially consists of a single edge between v_7 and v_8 in $G(t_4)$, which is not of much practical interest.*

Algorithm 1: $FindDBS(\mathcal{G}(t_0,t_c),\theta,(\mathbf{x},T))$

Input: $\mathcal{G}(t_0,t_c)$, θ, $(\mathbf{x},T)$:= an initial temporal subgraph.
Output: $(\mathbf{x}^*,T^*)$:= a DBS in $\mathcal{G}(t_0,t_c)$.

1: **repeat**
2: Solve the CQP problem by IID [8, 10]:
$\mathbf{x} \leftarrow \underset{\mathbf{x}}{\operatorname{argmax}}\, \mathbf{x}^\top A(t_{b+1},t_e)\mathbf{x}$, s.t. $\mathbf{x} \in \triangle^n$.
3: Solve the MDS problem by MDSD [12]:
$T \leftarrow \underset{T}{\operatorname{argmax}}\, \frac{s_{\mathbf{x}}(t_e)-s_{\mathbf{x}}(t_b)}{t_e-t_b}$, s.t. $t_e - t_b \geq \theta$.
4: **until** The value of $g(\mathbf{x},T)$ does not increase.
5: **return** $(\mathbf{x}^*,T^*)=(\mathbf{x},T)$.

Now we are ready to introduce the top-k DBS finding (TDF) problem and prove that it is NP-hard.

DEFINITION 2. *Given a temporal graph $\mathcal{G}(t_0,t_c)$, a minimum duration threshold θ, and a positive integer k, **the problem of top-k DBS finding** (TDF for short) is to compute the set of DBSs in $\mathcal{G}(t_0,t_c)$ that have the top-k largest burstiness.*

THEOREM 1. *The problem in Definition 2 is NP-hard.*

4. FINDING A SINGLE DBS

In this section, we introduce how to find a single DBS by solving a mixed integer programming (MIP) problem.

According to Definition 1, a DBS $(\mathbf{x}^*,T^*)$ is a local maximum point of the following **MIP problem**.

$$\underset{(\mathbf{x},T)}{\operatorname{argmax}}\, g(\mathbf{x},T),\ \text{s.t. } \mathbf{x}\in\triangle^n, T=(t_b,t_e], t_e-t_b\geq\theta. \quad (3)$$

To find a single DBS, we find a local maximum point of the MIP problem by iteratively updating $\mathbf{x}$ and T to monotonously increase $g(\mathbf{x},T)$. Technically, we update $\mathbf{x}$ and T by solving a **constrained quadratic programming (CQP) problem** [27] and a **maximum density segment (MDS) problem** [9], respectively.

First, let us introduce the CQP problem. Given a time interval $T=(t_b,t_e]$ such that $t_e-t_b\geq\theta$, by plugging Equation 2 into Equation 3 and omitting the constant factor t_e-t_b, we transform the MIP problem into the following **CQP problem** [27].

$$\underset{\mathbf{x}}{\operatorname{argmax}}\, \mathbf{x}^\top A(t_{b+1},t_e)\mathbf{x},\ \text{s.t. } \mathbf{x}\in\triangle^n.$$

According to the previous studies on dense subgraph detection [8, 10, 22, 27], a local maximum point $\mathbf{x}^*$ of the CQP problem induces a clique-like dense subgraph $G_{\mathbf{x}^*}(t_{b+1},t_e)$ in the accumulated graph $G(t_{b+1},t_e)$, and the size of $G_{\mathbf{x}^*}(t_{b+1},t_e)$ is usually small if $G(t_{b+1},t_e)$ is sparse.

We can efficiently find a local maximum point of the CQP problem by the IID method [8]. The time complexity of IID is $O(\lambda n)$, where λ is the number of iterations of IID, and n is the volume of V. Since $\mathbf{x}\in\triangle^n$ is usually very sparse, we can efficiently solve the CQP problem using a small submatrix of $A(t_{b+1},t_e)$ [10].

Now we are ready to introduce the MDS problem.

Given $\mathbf{x}\in\triangle^n$, denote by $s_{\mathbf{x}}(t_r)=\sum_{h=0}^{r} q_{\mathbf{x}}(t_h)$ the sum of cohesiveness from t_0 to t_r. We rewrite the burstiness in Equation 2 as $g(\mathbf{x},T)=\frac{s_{\mathbf{x}}(t_e)-s_{\mathbf{x}}(t_b)}{t_e-t_b}$, which is exactly the **slope** between two **points**, $(t_b,s_{\mathbf{x}}(t_b))$ and $(t_e,s_{\mathbf{x}}(t_e))$, in the 2-dimensoinal Cartesian coordinate system.

Using this slope representation of burstiness, we convert the MIP problem into the following **MDS problem** [9].

$$\underset{T}{\operatorname{argmax}}\, \frac{s_{\mathbf{x}}(t_e)-s_{\mathbf{x}}(t_b)}{t_e-t_b},\ \text{s.t. } t_e-t_b\geq\theta.$$

Denote by $\mathcal{P}_{\mathbf{x}}=\{(t_h,s_{\mathbf{x}}(t_h)) \mid t_h\in\{t_0,\ldots,t_c\}\}$ the set of points induced by $\mathbf{x}$. Solving the MDS problem is equivalent to finding $T^*=(t_{b^*},t_{e^*}]$ such that $t_{e^*}-t_{b^*}\geq\theta$, and the points $(t_{b^*},s_{\mathbf{x}}(t_{b^*}))$ and $(t_{e^*},s_{\mathbf{x}}(t_{e^*}))$ in $\mathcal{P}_{\mathbf{x}}$ achieve the global maximum slope.

We can efficiently compute T^* by the MDS detection method [12]. The time complexity of this method is $O(c+1)$, where $c+1$ is the number of snapshots in $\mathcal{G}(t_0,t_c)$.

We summarize the $FindDBS$ method in Algorithm 1, which starts from an initial temporal subgraph $(\mathbf{x},T)$ and finds a DBS by iteratively solving the CQP problem and the MDS problem. Here, an **initial temporal subgraph** $(\mathbf{x},T)$ is a tuple containing the initial values of $\mathbf{x}$ and T, which are used as the initial points to iteratively solve the CQP problem and MDS problem. We will illustrate how to initialize $\mathbf{x}$ and T for each of the proposed methods later.

5. A STATIC BASELINE AND A CHALLENGE FOR ONLINE SOLUTIONS

The TDF problem requires to find multiple local maximum points of the non-concave MIP problem in Equation 3. Normally, to find a set of good solutions to the non-concave MIP problem, we first find multiple DBSs by running $FindDBS$ multiple times with different initializations, and then return the set of DBSs that have the top-k largest burstiness.

Following the above idea, we propose a static baseline method, called $SlideWin$. To obtain as many different initializations as possible, $SlideWin$ initializes T by every time interval $(t_b,t_e]$ such that $t_e-t_b\geq\theta$. For every T, $SlideWin$ initializes $\mathbf{x}$ in the same way as IID [8]. That is, for each $v_i\in V(t_{b+1},t_e)$, we set $\mathbf{x}=\mathbf{u}(v_i)$, where $\mathbf{u}(v_i)\in\triangle^n$ is an n-dimensional vector such that only the i-th entry is 1 and all the other entries are 0's. Initializing $\mathbf{x}$ in this way keeps the size of $V_{\mathbf{x}}$ small and improves the efficiency of IID [8]. A vertex v_i used to initialize $\mathbf{x}$ is called a **seed vertex** for $FindDBS$. At last, $SlideWin$ calls $FindDBS$ with every initialization $(\mathbf{x},T)$ to find multiple DBSs, and return the set of DBSs with the top-k largest burstiness.

The time cost of $SlideWin$ can be measured by the number of times it calls $FindDBS$. In the worst case such that $V(t_{b+1},t_e)=V$ and $t_e-t_b\geq\theta$ for any time internal $(t_b,t_e]$ of the temporal graph $\mathcal{G}(t_0,t_c)$, $SlideWin$ will call $FindDBS$ $\frac{nc(c+1)}{2}$ times, where n is the number of vertices in V, and c is the number of snapshots in $\mathcal{G}(t_1,t_c)$. Denote by τ the maximum time cost to call $FindDBS$ once, the time cost of $SlideWin$ is $O(\frac{\tau nc(c+1)}{2})$.

Denote by κ the maximum number of edges in a snapshot, the memory cost of $SlideWin$ is $O(\kappa c)$ due to its requirement to buffer all the snapshots in $\mathcal{G}(t_0,t_c)$.

The major drawback of $SlideWin$ is that it costs a large amount of time and memory when the number of snapshots c is large. Since $\mathcal{G}(t_0,t_c)$ may be a long or even endless stream of snapshots, the value of c may often be very large in practice, which makes it infeasible to buffer and process all snapshots in $\mathcal{G}(t_0,t_c)$. This significantly reduces the applicability of $SlideWin$.

As a result, we are interested in an online method that maintains the top-k DBSs in real time using a small number of buffered snapshots. However, designing an online method for the TDF problem is still challenging, because

the minimum number of snapshots to be buffered by an online method is lower bounded by the maximum duration of DBSs, which, as shown in Theorem 2, can be as large as $t_c - t_0$ in a temporal graph $\mathcal{G}(t_0, t_c)$.

THEOREM 2. *There exists a temporal graph $\mathcal{G}(t_0, t_c)$ such that the maximum duration of a DBS in $\mathcal{G}(t_0, t_c)$ is $t_c - t_0$.*

Theorem 2 indicates a big challenge in designing online methods for the TDF problem. To find a DBS with duration $t_c - t_0$, we have to buffer all the snapshots in $\mathcal{G}(t_0, t_c)$, which, unfortunately, may be a long or even endless stream of snapshots.

6. DBS DECOMPOSITION

In this section, we make a critical observation: *a long DBS can be easily decomposed into a set of shorter DBSs that have the same or larger burstiness.* We also show that the durations of the shorter DBSs are upper bounded, which makes it possible to design a highly efficient online algorithm for the TDF problem.

Given a DBS $(\mathbf{x}^*, (t_{b^*}, t_{e^*}])$, if $\exists t_h \in (t_{b^*}, t_{e^*}]$ such that $t_{b^*} + \theta \leq t_h \leq t_{e^*} - \theta$, then $(\mathbf{x}^*, (t_{b^*}, t_{e^*}])$ is said to be **decomposable** at time t_h. Otherwise, $(\mathbf{x}^*, (t_{b^*}, t_{e^*}])$ is **indecomposable**. If a DBS $(\mathbf{x}^*, (t_{b^*}, t_{e^*}])$ is decomposable at time t_h, the two temporal subgraphs $(\mathbf{x}^*, (t_{b^*}, t_h])$ and $(\mathbf{x}^*, (t_h, t_{e^*}])$ are called the **components** of $(\mathbf{x}^*, (t_{b^*}, t_{e^*}])$ at time t_h.

Denote by $prev(t_e) = \max\{t_h \mid t_h \leq t_e - \theta\}$ the time of the last snapshot that arrives no later than $t_e - \theta$, by $next(t_b) = \min\{t_h \mid t_h \geq t_b + \theta\}$ the time of the first snapshot that arrives no earlier than $t_b + \theta$, and by $\mathcal{T}(t_b, t_e) = \{t_h \mid next(t_b) \leq t_h \leq prev(t_e)\}$ the set of times between $next(t_b)$ and $prev(t_e)$.

Clearly, $(\mathbf{x}^*, (t_{b^*}, t_{e^*}])$ is indecomposable if and only if $\mathcal{T}(t_{b^*}, t_{e^*}) = \emptyset$. If $\mathcal{T}(t_{b^*}, t_{e^*}) \neq \emptyset$, then $(\mathbf{x}^*, (t_{b^*}, t_{e^*}])$ is decomposable at any time $t_h \in \mathcal{T}(t_{b^*}, t_{e^*})$.

We make an important observation that a decomposable DBS has exactly the same burstiness as its components.

THEOREM 3. *For a decomposable DBS $(\mathbf{x}^*, T^*)$ where $T^* = (t_{b^*}, t_{e^*}]$, any component of $(\mathbf{x}^*, T^*)$ at time $t_h \in \mathcal{T}(t_{b^*}, t_{e^*})$ has the same burstiness as $(\mathbf{x}^*, T^*)$.*

According to Theorem 3, a decomposable DBS $(\mathbf{x}^*, T^*)$ can be decomposed into two components, $(\mathbf{x}^*, (t_{b^*}, t_h])$ and $(\mathbf{x}^*, (t_h, t_{e^*}])$, that have the same burstiness as $(\mathbf{x}^*, T^*)$. However, since $\mathbf{x}^*$ may not be a local maximum point of $g(\mathbf{x}, (t_{b^*}, t_h])$ or $g(\mathbf{x}, (t_h, t_{e^*}])$, the components of $(\mathbf{x}^*, T^*)$ may not be DBSs.

If a component $(\mathbf{x}', T')$ of a decomposable DBS $(\mathbf{x}^*, T^*)$ is not a DBS, we can further increase the burstiness of $(\mathbf{x}', T')$ by feeding $(\mathbf{x}', T')$ as the initial temporal subgraph into $FindDBS$. The output of $FindDBS$ is a new DBS $(\mathbf{x}^*_{new}, T^*_{new})$ such that $g(\mathbf{x}^*_{new}, T^*_{new}) > g(\mathbf{x}', T') = g(\mathbf{x}^*, T^*)$.

If $(\mathbf{x}^*_{new}, T^*_{new})$ is also decomposable, we can keep decomposing it and updating its components by $FindDBS$. Eventually, we can decompose a decomposable DBS $(\mathbf{x}^*, T^*)$ into a set of indecomposable DBSs with the same or even larger burstiness.

Interestingly, the duration of any indecomposable DBS $(\mathbf{x}^*, (t_{b^*}, t_{e^*}])$ is upper bounded by $t_{e^*} - prev(prev(t_{e^*}))$, because $\mathcal{T}(t_{b^*}, t_{e^*}) = \emptyset$ if and only if $t_{b^*} > prev(prev(t_{e^*}))$. This makes it possible to detect indecomposable DBSs in an online manner without buffering all snapshots.

Another reason to find indecomposable DBSs is that most DBSs with large burstiness are indecomposable in practice. We will analyze this phenomenon using the experiments in Table 7 and Section 8.2.

7. ONLINE TOP-K DBS DETECTION

Enabled by the observations in Section 6, in this section, we define the online top-k DBS finding (OTDF) problem and develop an efficient online DBS detection method named $TopkDBSOL$.

7.1 The OTDF problem

Denote by $\mathcal{M}(t_c)$ the set of indecomposable DBSs in $\mathcal{G}(t_0, t_c)$. We define the OTDF problem as follows.

DEFINITION 3. *Given $\mathcal{M}(t_{c-1})$ and a sequence of buffered snapshots $\mathcal{G}(t_s, t_c)$, **the problem of online top-k density bursting subgraph finding** (OTDF for short) is to compute the top-k indecomposable DBSs in $\mathcal{M}(t_c)$.*

The OTDF problem is NP-hard following the same reduction in the proof of Theorem 1.

Since the duration of any indecomposable DBS ending at time t_c is upper bounded by $t_c - prev(prev(t_c))$, we set $t_s = prev(prev(t_c))$, so that any indecomposable DBS in $\mathcal{M}(t_c) \setminus \mathcal{M}(t_{c-1})$ is an indecomposable DBS in $\mathcal{G}(t_s, t_c)$.

However, an indecomposable DBS $(\mathbf{x}^*, T^*)$ in $\mathcal{G}(t_s, t_c)$ may not be an indecomposable DBS in $\mathcal{M}(t_c) \setminus \mathcal{M}(t_{c-1})$, because, without the snapshots in $\mathcal{G}(t_0, t_{s-1})$, we cannot verify whether or not T^* is the global maximum point of $g(\mathbf{x}^*, T)$ in $\mathcal{G}(t_0, t_c)$. As a result, we cannot compute $\mathcal{M}(t_c)$ directly.

To tackle this problem, we compute a superset of $\mathcal{M}(t_c)$ by finding a set of indecomposable DBS candidates, and return the top-k indecomposable DBS candidates as the final result. Here, an **indecomposable DBS candidate** in $\mathcal{G}(t_0, t_c)$ is a temporal subgraph $(\hat{\mathbf{x}}, \hat{T})$ that has a begin time $t_{\hat{b}} \in (t_0, t_c]$, and is an indecomposable DBS in $\mathcal{G}(t_{\hat{b}}, t_c)$. Obviously, the duration of any indecomposable DBS candidate with an end time t_c is also upper bounded by $t_c - prev(prev(t_c))$.

According to the definition of indecomposable DBS candidate, any indecomposable DBS in $\mathcal{G}(t_0, t_c)$ is an indecomposable DBS candidate in $\mathcal{G}(t_0, t_c)$. Therefore, the set of indecomposable DBS candidates in $\mathcal{G}(t_0, t_c)$, denoted by $\mathcal{D}(t_c)$, is a super set of $\mathcal{M}(t_c)$.

As illustrated in the rest of this section, given $\mathcal{D}(t_{c-1})$ and the buffered snapshots $\mathcal{G}(t_s, t_c)$, we can efficiently find a good solution to the OTDF problem in the following three steps. First, we find **new indecomposable DBS candidates** (**NDBSC**) in $\mathcal{D}(t_c) \setminus \mathcal{D}(t_{c-1})$. Second, we update **old indecomposable DBS candidates** (**ODBSC**) in $\mathcal{D}(t_{c-1})$. Last, we return the top-k indecomposable DBS candidates in $\mathcal{D}(t_c)$ as the final result.

7.2 Finding the Set of NDBSCs

Since NDBSCs are indecomposable DBS candidates in $\mathcal{D}(t_c) \setminus \mathcal{D}(t_{c-1})$, a NDBSC is an indecomposable DBS candidate with an end time t_c. Since $t_s = prev(prev(t_c))$ and the duration of any indecomposable DBS candidate ending at time t_c is upper bounded by $t_c - prev(prev(t_c))$, the buffered snapshots $\mathcal{G}(t_s, t_c)$ contains all NDBSCs. As a result, to obtain the set of NDBSCs, we only need to find indecomposable DBSs ending at time t_c in $\mathcal{G}(t_s, t_c)$.

A straightforward way to find these indecomposable DBSs in $\mathcal{G}(t_s, t_c)$ is to call $SlideWin$. However, $SlideWin$ is inefficient because it calls $FindDBS$ too many times by using every vertex $v_i \in V(t_{b+1}, t_e)$ as a seed vertex.

Interestingly, we observed that a large proportion of the vertices in $V(t_{b+1}, t_e)$ are not contained in any top-k indecomposable DBS. Inspired by this observation, we first derive an upper bound for the burstiness of any indecomposable DBS that contains a vertex v_i, then we propose a smart initialization heuristic, which applies the upper bound

Algorithm 2: $FindNDBSC(\mathcal{G}(t_s,t_c),\theta,\mathcal{D}(t_{c-1}),k)$

Input: $\mathcal{G}(t_s,t_c),\theta,\mathcal{D}(t_{c-1}),k$.
Output: $\mathcal{N}(t_c)$:= the set of NDBSCs.

1: $\mathcal{N}(t_c)\leftarrow\emptyset$ and compute $\epsilon_k(t_{c-1})$.
2: **for** each $t_b\in\{t_s,\ldots,t_{c-1}\}$ such that $t_c-t_b\geq\theta$ **do**
3: $T\leftarrow(t_b,t_c]$.
4: **for** each $v_i\in V(t_{b+1},t_c)$ **do**
5: **if** $\alpha(v_i,T)\geq\epsilon_k(t_{c-1})$ **then**
6: $\mathbf{x}\leftarrow\mathbf{o}(v_i,T)$.
7: $(\hat{\mathbf{x}},\hat{T})\leftarrow FindDBS(\mathcal{G}(t_s,t_c),\theta,(\mathbf{x},T))$.
8: **if** $(\hat{\mathbf{x}},\hat{T})\notin\mathcal{D}(t_{c-1})$ **then**
9: $\mathcal{N}(t_c)\leftarrow\mathcal{N}(t_c)\cup(\hat{\mathbf{x}},\hat{T})$.
10: **end if**
11: **end if**
12: **end for**
13: **end for**
14: **return** $\mathcal{N}(t_c)$.

Algorithm 3: $UpdateODBSC(\mathcal{G}(t_s,t_c),\theta,\mathcal{D}(t_{c-1}))$

Input: $\mathcal{G}(t_s,t_c)$, θ, $\mathcal{D}(t_{c-1})$.
Output: $\mathcal{U}(t_c)$:= the set of updated ODBSCs.

1: $\mathcal{U}(t_c)\leftarrow\emptyset$.
2: **for** each $(\hat{\mathbf{x}},\hat{T})\in\mathcal{D}(t_{c-1})$ **do**
3: **if** $t_{\hat{b}}\geq t_s$ **then**
4: $\mathcal{U}(t_c)\leftarrow\mathcal{U}(t_c)\cup FindDBS(\mathcal{G}(t_s,t_c),\theta,(\hat{\mathbf{x}},\hat{T}))$.
5: **else if** $\exists t_h\in(t_s,prev(t_c)]:g(\hat{\mathbf{x}},(t_h,t_c])>g(\hat{\mathbf{x}},\hat{T})$ **then**
6: $\mathcal{U}(t_c)\leftarrow\mathcal{U}(t_c)\cup FindDBS(\mathcal{G}(t_s,t_c),\theta,(\hat{\mathbf{x}},(t_h,t_c]))$.
7: **else**
8: $\mathcal{U}(t_c)\leftarrow\mathcal{U}(t_c)\cup(\hat{\mathbf{x}},\hat{T})$.
9: **end if**
10: **end for**
11: **return** $\mathcal{U}(t_c)$.

to filter out a large proportion of vertices that are not contained in any top-k indecomposable DBS. This significantly reduces the number of calls of $FindDBS$, and achieves a speedup of two orders of magnitudes in our experiments.

Next, we show that $\alpha(v_i,T)=\frac{\max_j A_{ij}(t_{b+1},t_e)}{t_e-t_b}$ is an upper bound of the burstiness of any indecomposable DBS that contains a vertex v_i during a time interval $(t_b,t_e]$.

THEOREM 4. *For any indecomposable DBS* $(\mathbf{x}^*,T^*)$*, if* $v_i\in(\mathbf{x}^*,T^*)$*, then* $g(\mathbf{x}^*,T^*)\leq\alpha(v_i,T^*)$.

Now, we introduce the **smart initialization** method that only calls $FindDBS$ for a small set of selected seed vertices.

Denote by $\epsilon_k(t_{c-1})$ the k-th largest burstiness of all ODBSCs in $\mathcal{D}(t_{c-1})$. By Theorem 4, for any time interval T, if $\alpha(v_i,T)<\epsilon_k(t_{c-1})$, then v_i is not contained in any indecomposable DBS $(\mathbf{x}^*,T^*)$ such that $g(\mathbf{x}^*,T^*)\geq\epsilon_k(t_{c-1})$ and $T^*=T$.

Since we are only interested in the top-k indecomposable DBSs whose burstiness is larger than $\epsilon_k(t_{c-1})$, we only use a vertex v_i as a seed vertex for $FindDBS$ if $\alpha(v_i,T)\geq\epsilon_k(t_{c-1})$, and skip each vertex v_h such that $\alpha(v_h,T)<\epsilon_k(t_{c-1})$.

For a seed vertex v_i such that $\alpha(v_i,T)\geq\epsilon_k(t_{c-1})$, we initialize $FindDBS$ by $\mathbf{x}=\mathbf{o}(v_i,T)=0.5\mathbf{u}(v_i)+0.5\mathbf{u}(v_j)$, where $v_j=\text{argmax}_{v_j}A_{ij}(t_{b+1},t_e)$ is the nearest neighbor of v_i in $G(t_{b+1},t_e)$. As to be shown in Theorem 6, by setting $\mathbf{x}=\mathbf{o}(v_i,T)$, we achieve a 2-approximation of the top-1 indecomposable DBS.

Algorithm 2 summarizes $FindNDBSC$, which efficiently finds a set of NDBSCs by finding indecomposable DBSs with an end time t_c in $\mathcal{G}(t_s,t_c)$. The smart initialization is performed in steps 5-6.

Algorithm 4: $TopkDBSOL(\mathcal{G}(t_s,t_c),\theta,\mathcal{D}(t_{c-1}),k)$

Input: $\mathcal{G}(t_s,t_c)$, θ, $\mathcal{D}(t_{c-1})$, k.
Output: $\mathcal{D}(t_c)$ and $TOPK(\mathcal{D}(t_c))$.

1: $\mathcal{N}(t_c)\leftarrow FindNDBSC(\mathcal{G}(t_s,t_c),\theta,\mathcal{D}(t_{c-1}),k)$.
2: $\mathcal{U}(t_c)\leftarrow UpdateODBSC(\mathcal{G}(t_s,t_c),\theta,\mathcal{D}(t_{c-1}))$.
3: $\mathcal{D}(t_c)\leftarrow\mathcal{N}(t_c)\cup\mathcal{U}(t_c)$.
4: **return** $\mathcal{D}(t_c)$ and $TOPK(\mathcal{D}(t_c))$.

7.3 Updating the Set of ODBSCs

When a new snapshot arrives at time t_c, the ODBSCs in $\mathcal{D}(t_{c-1})$ need to be updated because a ODBSC in $\mathcal{D}(t_{c-1})$ may not be necessarily an indecomposable DBS candidate in $\mathcal{D}(t_c)$.

Denote by $(\hat{\mathbf{x}},\hat{T})$, $\hat{T}=(t_{\hat{b}},t_{\hat{e}}]$ an ODBSC in $\mathcal{D}(t_{c-1})$. If $t_{\hat{b}}\geq t_s$, we can directly update $(\hat{\mathbf{x}},\hat{T})$ by calling $FindDBS(\mathcal{G}(t_s,t_c),\theta,(\hat{\mathbf{x}},\hat{T}))$.

However, when $t_{\hat{b}}<t_s$, we cannot update $(\hat{\mathbf{x}},\hat{T})$ by calling $FindDBS(\mathcal{G}(t_s,t_c),\theta,(\hat{\mathbf{x}},\hat{T}))$, because the snapshots before time t_s are not buffered in $\mathcal{G}(t_s,t_c)$. For such ODBSCs, we show in Theorem 5 a necessary and sufficient condition to verify whether it is contained in $\mathcal{D}(t_c)$.

THEOREM 5. *For any ODBSC* $(\hat{\mathbf{x}},\hat{T})\in\mathcal{D}(t_{c-1})$ *where* $\hat{T}=(t_{\hat{b}},t_{\hat{e}}]$ *and* $t_{\hat{b}}<t_s$, $(\hat{\mathbf{x}},\hat{T})\in\mathcal{D}(t_c)$ *if and only if* $\nexists t_h\in(t_s,prev(t_c)]$ *such that* $g(\hat{\mathbf{x}},(t_h,t_c])>g(\hat{\mathbf{x}},\hat{T})$.

For any ODBSC $(\hat{\mathbf{x}},\hat{T})$ with a begin time $t_{\hat{b}}<t_s$, Theorem 5 provides a straightforward way to verify whether it is contained in $\mathcal{D}(t_c)$ as follows. If $\nexists t_h\in(t_s,prev(t_c)]$ such that $g(\hat{\mathbf{x}},(t_h,t_c])>g(\hat{\mathbf{x}},\hat{T})$, then $(\hat{\mathbf{x}},\hat{T})\in\mathcal{D}(t_c)$, and we keep $(\hat{\mathbf{x}},\hat{T})$ in $\mathcal{D}(t_c)$. If $\exists t_h\in(t_s,prev(t_c)]$ such that $g(\hat{\mathbf{x}},(t_h,t_c])>g(\hat{\mathbf{x}},\hat{T})$, then $(\hat{\mathbf{x}},\hat{T})\notin\mathcal{D}(t_c)$. In this case, we update $(\hat{\mathbf{x}},\hat{T})$ by calling $FindDBS(\mathcal{G}(t_s,t_c),\theta,(\hat{\mathbf{x}},(t_h,t_c]))$, which will find a NDBSC with a larger burstiness than the ODBSC $(\hat{\mathbf{x}},\hat{T})$.

We summarize $UpdateODBSC$ and $TopkDBSOL$ in Algorithm 3 and Algorithm 4, respectively.

Next, we show that $TopkDBSOL$ achieves a 2-approximation of the top-1 indecomposable DBS in $\mathcal{G}(t_0,t_c)$.

THEOREM 6. *TopkDBSOL produces a 2-approximation of the top-1 indecomposable DBS in* $\mathcal{G}(t_0,t_c)$.

8. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the proposed methods $SlideWin$ (**SW**) and $TopkDBSOL$ (**OL**), and compare OL with four baseline methods, $DenseAlert$ (**DA**) [30], $DenseStream$ (**DS**) [30], **DYN** [14] and **FIDES**$^+$ [25]. We also implement an algorithm named $TopkDBSOL^{nsi}$ (**OL**$^{\mathbf{nsi}}$) by disabling the smart initialization of OL to further evaluate the effect of the smart initialization introduced in Section 7.2.

Table 1 summarizes the default parameter settings. For all methods, if not specified otherwise, by default the minimum duration of detected temporal subgraphs is $\theta=3$, and the number of temporal subgraphs to detect is $k=100$. For DA and DS, the order of the input tensor is set to $N=2$ to deal with matrix input. The window size of DA is optimally set to $\triangle T=2$. The approximation error of DYN is set to $\epsilon=0.01$ by default. The number of candidate intervals for FIDES$^+$ is set to $C=100$.

The code of all baseline methods is provided by their authors. Our algorithms are implemented in C++. All experiments are conducted on a PC with Core-i7-3370 CPU

Table 1: Summary of default parameter settings.

Methods	ALL	DA	DS	DYN	FIDES$^+$
Settings	$\theta = 3$ $k = 100$	$N = 2$ $\triangle T = 2$	$N = 2$	$\epsilon = 0.01$	$C = 100$

(3.40 GHz), 16GB main memory, and a 5400 rpm hard drive running Windows 7 OS.

The efficiency of a method is measured by **running time** (**RT**). The quality of a detected temporal subgraph is measured by **edge density burstiness** (**EDB**), which is the speed that the detected temporal subgraph accumulates edge density [11, 32, 20].

Here, edge density is a well recognized evaluation metric to measure the average connection strength between the vertices of a subgraph [11, 20, 32]. Denote by $G_S(t_h)$ a subgraph induced by a set of vertices S from a snapshot $G(t_h)$, $t_h \in \{t_1, \ldots, t_c\}$. The **edge density** (**ED**) of $G_S(t_h)$ is computed by

$$\mathrm{ED}_S(t_h) = \frac{\sum_{v_i \in S} \sum_{v_j \in S} A_{ij}(t_h)}{|S|(|S|-1)},$$

where $|S|$ is the volume of S.

Since a snapshot $G(t_h)$ carries the updates on the edge weights of the accumulated graph $G(t_0, t_{h-1})$, $\mathrm{ED}_S(t_h)$ is actually the amount of edge density accumulated by the subgraph $G_S(t_h)$ during time interval $(t_{h-1}, t_h]$.

Denote by $(S, (t_b, t_e])$ a detected temporal subgraph induced by S from the snapshots of $\mathcal{G}(t_{b+1}, t_e)$. The edge density accumulated by $(S, (t_b, t_e])$ during time interval $(t_b, t_e]$ is the sum of the edge density accumulated by each of the subgraphs $G_S(t_{b+1}), \ldots, G_S(t_e)$, which is computed by

$$\sum_{h=b+1}^{e} \mathrm{ED}_S(t_h) = \frac{\sum_{v_i \in S} \sum_{v_j \in S} A_{ij}(t_{b+1}, t_e)}{|S|(|S|-1)}.$$

Since the duration for the temporal graph $(S, (t_b, t_e])$ to accumulate its edge density is $t_e - t_b$, we define the **EDB** of $(S, (t_b, t_e])$ as $\mathrm{EDB} = \frac{1}{t_e - t_b} \sum_{h=b+1}^{e} \mathrm{ED}_S(t_h)$, which is exactly the speed that $(S, (t_b, t_e])$ accumulates its edge density.

Obviously, a larger EDB means a faster speed in forming a strongly connected dense subgraph, which further indicates a higher quality of the detected temporal subgraph.

We use the following five public real world data sets.

Facebook Wall Posts (FBWP) Data Set [33]. This data set is the wall post network of Facebook. Each vertex is a user. Each edge represents the wall post activity between two users. The edge weight is the number of wall posts between two users. A DBS in FBWP identifies a fast emerging community of users who actively interact with each other.

ENRON Data Set [33]. This data set is the email communication network of the Enron company. Each vertex is an employee. Each edge represents the email communication between two employees. The edge weight is the number of emails sent between two users. A DBS in this data set identifies a fast emerging group of employees who frequently communicate with each other.

DBLP Coauthorship (DBLP) Data Set [33]. This data set is the co-authorship network in DBLP. Each vertex is an author. Each edge represents the co-authorship between two authors. The edge weight is the number of coauthored publications. Finding DBSs in this type of data identifies fast emerging groups of researchers who frequently collaborate with each other.

TAXI-1 and TAXI-2 Data Sets. These data sets are constructed using the taxi trips in July 2017 in the city of Chicago [31].We uniformly partition the city into 362 blocks, each of which corresponds to a vertex in the temporal graph. The period of one day is uniformly divided into 96 snapshots, each of which corresponds to a time interval of 15 minutes. The taxi trips during the same time interval of all days are grouped into the same snapshot. For each snapshot, the edge weight between two vertices is the number of taxi trips between two locations. TAXI-1 and TAXI-2 consist of the taxi trips of weekdays and weekends, respectively. As shown in the first case study of Section 8.1, a detected DBS in these data sets reveals a burst of taxi trips among a group of locations, which reveals interesting travel patterns of people.

Keyword Association Network (KAN) Data Set. This data set is a keyword association network [3] extracted from the DBLP-Citation-Network V10 data set [17].We use the abstracts of the papers published in some well established data mining venues, such as KDD, ICDM, SDM, PKDD, PAKDD, TKDE and TKDD. Each vertex is a keyword of a paper. Each edge represents the co-occurrence of two keywords in the same abstract. The edge weight is the number of co-occurring abstracts. As shown in the second case study of Section 8.1, a DBS in this data set identifies a set of keywords that accurately describe a fast emerging research topic.

Table 2 shows some details about the data sets. Since the time intervals (TI for short) between neighbor snapshots have the same duration, we use TI as the base unit of time to measure the durations of time intervals, and directly define the arrival time of a snapshot $G(t_i)$ as $t_i = i$, which means $G(t_i)$ arrives at the end of the i-th time interval.

Table 2: Detailed information of data sets. "V", "E", "S" and "TI" represent "Vertices", "Edges", "Snapshots" and "Time Interval", respectively. TI is the time interval between the arrival time of two neighbor snapshots.

Data Set	# V	# E	# S	TI
FBWP	46,952	585,932	1,591	1 DAY
ENRON	87,273	920,478	2,222	1 DAY
DBLP	1,282,461	7,354,929	45	1 YEAR
TAXI-1	362	88,547	96	15 MIN.
TAXI-2	362	70,202	96	15 MIN.
KAN	33,967	142,068	20	1 YEAR

Table 3: The information of locations in Figure 2.

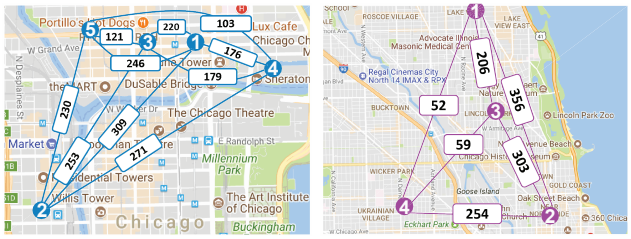
ID	Locations in Figure 2(a).
1	Downtown area, shopping centre (i.e., Nordstrom), cosmetic boutique restaurants and so on.
2	Office buildings and companies.
3	Various restaurants.
4	Open bars, restaurants.
5	Companies, restaurants and bars.

ID	Locations in Figure 2(b).
1	16 bars or clubs that are still open during 22:30-02:00.
2	College town surrounded by colleges and universities.
3	Oz park, clubs, midnight pubs and restaurants.
4	3 bars or clubs within 150 meters range.

8.1 Case Studies

In this subsection, we demonstrate the effectiveness of OL in discovering meaningful patterns by the case studies on TAXI-1, TAXI-2, KAN, and DBLP.

First, we show some interesting patterns of taxi trips discovered by finding the most significant DBSs from TAXI-1 and TAXI-2. Figure 2(a) shows the bursting taxi trips between a set of locations during 17:30-18:45 on weekdays. The location 2 is surrounded by office buildings, the locations 1, 3, 4 and 5 are surrounded by shopping centres and



(a) Weekdays (17:30-18:45) (b) Weekends (22:45-23:45)

Figure 2: The bursting taxi trips between a set of locations in the city of Chicago in US. (a) shows the number of taxi trips between a set of locations during 17:30-18:45 on TAXI-1. (b) shows the number of taxi trips between a set of locations during 22:45-23:45 on TAXI-2. Detailed information of the locations in (a) and (b) are listed in Table 3.

Table 4: The topics and durations of detected DBSs by OL.

ID	Top-6 sets of keywords (Years) detected by OL
1	Deep learning, image patch, feature representation. (2012-2015)
2	Multiple type, link prediction, heterogeneous network. (2011-2014)
3	Visual word, local feature, inverted index. (2010-2013)
4	Domain adaption, conditional probability, semg signal. (2010-2013)
5	Graph laplacian, cluster label, feature selection algorithm, spectral feature, unsupervised feature selection. (2012-2015)
6	Loss function, machine learning, vast amount, large scale. (2011-2015)

restaurants. The number of taxi trips between these locations bursts during 17:30-18:45 on weekdays, because people often go shopping and dine out after a day's busy work. Figure 2(b) shows an even more interesting pattern of bursting taxi trips during 22:45-23:45 on weekends.

Where are people going at midnight? The answer lies in the locations 1, 3 and 4, where the neighborhoods have many mid-night pubs and restaurants that open late at night.

Who are those people? We investigate the neighborhood of location 2 and find that it is a college town surrounded by several universities and colleges. Most of the taxi trips are related to location 2. Obviously, midnight parties at pubs are one of the favourite weekend entertainments of young students. The pubs and restaurants start to close at 22:30, and many people go home before 00:00, therefore, the number of taxi trips bursts during 22:45-23:45.

In this simple case study, finding DBSs from the temporal network of taxi trips discovers interesting travel patterns that provide useful insights into people's behavior patterns. Such patterns can be used to, for example, improve taxis dispatch plans and develop better public traffic designs.

Second, we show some interesting research topics discovered by finding DBSs from the KAN data set. Table 4 shows the top-6 sets of keywords detected by OL. Each row shows a set of keywords, which describe a meaningful research topic that emerges at the fastest speed during the corresponding years. These topics are: (1) learning representations by deep learning; (2) link prediction in heterogeneous networks; (3) image retrieval based on visual word; (4) domain adaption for semg signal; (5) unsupervised feature selection based on graph spectrum; and (6) large scale machine learning.

Table 5: The topics and durations of detected DBSs by DA.

ID	Top-2 sets of keywords (Years) detected by DA
1	Different population, dynamic feature, critical challenge, twitter data, spatial correlation, shared information, static feature, extensive experimental evaluation, dynamic pattern, effective model, model training, multi-task learning. (2013-2016)
2	Similarity matrix, class label, loss function, scalable approach, target data, target set, concept drift, wide range, optimization framework, class membership, transfer learning, unsupervised model, cluster ensemble, soft constraint, bregman divergence, target distribution. (2011-2014)

Table 6: The researcher groups and durations of detected DBSs by OL.

ID	Top-4 researcher groups (Years) detected by OL
1	Fatos Xhafa, Leonard Barolli, Evjola Spaho. (2010-2013)
2	Tomoya Enokido, Makoto Takizawa, Ailixier Aikebaier. (2009-2012)
3	Patrick Girard, Arnaud Virazel, Luigi Dilillo, Alberto Bosio". (2008-2012)
4	Hiroshi Harada, Chin-Sean Sum, Tuncer Baykas, Junyi Wang, Shuzo Kato. (2007-2010)

Table 5 shows the top-2 sets of keywords detected by DA. Comparing to the keywords detected by OL, each set of the keywords detected by DA contains more words, but these words hardly describe a meaningful and focused topic.

The sets of keywords detected by DS, DYN and FIDES$^+$ contain hundreds of keywords, which are not shown here limited by space. We comprehensively analyze the performance of these methods in Section 8.4.

To validate the emerging years of the topics, we crawled the annual popularity of each keyword from AMINER (`https://aminer.org/`), and compute the popularity of each topic by the product of the popularities of all related keywords. As shown in Figure 3 and Tables 4 and 5, the durations of all topics detected by OL accurately highlight the time when their popularities rose fast. However, the durations of the topics detected by DA miss the time when those topics quickly gained popularity.

In this case study, by finding DBSs in the network of KAN, OL effectively detects hot research topics, as well as the time interval when their popularities rise fast.

Last, we show some emerging groups of researchers discovered by using OL to find DBSs from the DBLP data set. Table 6 shows the top-4 detected researcher groups and their emerging years. To validate the emerging years of these researcher groups, we use the names of all authors in the same group as a query to search Google Scholar, and regard the number of returned results as the popularity of the researcher group. As shown in Figure 4 and Table 6, the durations of all researcher groups detected by OL accurately highlight the time when their popularities arise fast.

In summary, the above case studies show that finding DBSs from the temporal graphs of TAXI-1, TAXI-2, KAN and DBLP discovers interesting patterns in the real world.

8.2 Effect of Parameters

In this subsection, we analyze the effects of parameters θ and k on the proposed methods OL, OL$^{\text{nsi}}$ and SW. The performance of each method is evaluated in RT and the average EDB of the top-k detected DBSs.

Since SW detects DBSs by scanning every possible time intervals of a temporal graph in a brute force manner, it usu-

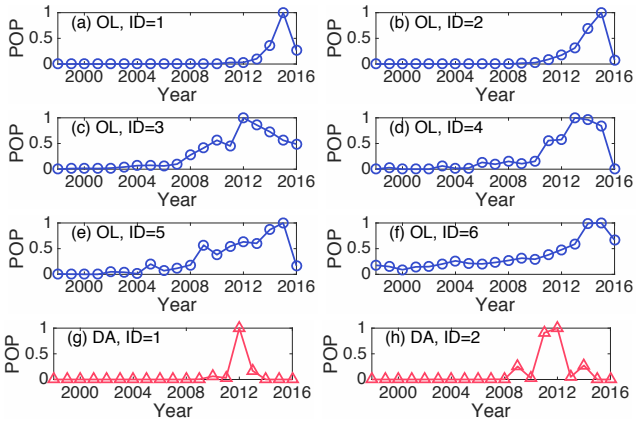


Figure 3: The normalized popularity (POP) of detected topics on KAN. x-axis is the time line of years. (a)-(f) show the POP of OL. (g)-(h) show the POP of DA. The IDs are shown in Tables 4 and 5.

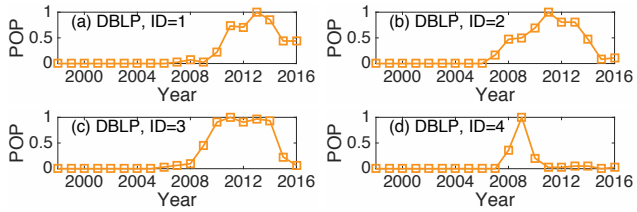


Figure 4: The normalized popularity (POP) of detected researcher groups on DBLP. The x-axis is the time line in years. The IDs are shown in Table 6.

ally produces a very good DBS detection result. Therefore, SW can serve as a strong baseline to compare with OL^{nsi} and OL. However, the major problem of SW is that it runs too slow on the full data sets of FBWP, ENRON and DBLP. Therefore, we sample a small data set from each full data set to conduct our experiments in this subsection.

For DBLP, we first sample 10,000 vertices from the accumulated graph of all snapshots by a breath-first search, and then use the temporal subgraph induced by the sampled vertices from each snapshot to form a sample of a snapshot. For FBWP and ENRON, we use the complete set of vertices, and sample two temporal graphs with continuous durations of 30 days and 50 days, respectively.

Figure 5 shows the effect of θ. We can see that the average EDB decreases when θ increases. This is because most social events in FBWP only heat up at the fastest speed for a short time, the email frequencies between colleagues in ENRON do not stay high for long, and most co-authors in DBLP do not stay highly productive together for a long time. When θ increases, the RT of OL and OL^{nsi} increases, because a large θ increases the number of buffered snapshots in $\mathcal{G}(t_s, t_c)$, which increases the running time of $FindDBS$ in Algorithm 2 and Algorithm 3. The RT of SW stays stable, because increasing θ increases the time cost of $FindDBS$, however, reduces the number of times to call $FindDBS$ in $SlideWin$.

Figure 6 shows the effect of k. Since the detected DBSs are ranked by burstiness, the average EDB decreases when k increases. The efficiency of OL^{nsi} and SW are irrelevant to k, thus their RT stays stable. The RT of OL increases, because a larger k decreases the value of $\epsilon_k(t_{c-1})$, which is the k-th largest burstiness of all ODBSCs in $\mathcal{D}(t_{c-1})$. This increases the number of times to call $FindDBS$ in Algorithm 2.

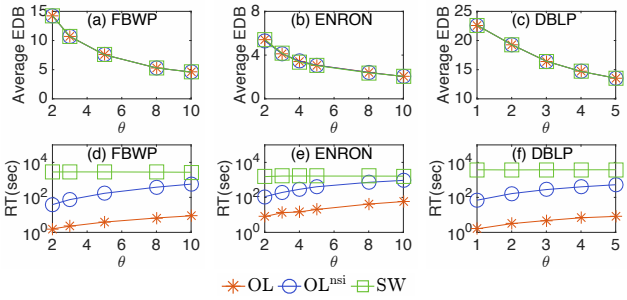


Figure 5: The effect of parameter θ when $k = 30$.

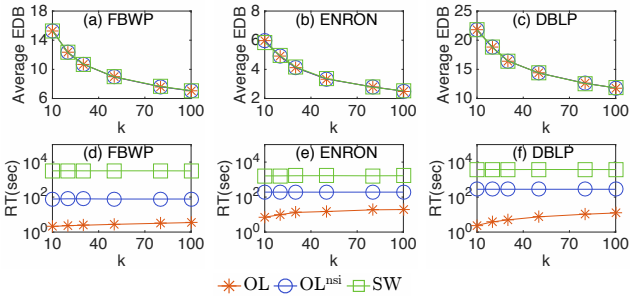


Figure 6: The effect of parameter k when $\theta = 3$.

As shown in Figures 5 and 6, the average EDB of OL and SW are highly comparable, because both OL and SW call $FindDBS$ to find DBSs. The difference lie in their ways to buffer snapshots and generate initializations for $FindDBS$. SW buffers all the snapshots and calls $FindDBS$ with as many initializations as possible in a brute force manner, however, OL buffers a limited number of snapshots and applies smart initialization to significantly reduce the number of times to call $FindDBS$. As a result, OL is dramatically faster than SW by orders of magnitudes without sacrificing the quality of detected DBSs.

We can also see that the average EDB performance of OL and OL^{nsi} are identical for all values of θ and k. This demonstrates that the smart initialization proposed in Section 7.2 does not affect the average EDB of OL.

Table 7 shows the effect of θ on the numbers of indecomposable DBS and decomposable DBS. Both #INDEC and #DEC decrease when θ increases, because θ is the minimum duration threshold of DBS, and a larger θ rules out more DBSs whose duration is smaller than θ.

We can also see from Table 7 that #DEC is less than 1% of all detected DBSs, and the decomposable DBS with the largest burstiness has a very low RANK. These results verify our observation in Section 6: regarding the necessary condition in Theorem 3 for a DBS to be decomposable, most DBSs with large burstiness are indecomposable in practice.

8.3 Scalability Analysis

In this subsection, we compare the scalability of OL, OL^{nsi} and SW on FBWP, ENRON and DBLP. To obtain a series of samples of different sizes, for each data set, we sample four temporal graphs as follows.

First, we start a breadth first search from a randomly picked vertex on the accumulated graph of all snapshots. Second, let S be the set of all vertices visited by the breadth first search, we use S to induce a subgraph from each of the snapshots of the original temporal graph. Last, we use the sequence of induced subgraphs as a sampled temporal graph.

Table 7: The numbers of indecomposable DBSs (#INDEC) and decomposable DBSs (#DEC) detected by SW. We rank all DBSs in descending order of burstiness, and RANK is the rank of the decomposable DBS with the largest burstiness.

Data set	θ	# INDEC	# DEC	RANK
FBWP	2	35,788	47	310
	5	29,988	0	N/A
	10	23,806	0	N/A
ENRON	2	9,074	14	244
	5	6,197	0	N/A
	10	4,355	0	N/A
DBLP	2	4,985	51	613
	3	3,915	6	402
	5	2,684	0	N/A

Table 8: The numbers of vertices (#V) and edges (#E) of the data sets sampled from FBWP, ENRON and DBLP. The 5-th data set (i.e., ID=5) is simply the complete data set.

ID	FBWP		ENRON		DBLP	
	#V ($\times 10^3$)	#E ($\times 10^5$)	#V ($\times 10^3$)	#E ($\times 10^5$)	#V ($\times 10^3$)	#E ($\times 10^5$)
1	2.0	0.2	1.0	0.5	20.0	2.5
2	6.0	0.9	5.0	2.3	100.0	11.6
3	10.0	1.8	9.0	4.8	300.0	30.8
4	20.0	3.8	15.0	6.3	500.0	46.0
5	47.0	5.9	87.3	9.2	1282.5	73.5

The numbers of vertices and edges of the sampled temporal graphs are listed in Table 8.

Figure 7 shows the RT performance of OL, OL$^{\text{nsi}}$ and SW, respectively. We do not report the RT of SW on FBWP and ENRON, because SW cannot finish in 24 hours due to its quadratic time complexity with respect to the number of snapshots. Since the smart initialization effectively reduces the number of calls of *FindDBS* in Algorithm 2, OL always completes in less than 100 seconds, which is 2 orders of magnitudes faster than OL$^{\text{nsi}}$ and is at least 4 orders of magnitudes faster than SW.

8.4 Comparison with Baseline Methods

In this subsection, we evaluate the DBS detection performance of OL and four baseline methods, such as **DA** [30], **DS** [30], **DYN** [14] and **FIDES**$^+$ [25].

The methods OL, DA, DS and FIDES$^+$ model an accumulated graph as a weighted graph that accumulates the real-valued weights of edge updates. However, DYN only accepts binary edge updates, and it models an accumulated graph as an unweighted graph that does not add up the weights of edge updates.

To accommodate the above inconsistency between the inputs of DYN and the other methods, we convert the **original data sets** FBWP, ENRON and DBLP into temporal graphs with binary edge updates by quantizing the weights of all edge updates. If an edge is updated by a positive weight between vertices v_i and v_j at time t, we set $A_{ij}(t) = 1$; otherwise, we set $A_{ij}(t) = 0$. We write the **converted data sets** as FBWP-B, ENRON-B and DBLP-B, respectively.

It is also difficult to directly compare the outputs of DA, DS, DYN and FIDES$^+$ with that of OL. Because OL is designed to find top-k dense temporal subgraphs, however, DA, DS and DYN cannot maintain more than one dense subgraph at a time, and FIDES$^+$ only finds one dense temporal subgraph in a temporal graph.

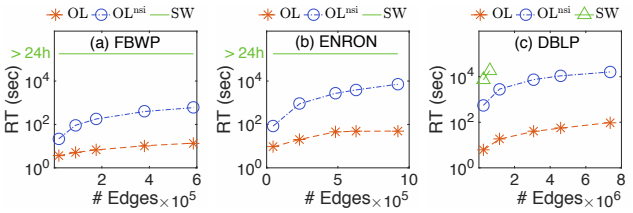


Figure 7: The RT of OL, OL$^{\text{nsi}}$ and SW on the temporal graphs sampled from FBWP, ENRON and DBLP. The parameters are $\theta = 3$ and $k = 30$.

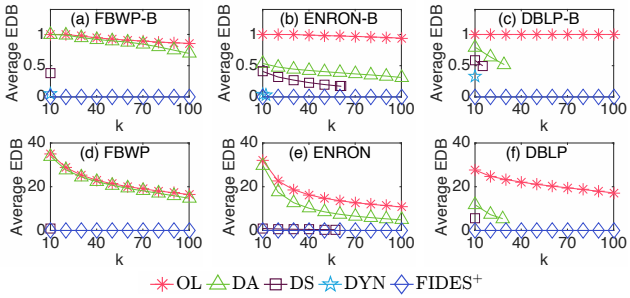


Figure 8: The average EDB of top-k detected temporal subgraphs.

We tackle this problem by extending DA, DS, DYN and FIDES$^+$ to produce multiple dense temporal subgraphs.

The extension on FIDES$^+$ is straightforward. Since FIDES$^+$ detects k candidate dense temporal subgraphs before it selects the best one, we extend FIDES$^+$ by simply returning all the k candidates.

Now we illustrate how to extend DA, DS and DYN. Each of DA, DS and DYN maintains the top-1 densest subgraph that may change when edges are updated. Every time the maintained subgraph changes, we get its set of vertices, denoted by S, and search the entire temporal graph to find the optimal time interval $(t_b, t_e], t_e - t_b \geq \theta$ that maximizes the EDB of temporal subgraph $(S, (t_b, t_e])$. Then, $(S, (t_b, t_e])$ is returned as a detected temporal subgraph. Since the maintained subgraph changes many times along the stream of edge updates, the extended DA, DS and DYN can find multiple dense temporal subgraphs.

However, as to be shown in Figures 8, 9 and 10, the extended DA, DS and DYN still cannot find as many dense temporal subgraphs as OL on some data sets. The reason is that the maintained top-1 densest subgraph is stably dominated by a small number of dense subgraphs, thus it does not change many times along the stream of edge updates.

We evaluate the performance of all methods in finding top-k ($k \leq 100$) dense temporal subgraphs ranked in descending order of EDB. OL, DA, DS and FIDES$^+$ are evaluated on both the original data sets and the converted data sets. However, since DYN cannot process the weighted edge updates in the original data sets, it is only evaluated on the converted data sets.

Figure 8 shows the average EDB of the top-k dense temporal subgraphs detected by each of the compared methods. OL achieves the best average EDB performance on all data sets, because it focuses on detecting DBSs that rapidly gain a large cohesiveness in a short time.

In Figures 8(a) and 8(d), DA achieves a similar average EDB to OL. This is due to the highly sparse networks of FBWP-B and FBWP, which force both DA and OL to find

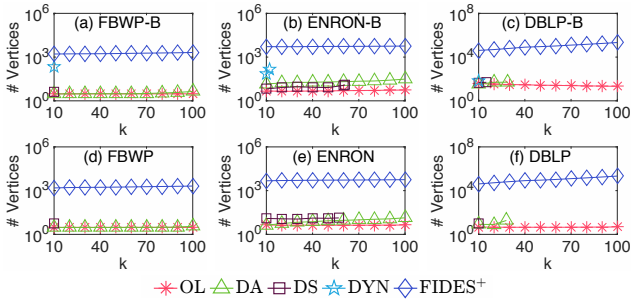


Figure 9: The average number of vertices in top-k detected temporal subgraphs.

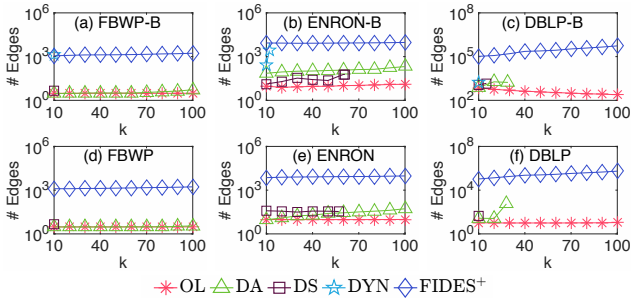


Figure 10: The average number of edges in top-k detected temporal subgraphs.

small dense temporal subgraphs with similar EDB. However, as shown in Figures 8(b), 8(c), 8(e) and 8(f), the difference between the average EDB of OL and DA becomes more significant on the denser networks, such as ENRON-B, DBLP-B, ENRON and DBLP.

As shown in Figure 8, most of the baseline methods do not achieve a good average EDB performance due to the fact that they are not designed to find DBSs.

For example, FIDES$^+$ detects connected subgraphs with large sum of edge weights in an accumulated graph, thus it naturally favours large subgraphs with low edge density. As shown in Figures 8, 9 and 10, FIDES$^+$ always detects large temporal subgraphs that contain thousands of vertices and edges, but have a very low average EDB.

DS and DYN maintain the densest subgraph with the largest average degree in an accumulated graph. However, since a subgraph can slowly accumulate a large density during an arbitrarily long time, a subgraph with large density is not necessarily a density bursting subgraph. As a result, both DS and DYN tend to detect temporal graphs that accumulate a large average degree for a long time. As shown in Figure 8, these subgraphs usually have a low EDB.

DA maintains a temporal subgraph that has the largest average degree in a time window with a fixed length. However, according to Tsourakakis *et al.* [32], such a dense subgraph with the largest average degree is typically a large graph with large diameter and small edge density. As a result, we can see in Figures 8, 9 and 10 that the temporal subgraphs detected by DA usually have a larger size but a lower average EDB than the DBSs detected by OL.

A closer look at Figures 9 and 10 shows that the DBSs found by OL are small in size. As illustrated in Section 4, every DBS is a dense subgraph that have a large cohesiveness in an accumulated graph. These dense subgraphs are usually small clique-like subgraphs that have a large edge density in sparse accumulated graphs [20, 21, 27].

As already shown in the case studies of Section 8.1, the small clique-like DBSs detected by OL accurately identify interesting real world patterns, which could be easily missed by methods that find large subgraphs with small EDB.

9. CONCLUSIONS

In this paper, we tackle the novel problem of finding top-k DBS from temporal graphs. We formulate the top-k DBS finding problem as a MIP problem and, as a baseline, solve it by $SlideWin$. By investigating the decomposition property of DBSs, we further design $TopkDBSOL$ to find the set of top-k indecomposable DBSs in an online manner. Extensive experiments show that $TopkDBSOL$ finds a comparably good solution as $SlideWin$, and improves the efficiency of $SlideWin$ by orders of magnitudes. As future work, we will extend $TopkDBSOL$ to detect DBSs from signed temporal networks.

APPENDIX

A.1 The Proof of Theorem 1

Proof. We only need to prove that the top-1 (i.e., $k = 1$) DBS finding (TDF) problem is NP-hard. Consider an arbitrary unweighted and undirected graph G, whose affinity matrix is A. The entries of A are either 0 or 1. We create an instance of the TDF problem by constructing a temporal graph $\mathcal{G}(t_0, t_c) = \langle G(t_0), \ldots, G(t_c)\rangle$, such that $t_c - t_0 = \theta = 1$, $A(t_c) = A$ and the affinity matrices of all the snapshots in $\mathcal{G}(t_0, t_{c-1})$ are matrices of all 0's.

Since $t_c - t_0 = \theta = 1$, it follows Definition 1 that $T^* = (t_0, t_c]$ is the only optimal time interval for any DBS in $\mathcal{G}(t_0, t_c)$. Thus, the following optimization problem

$$\operatorname*{argmax}_{\mathbf{x}} g(\mathbf{x}, T^*), \text{ s.t. } \mathbf{x} \in \triangle^n$$

can be reduced to the TDF problem. Since $T^* = (t_0, t_c]$ and $t_c - t_0 = 1$, we have $g(\mathbf{x}, T^*) = \mathbf{x}^\top A\mathbf{x}$. The above optimization problem is rewritten as

$$\operatorname*{argmax}_{\mathbf{x}} \mathbf{x}^\top A\mathbf{x}, \text{ s.t. } \mathbf{x} \in \triangle^n,$$

which is a NP-hard problem [26]. □

A.2 The Proof of Theorem 2

Proof. We prove by constructing a temporal graph $\mathcal{G}(t_0, t_c)$ such that $t_c - t_0 \geq \theta$, every snapshot in $\mathcal{G}(t_1, t_c)$ has exactly the same non-empty affinity matrix, and $\forall i \in \{1, \ldots, c-1\}$, $t_i - t_{i-1} = t_{i+1} - t_i$.

According to Definition 1, for any local maximum point $\mathbf{x}^* \in \triangle^n$ of $q_{\mathbf{x}}(t_1)$, $(\mathbf{x}^*, (t_0, t_c])$ is a DBS in $\mathcal{G}(t_0, t_c)$. □

A.3 The Proof of Theorem 3

Proof. Consider the slope representation of $(\mathbf{x}^*, T^*)$ in Figure 11(a), where z_1 and z_2 are contained in $\mathcal{P}_{\mathbf{x}^*}$.

First, we prove by contradiction that the triangle regions P and Q do not contain any point in $\mathcal{P}_{\mathbf{x}^*}$. Assume P contains $z_3 \in \mathcal{P}_{\mathbf{x}^*}$. Then, the slope between z_1 and z_3 is larger than the slope between z_1 and z_2. Thus, T^* is not the global maximum ponit of the MDS problem. This contradicts with the condition that $(\mathbf{x}^*, T^*)$ is a DBS. Therefore, P does not contain any point z_3 in $\mathcal{P}_{\mathbf{x}^*}$. Similarly, Q does not contain any point z_4 in $\mathcal{P}_{\mathbf{x}^*}$.

Second, we prove that $(\mathbf{x}^*, T^*)$ and its components have the same burstiness. Since $(\mathbf{x}^*, T^*)$ is decomposable, we know $\mathcal{T}(t_{b^*}, t_{e^*}) \neq \emptyset$ and $(\mathbf{x}^*, T^*)$ is decomposable at any time $t_h \in \mathcal{T}(t_{b^*}, t_{e^*})$. Recall that any point z_h in $\mathcal{P}_{\mathbf{x}^*}$

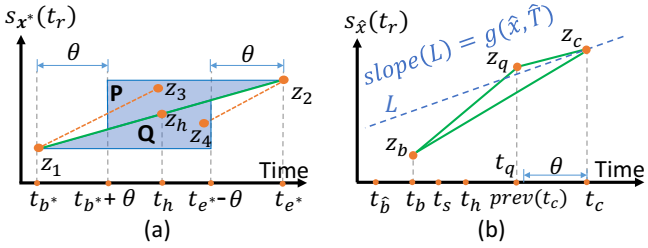


Figure 11: (a) shows the slope representation of a decomposable DBS $(\mathbf{x}^*, T^*)$, where $T^* = (t_{b^*}, t_{e^*}]$ and $\mathcal{T}(t_{b^*}, t_{e^*}) \neq \emptyset$. z_1 and z_2 are contained in $\mathcal{P}_{\mathbf{x}^*}$. P and Q are the two blue triangle regions, respectively. (b) shows the slope representation to prove Theorem 5. L is an auxiliary line that crosses $z_c \in \mathcal{P}_{\hat{\mathbf{x}}}$. The slope of L is $g(\hat{x}, \hat{T})$, and $t_q = prev(t_c)$.

cannot be contained by P or Q, since $\forall t_h \in \mathcal{T}(t_{b^*}, t_{e^*})$, $t_{b^*} + \theta \leq t_h \leq t_{e^*} - \theta$, z_h must reside on the segment between z_1 and z_2. By the slope representation of burstiness, $g(\mathbf{x}^*, (t_{b^*}, t_h]) = g(\mathbf{x}^*, (t_h, t_{e^*}]) = g(\mathbf{x}^*, T^*)$. □

A.4 The Proof of Theorem 4

Proof. Since $(\mathbf{x}^*, T^*)$ is an indecomposable DBS, $\mathbf{x}^*$ is a local maximum point of $\mathbf{x}^\top A(t_{b^*+1}, t_{e^*})\mathbf{x}$. Since $v_i \in (\mathbf{x}^*, T^*)$, we have $v_i \in V_{\mathbf{x}^*}$.

Since $v_i \in V_{\mathbf{x}^*}$ and $\mathbf{x}^*$ is a local maximum point of $\mathbf{x}^\top A(t_{b^*+1}, t_{e^*})\mathbf{x}$, the following equation holds [22].

$$(\mathbf{x}^*)^\top A(t_{b^*+1}, t_{e^*})\mathbf{x}^* = \sum_j \mathbf{x}_j^* A_{ij}(t_{b^*+1}, t_{e^*}).$$

Therefore, we have

$$g(\mathbf{x}^*, T^*) = \frac{(\mathbf{x}^*)^\top A(t_{b^*+1}, t_{e^*})\mathbf{x}^*}{t_{e^*} - t_{b^*}} = \frac{\sum_j \mathbf{x}_j^* A_{ij}(t_{b^*+1}, t_{e^*})}{t_{e^*} - t_{b^*}}.$$

Since $\mathbf{x}^* \in \triangle^n$, we can derive

$$\frac{\sum_j \mathbf{x}_j^* A_{ij}(t_{b^*+1}, t_{e^*})}{t_{e^*} - t_{b^*}} \leq \frac{\max_j A_{ij}(t_{b^*+1}, t_{e^*})}{t_{e^*} - t_{b^*}}.$$

Recall that $\alpha(v_i, T^*) = \frac{\max_j A_{ij}(t_{b^*+1}, t_{e^*})}{t_{e^*} - t_{b^*}}$, we have $g(\mathbf{x}^*, T^*) \leq \alpha(v_i, T^*)$. □

A.5 The Proof of Theorem 5

Proof. Please refer to the slope representation in Figure 11(b) for the following proof.

(Direction only-if) Suppose $\exists t_h \in (t_s, prev(t_c)]$ such that $g(\hat{\mathbf{x}}, (t_h, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$. Since $t_{\hat{b}} < t_s < t_h \leq prev(t_c)$, $\hat{T}$ is not a global maximum point of $g(\hat{\mathbf{x}}, T)$ in $\mathcal{G}(t_{\hat{b}}, t_c)$. Since $t_c - t_h \geq \theta$, $(\hat{\mathbf{x}}, \hat{T})$ is not an indecomposable DBS in $\mathcal{G}(t_{\hat{b}}, t_c)$, which means $(\hat{\mathbf{x}}, \hat{T}) \notin \mathcal{D}(t_c)$. As a result, if $(\hat{\mathbf{x}}, \hat{T}) \in \mathcal{D}(t_c)$, then $\nexists t_h \in (t_s, prev(t_c)]$ such that $g(\hat{\mathbf{x}}, (t_h, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$.

(Direction if) First, we prove $\nexists t_b \in (t_{\hat{b}}, t_s]$ such that $g(\hat{\mathbf{x}}, (t_b, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$ by contradiction.

Suppose $\exists t_b \in (t_{\hat{b}}, t_s]$ such that $g(\hat{\mathbf{x}}, (t_b, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$. Then, as shown in Figure 11(b), $z_b \in \mathcal{P}_{\hat{\mathbf{x}}}$ must be under L. Since $\nexists t_h \in (t_s, prev(t_c)]$ such that $g(\hat{\mathbf{x}}, (t_h, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$, hence for $t_q = prev(t_c)$ in Figure 11(b), $z_q \in \mathcal{P}_{\hat{\mathbf{x}}}$ must be above or on L. This means that the slope of the segment (z_b, z_q) is larger than the slope of L, thus $g(\hat{\mathbf{x}}, (t_b, t_q]) > g(\hat{\mathbf{x}}, \hat{T})$. Since $t_{\hat{b}} < t_b$, $t_q = prev(t_c) \leq t_{c-1}$ and $t_q - t_b \geq t_q - t_s \geq \theta$, it follows $g(\hat{\mathbf{x}}, (t_b, t_q]) > g(\hat{\mathbf{x}}, \hat{T})$ that $\hat{T}$ is not a global maximum point of $g(\hat{\mathbf{x}}, T)$ in $\mathcal{G}(t_{\hat{b}}, t_{c-1})$, which means $(\hat{\mathbf{x}}, \hat{T}) \notin \mathcal{D}(t_{c-1})$. This contradicts with the condition $(\hat{\mathbf{x}}, \hat{T}) \in \mathcal{D}(t_{c-1})$. Therefore, $\nexists t_b \in (t_{\hat{b}}, t_s]$ such that $g(\hat{\mathbf{x}}, (t_b, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$.

Second, we prove the direction if of the theorem.

Since $\nexists t_b \in (t_{\hat{b}}, t_s]$ such that $g(\hat{\mathbf{x}}, (t_b, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$ and $\nexists t_h \in (t_s, prev(t_c)]$ such that $g(\hat{\mathbf{x}}, (t_h, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$, it is obvious that $\nexists t_b \in (t_{\hat{b}}, prev(t_c)]$ such that $g(\hat{\mathbf{x}}, (t_b, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$. Recall that $(\hat{\mathbf{x}}, \hat{T}) \in \mathcal{D}(t_{c-1})$, since $\nexists t_b \in (t_{\hat{b}}, prev(t_c)]$ such that $g(\hat{\mathbf{x}}, (t_b, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$, $\hat{T}$ is also a global maximum point of $g(\hat{\mathbf{x}}, T)$ in $\mathcal{G}(t_{\hat{b}}, t_c)$. Therefore, $(\hat{\mathbf{x}}, \hat{T})$ is an indecomposable DBS in $\mathcal{G}(t_{\hat{b}}, t_c)$, which means $(\hat{\mathbf{x}}, \hat{T}) \in \mathcal{D}(t_c)$. As a result, if $\nexists t_h \in (t_s, prev(t_c)]$ such that $g(\hat{\mathbf{x}}, (t_h, t_c]) > g(\hat{\mathbf{x}}, \hat{T})$ then $(\hat{\mathbf{x}}, \hat{T}) \in \mathcal{D}(t_c)$. □

A.6 The Proof of Theorem 6

Proof. Denote by $(\tilde{\mathbf{x}}, \tilde{T})$ the real top-1 indecomposable DBS in $\mathcal{G}(t_0, t_c)$, where $\tilde{T} = (t_{\tilde{b}}, t_{\tilde{e}}]$.

First, we prove *TopkDBSOL* will produce a 2-approximation of $g(\tilde{\mathbf{x}}, \tilde{T})$ when $t_c = t_{\tilde{e}}$.

By Theorem 4, we have

$$\forall v_i \in (\tilde{\mathbf{x}}, \tilde{T}), g(\tilde{\mathbf{x}}, \tilde{T}) \leq \alpha(v_i, \tilde{T}). \tag{4}$$

Denote by $\epsilon_k(t_{\tilde{e}-1})$ the k-th largest burstiness of all indecomposable DBSs in $\mathcal{D}(t_{\tilde{e}-1})$. Since $(\tilde{\mathbf{x}}, \tilde{T})$ is the top-1 indecomposable DBS, we have $\epsilon_k(t_{\tilde{e}-1}) \leq g(\tilde{\mathbf{x}}, \tilde{T}) \leq \alpha(v_i, \tilde{T})$.

For $t_c = t_{\tilde{e}}$, since $\alpha(v_i, \tilde{T}) \geq \epsilon_k(t_{\tilde{e}-1})$, step 7 of *FindNDBSC* will call *FindDBS* with an initialization $(\mathbf{o}(v_i, \tilde{T}), \tilde{T})$ to find a NDBSC $(\hat{\mathbf{x}}, \hat{T}) \in \mathcal{D}(t_{\tilde{e}})$.

Since *FindDBS* monotonically increases the density of a temporal subgraph, we have

$$g(\mathbf{o}(v_i, \tilde{T}), \tilde{T}) \leq g(\hat{\mathbf{x}}, \hat{T}). \tag{5}$$

By definition of $(\mathbf{o}(v_i, \tilde{T}), \tilde{T})$, we expand $g(\mathbf{o}(v_i, \tilde{T}), \tilde{T})$ as

$$\frac{0.5 * 0.5 * \max_j A_{ij}(t_{\tilde{b}+1}, t_{\tilde{e}}) + 0.5 * 0.5 * \max_j A_{ij}(t_{\tilde{b}+1}, t_{\tilde{e}})}{t_{\tilde{e}} - t_{\tilde{b}}},$$

which is equal to $0.5 * \alpha(v_i, \tilde{T})$.

Since $g(\mathbf{o}(v_i, \tilde{T}), \tilde{T}) = 0.5 * \alpha(v_i, \tilde{T})$, we can derive from Equation 5 that

$$0.5 * \alpha(v_i, \tilde{T}) \leq g(\hat{\mathbf{x}}, \hat{T}). \tag{6}$$

Since $(\tilde{\mathbf{x}}, \tilde{T})$ is the real top-1 indecomposable DBS in $\mathcal{G}(t_0, t_c)$, we have

$$g(\hat{\mathbf{x}}, \hat{T}) \leq g(\tilde{\mathbf{x}}, \tilde{T}). \tag{7}$$

Now, we can derive from Equations 4, Equation 6 and Equation 7 that $0.5 * \alpha(v_i, \tilde{T}) \leq g(\hat{\mathbf{x}}, \hat{T}) \leq g(\tilde{\mathbf{x}}, \tilde{T}) \leq \alpha(v_i, \tilde{T})$, which means $g(\hat{\mathbf{x}}, \hat{T})$ is a 2-approximation of $g(\tilde{\mathbf{x}}, \tilde{T})$.

Second, we prove *TopkDBSOL* will produce a 2-approximation of $g(\tilde{\mathbf{x}}, \tilde{T})$ when $t_c > t_{\tilde{e}}$.

For $t_c > t_{\tilde{e}}$, there are two cases discussed as follows.

Case 1: if $(\hat{\mathbf{x}}, \hat{T}) \in \mathcal{D}(t_c)$, then $g(\hat{\mathbf{x}}, \hat{T})$ is already a 2-approximation of $g(\tilde{\mathbf{x}}, \tilde{T})$.

Case 2: if $(\hat{\mathbf{x}}, \hat{T}) \notin \mathcal{D}(t_c)$, *UpdateODBSC* will update $(\hat{\mathbf{x}}, \hat{T})$ to increase its burstiness. Obviously, the burstiness of the updated temporal subgraph is still a 2-approximation of $g(\tilde{\mathbf{x}}, \tilde{T})$. □

A. REFERENCES

[1] J. Abello, M. Resende, and S. Sudarsky. Massive quasi-clique detection. *Theoretical Informatics*, pages 598–612, 2002.

[2] C. C. Aggarwal, Y. Li, P. S. Yu, and R. Jin. On dense pattern mining in graph streams. *PVLDB*, 3(1-2):975–984, 2010.

[3] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6):574–585, 2012.

[4] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. Tsourakakis. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *ACM Symposium on Theory of Computing*, pages 173–182, 2015.

[5] B. Boden, S. Günnemann, H. Hoffmann, and T. Seidl. Mining coherent subgraphs in multi-layer graphs with edge labels. In *KDD*, pages 1258–1266, 2012.

[6] P. Bogdanov, M. Mongiovì, and A. K. Singh. Mining heavy subgraphs in time-evolving networks. In *ICDM*, pages 81–90, 2011.

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

[8] S. R. Bulò and I. M. Bomze. Infection and immunization: a new class of evolutionary game dynamics. *GEB*, 71(1):193–211, 2011.

[9] K. Chao. Maximum-density segment. In *Encyclopedia of Algorithms*, pages 1–99. 2008.

[10] L. Chu, S. Wang, S. Liu, Q. Huang, and J. Pei. ALID: scalable dominant cluster detection. *PVLDB*, 8(8):826–837, 2015.

[11] L. Chu, Z. Wang, J. Pei, J. Wang, Z. Zhao, and E. Chen. Finding gangs in war from signed networks. In *KDD*, pages 1505–1514, 2016.

[12] K. M. Chung and H. I. Lu. An optimal algorithm for the maximum-density segment problem. *SIAM Journal on Computing*, 34(2):373–387, 2005.

[13] S. Curtis and S. C. Mu. Calculating a linear-time solution to the densest-segment problem. *Journal of Functional Programming*, 25, 2015.

[14] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *WWW*, pages 300–310, 2015.

[15] M. H. Goldwasser, M. Y. Kao, and H. I. Lu. Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. *JCSS*, 70(2):128–144, 2005.

[16] X. Huang. An algorithm for identifying regions of a dna sequence that satisfy a content requirement. *Bioinformatics*, 10(3):219–225, 1994.

[17] KAN. DBLP-Citation-Network v10 data set. `https://static.aminer.org/lab-datasets/citation/dblp.v10.zip`.

[18] S. K. Kim. Linear-time algorithm for finding a maximum-density segment of a sequence. *IPL*, 86(6):339–342, 2003.

[19] Y.-L. Lin, T. Jiang, and K.-M. Chao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *JCSS*, 65(3):570–586, 2002.

[20] H. Liu, L. J. Latecki, and S. Yan. Fast detection of dense subgraphs with iterative shrinking and expansion. *TPAMI*, 35(9):2131–2142, 2013.

[21] H. Liu and S. Yan. Common visual pattern discovery via spatially coherent correspondences. In *CVPR*, pages 1609–1616, 2010.

[22] H. Liu and S. Yan. Robust graph mode seeking by graph shift. In *ICML*, pages 671–678, 2010.

[23] X. Liu, T. Ge, and Y. Wu. Finding densest lasting subgraphs in dynamic graphs: A stochastic approach. In *ICDE*, 2019.

[24] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai. Fast computation of dense temporal subgraphs. In *ICDE*, pages 361–372, 2017.

[25] S. Ma, R. Hu, L. Wang, X. Lin, and J.-P. Huai. An efficient approach to finding dense temporal subgraphs. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[26] T. S. Motzkin and E. G. Straus. Maxima for graphs and a new proof of a theorem of turán. *Canad. J. Math*, 17(4):533–540, 1965.

[27] M. Pavan and M. Pelillo. Dominant sets and pairwise clustering. *TPAMI*, 29(1):167–172, 2007.

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

[29] K. Shin, B. Hooi, J. Kim, and C. Faloutsos. D-cube: Dense-block detection in terabyte-scale tensors. In *WSDM*, pages 681–689, 2017.

[30] K. Shin, B. Hooi, J. Kim, and C. Faloutsos. Densealert: Incremental dense-subtensor detection in tensor streams. In *KDD*, pages 1057–1066, 2017.

[31] TAXI-1 and TAXI-2. Taxi trip data of the city of chicago. `https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew`.

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

[33] UKL. Konect. `http://konect.uni-koblenz.de/`, 2018.

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

[35] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. Lui. Diversified temporal subgraph pattern mining. In *KDD*, pages 1965–1974, 2016.