# Mining Stable Communities in Temporal Networks by Density-based Clustering

Hongchao Qin, Rong-Hua Li, Guoren Wang, Xin Huang, Ye Yuan, and Jeffrey Xu Yu

**Abstract**—Community detection is a fundamental task in graph data mining. Most existing studies in contact networks, collaboration networks, and social networks do not utilize the temporal information associated with edges for community detection. In this paper, we study a problem of finding stable communities in a temporal network, where each edge is associated with a timestamp. Our goal is to identify the communities in a temporal network that are stable over time. To efficiently find the stable communities, we develop a new community detection algorithm based on the density-based graph clustering framework. We also propose several carefully-designed pruning techniques to significantly speed up the proposed algorithm. We conduct extensive experiments on four real-life temporal networks to evaluate our algorithm. The results demonstrate the effectiveness and efficiency of the proposed algorithm.

**Index Terms**—Community Detection, Temporal Networks, Stable Community, Structural Graph Clustering, Density-based Clustering.

## 1 Introduction

Real-life networks such as social networks, biological networks, and communication networks exhibit the property of community structure [1]. Discovering communities in a network is a fundamental graph data mining task which has attracted much attention in recent years due to a large number of applications [2], [3], [4], [5], [6], [7], [8], [9]. In applications such as analysis of contact networks and scientific collaboration networks, the links are typically associated with temporal information. For example, in the contact network [10], [11], each link $(u, v, t)$ denotes a contact between two individuals $u$ and $v$ at time $t$. In the collaboration network, each edge represents the relationship that two authors, $u$ and $v$, coauthored a paper at time $t$. Most existing community detection algorithms such as [1], [2], [3], [4], [5], [6], [7] do not consider the temporal information of the links, thus they may fail to discover some important temporal patterns such as the pattern of stable community structure in the temporal network.

In this paper, we study the problem of detecting stable community structures in the temporal network, where each edge is associated with a timestamp. Our goal is to identify a densely-connected community structure that is stable over time. Such a stable community may be very useful for many network analysis applications. For example, consider an e-mail communication network between staffs in a university. A stable community may reveal a group of staffs that come from the same department, because the staffs in the same department are more likely to stably communicate with one another. In a scientific collaboration network, a stable community may reflect a group of researchers that maintain long-term collaborations. Such a group of researchers may be useful for the application of finding a stable team of experts to complete a special research project.

The community detection problem in static networks has been extensively studied in the literature, such as optimization-based algorithms [1], [4], [5], [12], dense subgraph mining (clique, quasi-clique) [13], [14] and some clique-relaxed model [6], [7], [15]. Many of the existing community detection algorithms [16] for those community models above focus mainly on traditional graphs, thus they also cannot be directly used for detecting stable communities in temporal graphs.

To find the stable communities, we develop a new community detection algorithm for temporal graphs based on the density-based graph clustering framework, since the density-based graph clustering framework has shown to be very efficient and effective for community detection applications [3], [17], [18], comparing with the other community models. Note that the traditional density-based graph clustering algorithms [3], [17], [18], [19] cannot be directly applied to detect stable communities, because all of them ignore the temporal information of the edges. To solve our problem, we need to integrate both the temporal information and the graph structure into the clustering procedure. The main contributions of our work are summarized as follows.

New concepts and problems. We first introduce a new concept, called $\epsilon$-stable similarity, to measure the *stability* of structural similarity between two nodes in the temporal network. Then, we propose a novel concept, called $(\mu, \tau, \epsilon)$-stable core, to characterize the stable core nodes of the

- *H.Qin is with the Department of Computer Science, Northeastern University, Shenyang, Liaoning 110004, China.*
  *E-mail: qhc.neu@gmail.com*
- *R.Li and G.Wang are with the Department of Computer Science, Beijing Institute of Technology, Beijing, Beijing 100081, China.*
  *E-mail: lironghuascut@gmail.com; wanggrbit@126.com*
- *X.Huang is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, China.*
  *E-mail: xinhuang@comp.hkbu.edu.hk*
- *Y.Yuan is with the Department of Computer Science, Northeastern University, Shenyang, Liaoning 110004, China.*
  *E-mail: yuanye@mail.neu.edu.cn*
- *J. Yu is with the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, China.*
  *E-mail: yu@se.cuhk.edu.hk*

*Manuscript received 2018*

clusters. A node $u$ is a $(\mu, \tau, \epsilon)$-stable core if $u$ has no less than $\mu$ neighbors that are simultaneously similar to $u$ in at least $\tau$ snapshots of the temporal graph. Based on these definitions, we formulate the density-based clustering problem to detect stable communities in temporal graphs. To the best of our knowledge, this is the first work to study the density-based clustering problem on temporal graphs.

New algorithms. The main technical challenge in our problem is to compute the $(\mu, \tau, \epsilon)$-stable cores. By our definition, a basic approach to determine a $(\mu, \tau, \epsilon)$-stable core needs to perform frequent pattern mining over all the snapshots of the temporal graph, thus it is very costly for large temporal graphs (it needs at least $O(C^{\tau}_{\mathcal{T}})$, $\mathcal{T}$ is the number of timestamps). To improve the efficiency of the basic algorithm, we propose two relaxed definitions of the $(\mu, \tau, \epsilon)$-stable core, called weak and strong core respectively. Instead of directly computing stable cores, our algorithm first makes use of the weak and strong cores to significantly prune the unpromising nodes, and then identifies the stable cores from the remaining nodes. Unlike the $(\mu, \tau, \epsilon)$-stable core, both the weak and strong core can be computed very quickly. Specifically, we develop two efficient algorithms with several carefully-designed pruning techniques to compute the weak and strong cores respectively. Those pruning techniques can reduce the graph into a much smaller size and they can get exact stable cores quickly.

Experimental evaluation. We conduct comprehensive experiments using four real-life temporal graphs to evaluate the proposed algorithm. The results indicate that our algorithm significantly outperforms the baselines in terms of the clustering quality. The results also demonstrate the high efficiency of the proposed algorithm. For example, our algorithm only takes 100 seconds to find stable communities in a large-scale temporal graph with 1,729,816 nodes and 12,007,380 edges under most parameter settings. In addition, we also perform a case study on the DBLP dataset. The results demonstrate that our approach can identify many meaningful and interesting stable communities that cannot be found by the other methods.

**Organization.** Section 2 introduces the model of stable communities and formulates our problem. The core reduction techniques for mining the stable communities are proposed in Section 3. Experimental studies are presented in Section 4. We review the related work in Section 5, and conclude this work in Section 6.

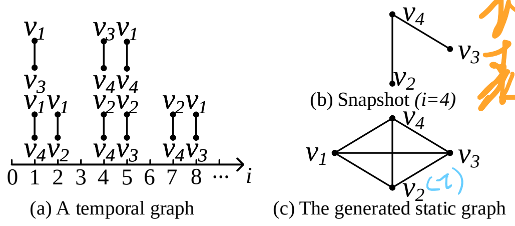
## 2 PRELIMINARIES

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected temporal graph, where $\mathcal{V}$ and $\mathcal{E}$ denote the set of nodes and the set of temporal edges respectively. Let $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$ be the number of nodes and temporal edges respectively. Each temporal edge $e \in \mathcal{E}$ is a triplet $(u, v, t)$, where $u, v$ are nodes in $\mathcal{V}$, and $t$ is the interaction time between $u$ and $v$. We assume that $t$ is an integer, because the timestamp is an integer in practice. For a temporal graph $\mathcal{G}$, the *de-temporal* graph of $\mathcal{G}$ is referred to as $G = (V, E)$ by ignoring all the timestamps associated with the temporal edges. Clearly, we have $V = \mathcal{V}$.

By sorting the temporal edges in a chronological order, the temporal graph can be represented as a link stream

TABLE 1
Main symbols

| Symbols | Definitions |
|---|---|
| $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ | the undirected temporal graph |
| $\mathcal{E}$ | temporal edges, set of triplet $(u, v, t)$ |
| $G = (V, E)$ | de-temporal graph of $\mathcal{G}$ |
| $N(u)$ (or $N_G(u)$) | neighborhood of node $u$ in $G$ |
| $\mathcal{G}_i = (\mathcal{V}, \mathcal{E}_i)$ | snapshot with $\mathcal{E}_i = \{(u, v, t) \mid t \in (t_{i-1}, t_i]\}$ |
| $G_i$ | de-temporal graph of $\mathcal{G}_i$ |
| $N_i(u)$ | neighborhood of node $u$ in $G_i$ |
| $\sigma_i(u, v)$ | structural similarity between nodes $u$ and $v$ in $G_i$ |
| StableCore | $(\mu, \tau, \epsilon)$-stable core (Definition 1) |
| $S_\epsilon(u, v)$ | $\epsilon$-stable similarity (Definition 2) |
| $(\tau, \epsilon)$-connected edge | edge $(u, v)$ satisfying $S_\epsilon(u, v) \geq \tau$ |
| $N_{(\tau,\epsilon)}(u)$ | node set of $(\tau, \epsilon)$-connected neighbors of $u$ |
| WeakCore | weak core model (Definition 6) |
| StrongCore | strong core model (Definition 7) |



不同的
表现形式

Fig. 1. Illustration of the key definitions of temporal graph

[20]. The widely-used approach to extract interesting patterns from a temporal graph relies on series of snapshots [20], [21], [22]. Considering an arithmetic time sequence $\{t_0, t_1, t_2 ... t_{\mathcal{T}}\}$ satisfying $t_i - t_{i-1}$ is a constant for each integer $i$, let $\mathcal{E}_i$ be a set of edges that are extracted from $\mathcal{E}$ in the time interval $(t_{i-1}, t_i]$. The $i$-th snapshot of $\mathcal{G}$ is a temporal subgraph $\mathcal{G}_i = (\mathcal{V}, \mathcal{E}_i)$. Let $\mathcal{T}$ be the number of snapshots of $\mathcal{G}$, and we have $\mathcal{T} \leq m$. Let $G_i = (V, E_i)$ be the *de-temporal* graph of the $i$-th snapshot $\mathcal{G}_i = (\mathcal{V}, \mathcal{E}_i)$. 层数[illegible] In the experiments, we set $t_i - t_{i-1}$ to a default value of 1 month/year which means that every snapshot contains all the temporal edges in a one month/year length sliding window. Fig. 1(a) illustrates a temporal graph with 9 temporal edges and 8 timestamps. $\tau$ While $i = 4$, snapshot $\mathcal{G}_4$ contains two edges $(v_2, v_4)$ and $(v_3, v_4)$, which is shown at Fig. 1(b). Fig. 1(c) shows the *de-temporal* graph of $\mathcal{G}$ by generating a static graph with ignoring all timestamps. For convenient, Table 1 lists the main symbols used in this paper and their definitions.

Based on the snapshots and the concepts in the density-based graph clustering framework (SCAN) [3], we introduce several useful definitions for the stable cluster mining problem on temporal graphs. At first, we can use structural similarity [3] to measure similarity of two nodes in the graph. It can capture the similarity of two nodes by computing the number of common neighbor between them. Besides, other similar definitions, such as Jaccard similarity, can also measure the similarity of them. However, those methods need to compute the set-intersection of neighbors of the considering two nodes. So, different definitions for the similarity of nodes make little difference to the acceleration ratio of the proposed optimized algorithms below. Without loss of generality, we use the structural similarity following the standard SCAN algorithm to measure the similarity of two nodes. Therefore, the structural similarity between two

stable
指标

len_v_u > eps · sqrt 时

len_v_u

nodes of an edge $(u, v)$ in $G_i$ is defined as

eps 计算v的邻居 + v自身

结构相似度SS $$\sigma_i(u,v) \triangleq \frac{|N_i[u] \cap N_i[v]|}{\sqrt{|N_i[u]| \times |N_i[v]|}} \qquad (1)$$

where $N_i(u) = \{v|(u,v) \in E_i\}$ is neighborhood of $u$ in $G_i$ and $N_i[u] = N_i(u) \cup \{u\}$. We define $\sigma_i(u,v) = 0$ if $(u,v) \notin G_i$. The set of $\epsilon$**-neighbors** of $v$ in $G_i$ is defined as $N_i^\epsilon(v) \triangleq \{u \in N_i(v)|\sigma_i(u,v) \geq \epsilon\}$. 表示相似度大于ε的邻居

By following the SCAN framework, we propose a new definition called $(\mu, \tau, \epsilon)$-stable core to characterize a stable core node of the cluster in a temporal network.

***Definition 1** **((μ, τ, ϵ)-stable core).*** A node $u \in V$ is called a $(\mu, \tau, \epsilon)$-stable core if there exists a set of neighbors $\tilde{N}(u) \subseteq N(u)$ of $u$ that satisfies the following conditions: (i) $|\tilde{N}(u)| \geq \mu$; (ii) there are at least $\tau$ snapshots $\{\mathcal{G}_{j_1}, \cdots, \mathcal{G}_{j_l}\}$ $(l \geq \tau)$ containing the star-shaped structure formed by $\tilde{N}(u) \cup \{u\}$; and (iii) in each snapshot $\mathcal{G}_{j_k}$ $(1 \leq k \leq l)$, $\sigma_{j_k}(u,v) \geq \epsilon$ for any $v \in \tilde{N}(u)$. ?

一组u的邻居,邻居数≥μ 所有图中的所有邻居结构S都>ε 图数≥τ

To find the stable core, we can consider some nodes which are stably similar to node $u$ in the temporal graphs. Based on Eq. (1), we propose a definition of stable similarity to measure whether two nodes are stably similar in the temporal graphs.

***Definition 2** **(ϵ-stable similarity).*** For nodes $u$ and $v$ in $\mathcal{G}$, the $\epsilon$-stable similarity $S_\epsilon(u,v)$ is defined by

结构S > ε的快照数

$$S_\epsilon(u,v) = \sum\nolimits_{i=1}^{\mathcal{T}} \mathcal{I}(\sigma_i(u,v) > \epsilon), \qquad (2)$$

where $\mathcal{I}$ is an indicator function which equals 1 if $\sigma_i(u,v) > \epsilon$, and 0 otherwise.

Intuitively, if two nodes are stably similar, they should be structurally similar in many snapshots. Based on this intuition, we define the $(\tau, \epsilon)$-connected edge as follows.

***Definition 3** **((τ, ϵ)-connected edge).*** For each edge $(u,v)$ in the de-temporal graph $G$, if $S_\epsilon(u,v) \geq \tau$, the edge $(u,v)$ is called a $(\tau, \epsilon)$-connected edge.

ε稳.S > τ 的连边

By Definition 3, the set of $(\tau, \epsilon)$**-connected neighbors** of $u$ in $G$ is defined as $N_{(\tau,\epsilon)}(u) \triangleq \{v \in N(u)|S_\epsilon(u,v) \geq \tau\}$. Clearly, a node $v \in N_{(\tau,\epsilon)}(u)$ is a *stably similar* neighbor of $u$. It is worth mentioning that the $(\tau, \epsilon)$-connected neighbor is defined on the de-temporal graph $G$, while the $\epsilon$-neighbor is defined on the de-temporal snapshot $G_i$.

满足的邻居

By Definition 1, a $(\mu, \tau, \epsilon)$-stable core $u$ has at least $\mu$ neighbors such that they are simultaneously similar to $u$ in at least $\tau$ snapshots, indicating that $u$ has no less than $\mu$ stably similar neighbors. If two nodes are reachable through a series of $(\mu, \tau, \epsilon)$-stable cores, they should also be similar to each other. Based on this, we define the structural reachability between two nodes as follows.

哪种

***Definition 4** **(structure-reachable).*** A node $v$ is *structure-reachable* from $u$ if there is a sequence of nodes $v_1, v_2, \cdots, v_l \in V$ $((l \geq 2))$ such that: (i) $v_1 = u$, $v_l = v$; (ii) $v_1, v_2, ..., v_{l-1}$ are $(\mu, \tau, \epsilon)$-stable cores; and (iii) $(v_i, v_{i+1})$ is a $(\tau, \epsilon)$-connected edge for each $1 \leq i \leq l-1$.

结构可达 稳.核网 不可能经过至少 l-2条路径, v只需一步就 能到達 向u的这条 路径上

Intuitively, all nodes in a stable cluster should be structure-reachable from each other. Based on this intuition, we define a stable cluster as follows.

***Definition 5** **((μ, τ, ϵ)-stable cluster).*** A set of nodes $C \subseteq V$ is called a stable cluster if the following conditions holds.

每条都需满足(τ, ε)-连边

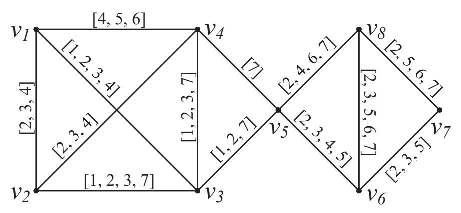


Fig. 2. Toy example of a temporal graph

C中包含自身 节点..所有可达的 稳核点..

*(i)* Maximality. For each $(\mu, \tau, \epsilon)$-stable core $u \in C$, all nodes that are structure-reachable from $u$ must be contained in $C$;

*(ii)* Connectivity. For any two nodes $v_1, v_2 \in C$, there is a $(\mu, \tau, \epsilon)$-stable core $u \in C$ such that both $v_1$ and $v_2$ are structure-reachable from $u$. (μ>2成立)

任意两节点 都能被同一 节点Reach

By Definition 5, we can obtain a $(\mu, \tau, \epsilon)$-stable cluster $C$ by merging all the nodes that are structure-reachable from the $(\mu, \tau, \epsilon)$-stable cores in $C$.

宽松的

Note that, Definition 5 is a relax model to characterize the stable communities in a temporal network. *(i)* It relaxes the exist time of some nodes in the stable cluster. Considering a (3, 3, 0.6)-stable cores $c_1$ in a temporal collaboration network. In an extreme case, suppose nodes $v_1$, $v_2$ and $v_3$ are simultaneously similar to $c_1$ in snapshots $t_1$, $t_2$, and $t_3$ and nodes $v_4$, $v_5$ and $v_6$ are simultaneously similar to $c_1$ in snapshots $t_4$, $t_5$, and $t_6$. This may be the case that professor $c_1$ moves to another place and has a new group of coauthors. According to Definition 5, all nodes will be assigned to the same stable cluster. However, $(v_1, v_2, v_3)$ and $(v_4, v_5, v_6)$ may not be in the same stable community. Otherwise, if we change the definition to distinguish those nodes which are clustered in different time, we will perform several more frequent pattern minings (each one need $O(C_{\mathcal{T}}^{\tau})$ time) and get much more overlap clusters. Those will result in much additional computation time and new problem of choosing the overlap clusters. So, we choose the current relaxed definition, results of four evaluation metrics and case studies in Section 4 confirm the effectiveness of our model. *(ii)* It relaxes the order of the exist timestamps of stable core. If we consider order of the snapshots in Definition 1, condition 2 can be changed into that their are at least $\tau$ continuous snapshots containing the star-shaped structure. Obviously, one core stratifies condition 1,3 and exists at {1, 2, 4, 5, 6} time will not be a stable core with $\tau = 5$ because the exist time is not continuous. However, we choose a relaxed version for the "stable" property. Following our definition, if one core exists at {1, 2, 4, 5, 6} time, it can be treated to be stable. Clearly, our current Definition 1 is more reasonable.

σ>ε

教授 换岗

第2个条 数据连续 变为连续

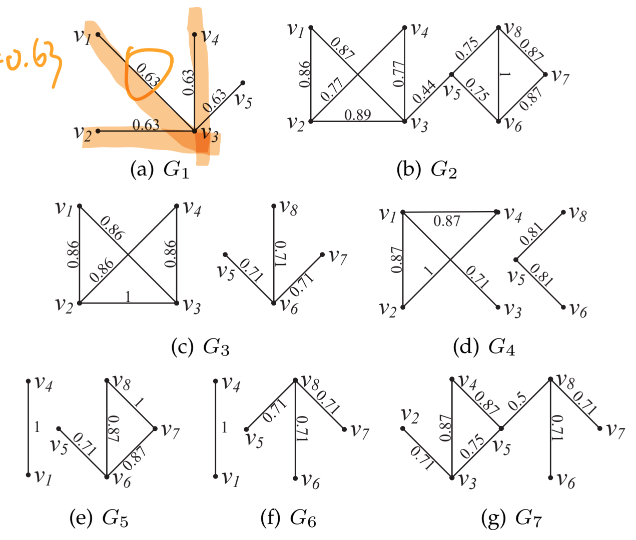
ε宽松 版不同

**Problem formulation.** Given a temporal graph $\mathcal{G}$ and parameters $\mu, \tau$ and $\epsilon$, the stable community mining problem is to identify all the $(\mu, \tau, \epsilon)$-stable clusters in $\mathcal{G}$.

Below, we make use of an example to illustrate the above key definitions.

**Example 1.** Fig. 2 shows a temporal graph, it can be represented by an attributed graph in which each edge has a vector of timestamps. For example, edge $(v_1, v_2)$ has a vector of $[2, 3, 4]$, which means that this edge exists at snapshots of $\mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4$. Also, edge $(v_1, v_2)$ will be in the *de-temporal*

$\epsilon = \frac{2}{\sqrt{5\times 2}} = 0.63$



(a) $G_1$ (b) $G_2$ (c) $G_3$ (d) $G_4$ (e) $G_5$ (f) $G_6$ (g) $G_7$

Fig. 3. De-temopral graph in each timestamp

graph $G_2, G_3, G_4$. If we take all the edges which contain a timestamp $t$ into consideration, the *de-temporal* graph $G_t$ of $t$-th snapshot will be formed.

Fig. 3(a) shows the *de-temporal* graph $G_1$ of snapshot $\mathcal{G}_1$. The number on each edge is the *structural similarity* between two nodes of the edge. By definition, $N_1[v_3] = \{v_1, v_2, v_4, v_5, v_3\}$, $N_1[v_1] = \{v_3, v_1\}$, $N_1[v_2] = \{v_3, v_2\}$, $N_1[v_4] = \{v_3, v_4\}$, , $N_1[v_5] = \{v_3, v_5\}$, so $\sigma_1(v_1, v_3) = \sigma_1(v_2, v_3) = \sigma_1(v_4, v_3) = \sigma_1(v_5, v_3) = 2/\sqrt{10} \approx 0.63$. We can also calculate the structural similarity of $G_2$ to $G_7$ in Fig. 3(b) to Fig. 3(g) which are the *de-temporal* graph of $\mathcal{G}$ at time 2 to 7 respectively.

Considering the *de-temporal* graph $G_1, G_2, G_3$ from Fig. 3(a) to (c), we can find that the $(3, 0.6)$-connected edges are edge $(v_1, v_3)$, $(v_2, v_3)$, $(v_4, v_3)$. It's easy to derive that those edges exist in three timestamps and the 0.6-stable similarity of the nodes on them are larger than 3.

Reconsider the *de-temporal* graph $\{G_1, G_2, G_3\}$. We can find that node $v_3$ is a $(3, 3, 0.6)$-stable core since it has more than three 0.6-neighbors in timestamps $[1, 2, 3]$. But $v_3$ is not a $(3, 3, 0.7)$-stable core, since $v_3$ has no 0.7-neighbors in $G_1$. By Definition 5, we can find the $(3, 3, 0.6)$-stable cluster up-to-now is $\{v_3, v_1, v_2, v_4\}$, since $v_3$ is a $(3, 3, 0.6)$-stable core and $\{v_1, v_2, v_4\}$ are all structure-reachable to $v_3$ (only edge $(v_1, v_3)$, $(v_2, v_3)$, $(v_4, v_3)$ are $(3, 0.6)$-connected edges). ■

**Challenges.** Unlike the traditional density-based graph clustering problem [3], [18], the challenge in our problem is to compute the $(\mu, \tau, \epsilon)$-stable cores. This is because we need to seek a frequent star-shaped structure over $\mathcal{T}$ snapshots to identify a stable core (see Definition 1), which is very expensive in practice. Specifically, to determine whether $u$ is a $(\mu, \tau, \epsilon)$-stable core or not, a basic algorithm is to compute the set of $\epsilon$-neighbors of $u$ (i.e., $N_i^{\epsilon}(u)$) in each snapshot. Then, for each snapshot, we treat $N_i^{\epsilon}(u)$ as a transaction, and then invoke any existing maximal frequent pattern mining algorithm [23] with support threshold $\tau$ to find all maximal frequent patterns. If there is a maximal frequent pattern with cardinality no less than $\mu$, the node $u$ is a $(\mu, \tau, \epsilon)$-stable core by Definition 1. To identify all the stable cores, we have to invoke such a basic algorithm $n$ times, which is very costly.

应从这也最少的翻阅下手

To tackle this challenge, we will develop several powerful punning techniques to efficiently compute the stable cores.

## 3 The Proposed Algorithms

In this section, we first introduce a basic clustering framework to solve our problem by adapting the state-of-the-art density-based graph clustering framework (PSCAN) [18]. Then, we develop an improved algorithm with several novel pruning techniques to compute the $(\mu, \tau, \epsilon)$-stable clusters efficiently. The striking features of our algorithm are twofold: (i) it can avoid repeated $\epsilon$-stable similarity computation, and (ii) it significantly reduces unnecessary computation for determining $(\mu, \tau, \epsilon)$-stable cores.

优点

### 3.1 The Basic Clustering Framework

Similar to the PSCAN framework, our clustering framework comprises two steps. In the first step, the algorithm groups the $(\mu, \tau, \epsilon)$-stable cores, and then in the second step, it clusters non-core nodes. The details of our framework, TSCAN-B (Temporal SCAN-Basic), is shown in Algorithm 1.

Algorithm 1 first initializes an empty graph $G_c$ to maintain the computed stable cores and connected edges (line 2). Then, for each node $u \in V$, the algorithm determines whether it is a stable core (lines 3-4). As discussed in Section 2, we can make use of any maximal frequent pattern mining algorithm [23] to identify the stable cores. If $u$ is a stable core, the algorithm adds it into $G_c$ and traverses its neighbors. For each $(\tau, \epsilon)$-connected neighbor $v$ of $u$ (line 9), if $v$ is also a stable core (line 10), the algorithm inserts $v$ and an edge $(u, v)$ into $G_c$ (line 11), which means that $u$ and $v$ will be grouped together. After exploring all nodes, the algorithm computes the connected components in $G_c$ (line 13), and assigns the non-core nodes to their corresponding stable clusters using a similar method proposed in [18] (line 14). In Algorithm 1, we can compute the $\epsilon$-stable similarity $S_\epsilon(u, v)$ by Definition 2 (lines 16-26). Clearly, $|N_i(u)|$ and $|N_i(v)|$ are easy to get and they can be stored as constants in the structure of the temporal network (similar to adjacency list in the static graph). We can iterate $\{N_i(u) + N_i(v)\}$ by node $w$ and use a $Hashset$ to store the considering node $w$. The value $Hashset.w$ will be increased by 1 if it has been existed (lines 19-22). Next, $|N_i(u)| \cap |N_i(v)|$ will be the number of $w$ satisfying that $HashSet.w > 0$ (line 23). The time complexity of the whole process is $O(m)$ since it only needs constant time to check whether $HashSet.has(w)$ (line 21). The algorithm terminates early if $S_\epsilon(u, v) \geq \tau$ (line 26). Note that the algorithm also keeps all the computed $\epsilon$-stable similarity scores $(S_\epsilon(u, v))$ in the main memory to avoid redundant computation (lines 7-8). The space overhead to store all those $\epsilon$-stable similarities is bounded by the number of edges in the de-temporal graph $G$. The correctness of our algorithm can be easily derived using a similar argument as shown in [18].

复刻
思路创新
?

?

Note that the main difference between Algorithm 1 and PSCAN [18] is that our algorithm relies on computing $\epsilon$-stable similarities and $(\mu, \tau, \epsilon)$-stable cores, while PSCAN is to calculate traditional structural similarities and cores. As discussed in Section 2, the computation of $(\mu, \tau, \epsilon)$-stable cores are quite nontrivial, since it needs to compute a

**Algorithm 1:** TSCAN-B $(\mathcal{G}, \mu, \tau, \epsilon)$

```
   Input: Temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters $\mu, \tau, \epsilon$.
   Output: The set of $(\mu, \tau, \epsilon)$-stable clusters $\mathcal{C}$
1  Let $G = (V, E)$ be the de-temporal graph of $\mathcal{G}$;
2  Initialize $G_c = (\phi, \phi)$ as an empty graph;
3  for each node $u \in V$ do
4      if isStableCore($u$) then        ? $S_\epsilon$计算怎会在其后？
5          Add $u$ into $G_c$ ;
6          for each unvisited node $v \in N(u)$ do
7              if $S_\epsilon(u, v)$ has not been computed then
8                  Compute $S_\epsilon(u, v)$ and store it;
9              if $S_\epsilon(u, v) \geq \tau$ then        由一个核出发添加相邻核结点
10                 if isStableCore($v$) then
11                     Add node $v$ and edge $(u, v)$ into $G_c$;
12     Mark $u$ as a visited node;
13 $\mathbb{C} \leftarrow$ the set of connected components in $G_c$;
14 $\mathcal{C} \leftarrow \{C \cup_{u \in C} N_{(\tau,\epsilon)}(u) | C \in \mathbb{C}\}$; // cluster non-core nodes
15 return $\mathcal{C}$;

16 Procedure Compute $S_\epsilon(u, v)$
17 $S_\epsilon(u, v) \leftarrow 0$;
18 for $i \leftarrow 1 : \mathcal{T}$ do
19     $HashSet \leftarrow \emptyset$;
20     for node $w$ in $\{N_i(u) \cup N_i(v)\}$ do
21         if $HashSet.has(w)$ then $HashSet.w++$;
22         else $HashSet.w \leftarrow 0$;        HashSet一直为 ∅？
23     $|N_i(u)| \cap |N_i(v)| \leftarrow \#w$ satisfying $HashSet.w > 0$;
24     if there is an temporal edge $(u, v, t)$ such that
       $t \in (t_{i-1}, t_i]$ then
25         if $\sigma_i(u, v) \geq \epsilon$ then $S_\epsilon(u, v) \leftarrow S_\epsilon(u, v) + 1$;
26     if $S_\epsilon(u, v) \geq \tau$ then break;
27 return $S_\epsilon(u, v)$;
```

**Algorithm 2:** WeakCore $(\mathcal{G}, \mu, \tau, \epsilon)$

```
1  Let $G = (V, E)$ be the de-temporal graph of $\mathcal{G}$;
2  Initialize the set of weak cores WC $= \emptyset$;
3  for each node $u \in V$ do        cd是 $S_\epsilon \geq \tau$ 的邻居数
4      cd$(u) \leftarrow 0$; $\overline{\text{cd}}(u) \leftarrow |N(u)|$;        $\overline{\text{cd}}$是 $S_\epsilon < \tau$ 的 ~
5  for each node $u \in V$ do
6      if cd$(u) < \mu$ and $\overline{\text{cd}}(u) \geq \mu$ then
7          for each node $v \in N(u)$ do
8              if $S_\epsilon(u, v)$ has not been computed then
9                  Compute $S_\epsilon(u, v)$ and store it;
10                 if $S_\epsilon(u, v) \geq \tau$ then
11                     cd$(u) \leftarrow$ cd$(u) + 1$; cd$(v) \leftarrow$ cd$(v) + 1$;
12                 else
13                     $\overline{\text{cd}}(u) \leftarrow \overline{\text{cd}}(u) - 1$; $\overline{\text{cd}}(v) \leftarrow \overline{\text{cd}}(v) - 1$;
14             if cd$(u) \geq \mu$ or $\overline{\text{cd}}(u) < \mu$ then break;        为何 $\overline{\text{cd}}(u)<\mu$ 要break？
15     if cd$(u) \geq \mu$ then add $u$ into WC;
16 return WC;
```

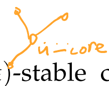
frequent star-shaped structure over $\mathcal{T}$ snapshots. Below, we develop a novel algorithm to efficiently compute the stable cores.

从每个节点都算$S_\epsilon$变为第二次只算WC内的点.

## 3.2 Efficient Stable Core Computation

To speed up the stable core computation, here we propose two novel pruning techniques to quickly prune the unqualified nodes that are definitely not stable cores.

### 3.2.1 The Weak Core Pruning



← star形 T包括μ个快照的结点

By Definition 1, if $u$ is a $(\mu, \tau, \epsilon)$-stable core, there is a frequent star $T$ rooted at $u$ such that the nodes $u$ and $v$ for each edge $(u, v) \in T$ are structurally similar in no less than $\tau$ snapshots. That is to say, each edge $(u, v) \in T$ is a $(\tau, \epsilon)$-connected edge by Definition 3. Since $T$ has $\mu$ edges by Definition 1, a $(\mu, \tau, \epsilon)$-stable core has at least $\mu$ $(\tau, \epsilon)$-connected neighbors. As a result, we can first check a node $u$ whether it has $\mu$ $(\tau, \epsilon)$-connected neighbors. If that is the case, $u$ is a candidate for a $(\mu, \tau, \epsilon)$-stable core, otherwise it is definitely not a $(\mu, \tau, \epsilon)$-stable core. More formally, we define a node $u$ as a weak core if $|N_{\tau,\epsilon}(u)| \geq \mu$.

不是已经成了吗？

用$S_\epsilon$筛选

***Definition 6 (weak core).*** Given a temporal graph $\mathcal{G}$ and parameters $\tau, \epsilon, \mu$, node $u$ is a weak core if $|N_{\tau,\epsilon}(u)| \geq \mu$.

stable-c ⊆ weak

?

***Lemma 1.*** Any $(\mu, \tau, \epsilon)$-stable core must be a weak core.

***Proof 1.*** The proof can be easily obtained by definitions, thus we omit it for brevity.

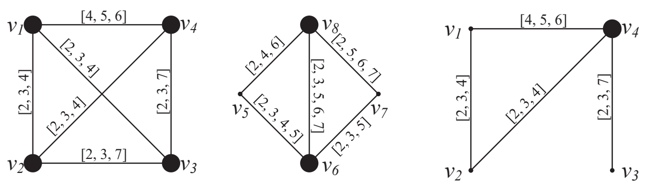
u的 $S_\epsilon(u) > \tau$ 的邻居数 ≥ μ. 即 WC

Based on Lemma 1, we can first identify all the weak cores in $\mathcal{G}$, and then verify whether they are stable cores by traditional maximal frequent pattern mining algorithms. The remaining question is how can we efficiently compute the weak cores? A naive approach is to compute $|N_{\tau,\epsilon}(u)|$ to determine whether $u$ is a weak core. This naive approach, however, is inefficient, since it needs to compute the $\epsilon$-stable similarities between $u$ and every neighbor of $u$. Moreover, it may incur redundant similarity computation between different nodes. To tackle these limitations, we propose a new algorithm to compute the weak core efficiently. The key idea of our algorithm is to maintain lower and upper bounds of $|N_{\tau,\epsilon}(u)|$ for each node $u$. If the lower bound of $|N_{\tau,\epsilon}(u)|$ for $u$ is no less than $\mu$, $u$ must be a weak core. Also, if its upper bound is smaller than $\mu$, $u$ is not a weak core, and thus can be pruned. The detailed description is shown in Algorithm 2.

In Algorithm 2, cd$(u)$ denotes the number of the computed $(\tau, \epsilon)$-connected neighbor of $u$, while $\overline{\text{cd}}(u)$ is an upper bound of $|N_{\tau,\epsilon}(u)|$. Initially, we set cd$(u) = 0$ and $\overline{\text{cd}}(u) = |N(u)|$ for each $u \in V$ (lines 3-4). Clearly, if cd$(u) \geq \mu$, $u$ is a weak core; and if $\overline{\text{cd}}(u) < \mu$, $u$ must not be a weak core. For each node $u$ with cd$(u) < \mu$ and $\overline{\text{cd}}(u) \geq \mu$, the algorithm computes all the $\epsilon$-stable similarities between $u$ and its neighbors and updates cd$(u)$ and $\overline{\text{cd}}(u)$ (lines 6-13). If $S_\epsilon(u, v) \geq \tau$, both cd$(u)$ and cd$(v)$ increase by 1, otherwise both $\overline{\text{cd}}(u)$ and $\overline{\text{cd}}(v)$ decrease by 1 (lines 10-13). The similarity computation procedure can terminate early if cd$(u) \geq \mu$ and $\overline{\text{cd}}(u) < \mu$ (line 14). Note that Algorithm 2 not only prunes unnecessary similarity computation by the early termination rule (line 14), but it can also avoid repeated similarity computation by keeping all computed $\epsilon$-stable similarities in the main memory.

在开始时

**Example 2.** We show the process of finding the weak core in the temporal graph of Fig. 2 with $(\mu = 3, \tau = 3, \epsilon = 0.7)$. The algorithm first initializes the upper bound $\overline{\text{cd}}(u)$ to be the degree in the de-temporal graph $G$. Then, we enumerate the nodes in the de-temporal graph to check whether they have more than three $(3, 0.7)$-connected neighbors. We can find that only $v_7$ has less than 3 neighbors in the de-



(a) $\epsilon = 0.7$ (b) $\epsilon = 0.75$

Fig. 4. WeakCore ($\mu = 3, \tau = 3$)

temporal graph, and it will not be taken into consideration. Subsequently, we compute the $S_\epsilon(u, v)$ on demand. Once a node already has three $(3, 0.7)$-connected neighbors, it will be added into the weak core set.

Fig. 4(a) shows the weak core in the temporal graph of Fig. 2 with ($\mu = 3, \tau = 3, \epsilon = 0.7$). The enlarged points are weak cores and all the edges are $(3, 0.7)$-connected edges. The vector on each edge is the timestamps in which the structural similarity of two nodes are larger than 0.7. It can be seen that in Fig. 4(a) the weak cores are $\{v_1, v_2, v_3, v_4, v_6, v_8\}$ and if $\epsilon$ raises to 0.75 in Fig. 4(b), the weak core has only one node $\{v_4\}$. Hence in a large temporal social network, the graph will be reduced by the weak core pruning rule with proper parameter settings. ■

### 3.2.2 The Strong Core Pruning

To further prune the unpromising nodes, we propose a stronger pruning rule called strong core pruning. A node $u$ is a strong core if it is a weak core satisfying $[\sum_{i=1} \mathcal{I}(|N_i^\epsilon(u)| \geq \mu)] \geq \tau$.

要在至少τ层上相似性 $\sigma > \epsilon$ 的邻居数需 $\geq \mu$

***Definition 7 (strong core).*** Given a temporal graph $\mathcal{G}$ and parameters $\tau, \epsilon, \mu$, a node $u$ is a strong core if $|N_{\tau,\epsilon}(u)| \geq \mu$ and $[\sum_{i=1} \mathcal{I}(|N_i^\epsilon(u)| \geq \mu)] \geq \tau$.

符号函数

The following lemma shows that any stable core is also a strong core.

***Lemma 2.*** Any $(\mu, \tau, \epsilon)$-stable core must be a strong core.

***Proof 2.*** For any $(\mu, \tau, \epsilon)$-stable core $u$, $u$ is a weak core, thus we have $|N_{\tau,\epsilon}(u)| \geq \mu$. By Definition 1, there exists a star $T$ rooted at $u$ such that $u$ and $v$ are structurally similar for any $(u, v) \in T$ in at least $\tau$ snapshots. Since $|T| \geq \mu + 1$, there exist no less than $\tau$ snapshots, and in each of them the number of $u$'s $\epsilon$-neighbors is no smaller than $\mu$. As a result, we have $[\sum_{i=1} \mathcal{I}(|N_i^\epsilon(u)| \geq \mu)] \geq \tau$. So, any $(\mu, \tau, \epsilon)$-stable core must be a strong core. ■

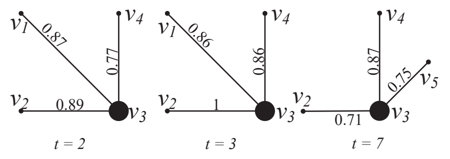
To compute the strong cores, we first identify all the weak cores by invoking Algorithm 2, since all strong cores must be contained in the set of weak cores. Then, for each weak core $u$, we verify whether $[\sum_{i=1} \mathcal{I}(|N_i^\epsilon(u)| \geq \mu)] \geq \tau$ holds. To efficiently implement this algorithm, we can maintain lower and upper bounds of $[\sum_{i=1} \mathcal{I}(|N_i^\epsilon(u)| \geq \mu)]$ for each weak core $u$. Initially, we set $[\sum_{i=1} \mathcal{I}(|N_i(u)| \geq \mu)]$ as an upper bound for $[\sum_{i=1} \mathcal{I}(|N_i^\epsilon(u)| \geq \mu)]$, because the number of neighbors of $u$ is no less than the number of $\epsilon$-neighbors in the same snapshot. The detailed implementation of our algorithm is shown in Algorithm 3.

In Algorithm 3, cs$(u)$ denotes the number of <u>c</u>onsidered <u>s</u>napshots in which $u$ has no less than $\mu$ $\epsilon$-neighbors. Initially, cs$(u) = 0$, and cs$(u) = [\sum_{i=1} \mathcal{I}(|N_i^\epsilon(u)| \geq \mu)]$ if all the snapshots are considered. $\overline{\text{cs}}(u)$ is an upper bound

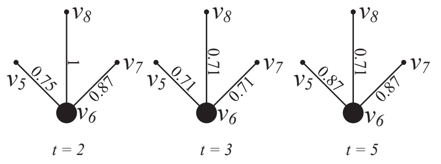
**Algorithm 3:** StrongCore $(\mathcal{G}, \mu, \tau, \epsilon)$

```
1  WC ←WeakCore $(\mathcal{G}, \mu, \tau, \epsilon)$;
2  Initialize the set of strong cores SC ← $\emptyset$;
3  for each node $u \in$ WC do
4      cs$(u) \leftarrow 0$; $\overline{\text{cs}}(u) \leftarrow 0$;
5      for $i \leftarrow 1 : \mathcal{T}$ do
6          sd$_i(u) \leftarrow 0$; $\overline{\text{sd}}_i(u) \leftarrow 0$;
7          if $|N_i(u)| \geq \mu$ then
8              $\overline{\text{cs}}(u) \leftarrow \overline{\text{cs}}(u) + 1$; $\overline{\text{sd}}_i(u) \leftarrow |N_i(u)|$;
9  for each node $u \in$ WC do
10     if cs$(u) < \tau$ and $\overline{\text{cs}}(u) \geq \tau$ then
11         for $i \leftarrow 1 : \mathcal{T}$ do
12             if $\overline{\text{sd}}_i(u) \geq \mu$ then
13                 for each $v \in N_i(u)$ do
14                     if $\sigma_i(u, v)$ has not been computed then
15                         Compute $\sigma_i(u, v)$ and store it;
16                         if $\sigma_i(u, v) \geq \epsilon$ then
17                             sd$_i(u) \leftarrow$ sd$_i(u) + 1$;
18                         else
19                             $\overline{\text{sd}}_i(u) \leftarrow \overline{\text{sd}}_i(u) - 1$;
20                         if sd$_i(u) \geq \mu$ then
21                             cs$(u) \leftarrow$ cs$(u) + 1$;
22                         if $\overline{\text{sd}}_i(u) < \mu$ then
23                             $\overline{\text{cs}}(u) \leftarrow \overline{\text{cs}}(u) - 1$;
24                         if $v \in$ WC and $\overline{\text{sd}}_i(v) \geq \mu$ then
25                             Update sd$_i(v)$, $\overline{\text{sd}}_i(v)$, cs$(v)$,
                               and $\overline{\text{cs}}(v)$ as lines 17-23;
26                         if sd$_i(u) \geq \mu$ or $\overline{\text{sd}}_i(u) < \mu$ then
27                             goto line 11;
28                     if cs$(u) \geq \tau$ or $\overline{\text{cs}}(u) < \tau$ then
29                         goto line 30;
30     if cs$(u) \geq \tau$ then add $u$ into SC;
31 return SC;
```

of $[\sum_{i=1} \mathcal{I}(|N_i^\epsilon(u)| \geq \mu)]$ for $u$, and it is initialized by $[\sum_{i=1} \mathcal{I}(|N_i(u)| \geq \mu)]$ (lines 3-8). For each snapshot, Algorithm 3 also makes use of a similar lower and upper bounding trick to compute $|N_i^\epsilon(u)|$. Specifically, for the $i$-th snapshot, sd$_i(u)$ denotes the number of computed $\epsilon$-neighbors of $u$ in the $i$-th snapshot, and $\overline{\text{sd}}_i(u)$ is the upper bound of $|N_i^\epsilon(u)|$ which is initialized by $|N_i(u)|$ (line 6-8). The algorithm progressively updates cs$(u)$ and $\overline{\text{cs}}(u)$ (lines 20-22). If cs$(u) \geq \tau$ or $\overline{\text{cs}}(u) < \tau$, the algorithm can terminate early (line 28). Similarly, if sd$_i(u) \geq \mu$ or $\overline{\text{sd}}_i(u) < \mu$, the algorithm is also able to terminate the computation of $|N_i^\epsilon(u)|$ (lines 26-27). Note that the algorithm maintains all the computed $\sigma_i(u, v)$ in the main memory to avoid redundant computation (lines 14-15). Since the number of weak cores is typically much smaller than $n$, the total space overhead to store all the computed $\sigma_i(u, v)$ is also very small. In our algorithm, it is sufficient to store a boolean variable to denote whether $\sigma_i(u, v) \geq \epsilon$. Since the number of weak cores is typically much smaller than $n$, the total space overhead to store all such boolean variables can be ignored.

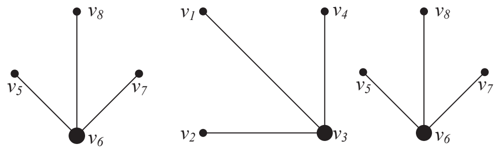


(a) The 0.7-**neighbors** of $v_3$



(b) The 0.7-**neighbors** of $v_6$

Fig. 5. StrongCore ($\mu = 3, \tau = 3, \epsilon = 0.7$)



(a) $\epsilon = 0.7$ (b) $\epsilon = 0.6$

Fig. 6. StableCore and StableCluster ($\mu = 3, \tau = 3$)

**Algorithm 4:** StableCore $(\mathcal{G}, \mu, \tau, \epsilon)$

```
1 SC ←StrongCore $(\mathcal{G}, \mu, \tau, \epsilon)$; $S \leftarrow \emptyset$;
2 Call the stored $\sigma_i(u, v)$;
3 for each $u \in$ SC do
4     Invoke a existing maximal frequent pattern mining
      algorithm Apriori to determine whether $u$ is a stable
      core;
5     if $u$ is a stable core then
6         Add $u$ into $S$;
7 return $S$;
```

**Example 3.** To find the strong cores, we first invokes Algorithm 2 to find the WeakCore. In this case, we get the weak core in Fig. 4(a) with ($\mu = 3, \tau = 3, \epsilon = 0.7$). Then, Algorithm 3 derives WeakCore to initialize $\overline{\text{cs}}(u)$ and $\overline{\text{sd}}_i(u)$. We can find that all the $\overline{\text{cs}}(u)$ of $\{v_1, v_2, v_4\}$ are less than 3 and they will be pruned firstly. Note that, $\overline{\text{cs}}(v_8)$ and $\overline{\text{sd}}_7(v_8)$ are initialized as 3. As shown in Fig. 3(g), $v_8$ has only two 0.7-neighbors and thus it will be pruned. After all the WeakCore are visited, the StrongCore only consists of nodes $v_3$ and $v_6$.

Fig. 5(a) shows the 0.7-neighbors of $v_3$ at time $\{2, 3, 7\}$. The number on each edge is the structural similarity of the two nodes in the corresponding time. It should be noticed that the 0.7-neighbors of $v_3$ at time $\{2, 3\}$ are $\{v_1, v_2, v_4\}$, but the 0.7-neighbors at time $\{7\}$ are $\{v_2, v_4, v_5\}$. By the definition of StrongCore, we can easily verify that this result is correct which indicates the main difference between StrongCore and StableCore. Fig. 5(b) shows the 0.7-neighbors of $v_3$ at time $\{2, 3, 5\}$. The 0.7-neighbors of $v_6$ in time $\{2, 3, 5\}$ are all the same. According to the definition of StableCore, $v_3$ is not a StableCore but $v_6$ is. Intuitively, $v_3$ is not stable since it has different similar neighbors at different time. ■

**Algorithm 5:** TSCAN-A $(\mathcal{G}, \mu, \tau, \epsilon)$

```
1 $G_c \leftarrow$ StableCore $(\mathcal{G}, \mu, \tau, \epsilon)$;
2 for each node $u \in G_c$ do
3     for each unvisited node $v \in N(u) \cap G_c$ do
4         if $S_\epsilon(u, v) \geq \tau$ then
5             Add an edge $(u, v)$ into $G_c$;
6     Mark $u$ as a visited node;
7 $\mathbb{C} \leftarrow$ the set of connected components in $G_c$;
8 $\mathcal{C} \leftarrow \{C \cup_{u \in C} N_{(\tau,\epsilon)}(u) | C \in \mathbb{C}\}$; // cluster non-core nodes
9 return $\mathcal{C}$;
```

### 3.2.3 *Computing the Stable Core and Stable Cluster*

The process of finding the StableCore can be shown at Algorithm 4. By Lemma 2, we can treat the strong cores as the candidates for the stable cores. For each strong core $u$, we apply an existing maximal frequent pattern mining algorithm Apriori [23] to derive whether $u$ is a $(\mu, \tau, \epsilon)$-stable core.

Recall algorithm 1, after getting the StableCore, algorithm 1 computes the connected components in $G_c$ in line 11, and assigns the non-core nodes to their corresponding stable clusters. The process of finding the $(\mu, \tau, \epsilon)$-stable clusters by StableCore can be shown at Algorithm 5. It first gets the StableCore by Algorithm 4. Then it clusters the cores by checking $S_\epsilon(u, v)$. Next it finds the connected components of the cores and clusters the non-core nodes to identify all the $(\mu, \tau, \epsilon)$-stable clusters.

TSCN-A

Fig. 6 shows the stable core and stable cluster of the temporal graph in Fig. 2(a) with ($\mu = 3, \tau = 3$). The enlarged nodes are stable cores and the other nodes are structure-reachable from it via a $(\tau, \epsilon)$-connected edge. The stable cores can be computed by invoking Algorithm 3 to get StrongCore first and then check whether those cores and their similar neighbors are frequent in more than $\tau$ times. Fig. 6(a) shows that if $\epsilon = 0.7$, $v_6$ is the only stable core and it forms a stable cluster of $\{v_6, v_5, v_7, v_8\}$. Fig. 6(b) shows that if $\epsilon = 0.6$, $\{v_3, v_6\}$ are stable cores and there are two stable clusters, $\{v_3, v_1, v_2, v_4\}$ and $\{v_6, v_5, v_7, v_8\}$. It should be noted that those two stable clusters are not reachable to each other.

### 3.2.4 *Complexity analysis*

As confirmed in our experiments, the number of strong cores is very small, thus the TSCAN algorithm with reducing by StrongCore is very efficient in practice. We analyze the time complexity of TSCAN-A below.

**Worst case.** Let $\mathcal{M}$ be the time cost of the maximal frequent pattern mining algorithm. At the worst case, $\mathcal{M}$ is $O(C_\mathcal{T}^\tau m^\tau)$ since it need to union the edge set $m$ for $\tau$ times and the total number of time sets is $C_\mathcal{T}^\tau$.

**Our proposed algorithm.** Let $|s|$ be the number of strong cores. In Algorithm 1, the time cost to compute $S_\epsilon(u, v)$ can be bounded by $O(m)$ by an incremental computation procedure. Note that TSCAN only computes each $S_\epsilon(u, v)$ once, the total time cost for the similarity computation is $O(m'm)$ in the worst case, where $m' \leq m$ is the number of edges in the de-temporal graph $G$. It is easy to show that the time complexity of Algorithm 3 can also be bounded by

TABLE 2
Statistics of datasets

| Dataset | $\lvert V\rvert = n$ | $\lvert E\rvert = m'$ | $\lvert\mathcal{E}\rvert = m$ | $d_{\max}$ | $\lvert\mathcal{T}\rvert$ | Time scale |
|---|---|---|---|---|---|---|
| Chess | 7,301 | 55,899 | 62,385 | 230 | 99 | month |
| Lkml | 26,885 | 159,996 | 328,092 | 14,172 | 96 | month |
| Enron | 86,978 | 297,456 | 499,983 | 4,311 | 48 | month |
| DBLP | 1,729,816 | 8,546,306 | 12,007,380 | 5,980 | 78 | year |

TABLE 3
Parameters adopted in the experiments and their reference values

| Symbol | Description | Value | Default Value |
|---|---|---|---|
| $\mu$ | number of similar neighbors | 2-8 | 5 |
| $\tau$ | number of similar timestamps | 2-8 | 3 |
| $\epsilon$ | degree of similarity | 0.2-0.8 | 0.5 |

$O(m'm)$. In Algorithm 5, the time complexity of finding the stable clusters can be bounded by $O(m)$. As a result, the time complexity of TSCAN with reducing by StrongCore is $O(m'm+|s|C_{\mathcal{T}}^{\tau}|s|^{\tau})$ in the worst case. Since our algorithm is integrated with several powerful pruning rules, $|s|$ is small and the practical performance of our algorithm is much better than the worst-case complexity. In our experiments, we show that our proposed algorithm is very fast in many real-world temporal graphs, and it can be scalable to million-sized temporal graphs.

## 4 Experiments

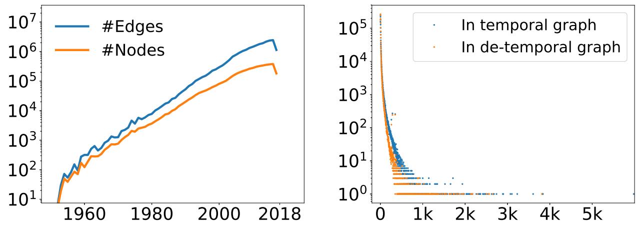
In this section, we conduct extensive experiments to evaluate the effectiveness and efficiency of the proposed algorithms. To the best of our knowledge, we are the first to study the problem of mining stable communities in temporal graphs. We implement four different algorithms for comparison:

- PSCAN-W is a baseline algorithm which clusters a transformed weighted de-temporal graph to find the stable clusters. Specifically, for each edge $(u, v)$, it first sets the number of temporal edges between $u$ and $v$ as a weight of $(u, v)$. Then, it invokes the state-of-the-art density-based graph clustering algorithm (PSCAN) [18] to find the clusters in the transformed weighted graph and the cores in PSCAN-W should have several similar neighbors whose sum of weights are larger than $\mu\tau$ in this paper.
- TSCAN-B is our basic algorithm without any pruning rules that uses StableCore for clustering (Algorithm 1).
- TSCAN-A is our improved algorithm to get stable cores with all the pruning rules in Section 3 and then uses StableCore for density-based clustering (Algorithm 5).
- TSCAN-S is a variant of our TSCAN algorithm that makes use of StrongCore for density-based clustering (replaced line 1 of Algorithm 5 by $G_c \leftarrow$ StrongCore).

All algorithm are implemented in Python. All the experiments are conducted on a server of Linux kernel 4.4 with Intel Core(TM) i5-6500@3.20GHz and 32 GB main memory.

**Datasets.** We use four different types of real-world temporal networks in the experiments. The detailed statistics of our datasets are summarized in Table 2.

- Chess is a temporal network where each temporal edge represents two chess players playing a game at time $t$ from 1998 to 2006.



(a) Numbers of nodes and edges in different snapshots (b) Distributions of the nodes' degree in temporal graph and de-temporal graph

Fig. 7. Data descriptions of DBLP

- Lkml is a temporal communication network of the Linux kernel mailing list, where a temporal edge $(u, v, t)$ denotes an email communication from a user $u$ to $v$ at time $t$ from 2001 to 2011.
- Enron is an email communication network between employees of Enron from 1999 to 2003, where each temporal edge denotes the email communication between the employees in Enron Corp.
- DBLP is a temporal collaboration network of authors in DBLP from 1940 to Feb. 2018.

In Table 2, $d_{\max}$ denotes the maximum number of temporal edges associated with a node, and $|\mathcal{T}|$ denotes the number of snapshots. The first three datasets are downloaded from konect.uni-koblenz.de, and DBLP is extracted from dblp.uni-trier.de/xml/.

**Parameter settings.** Table 3 shows the parameters adopted in the experiments and their reference values. There are three parameters $\mu$, $\tau$, and $\epsilon$ in our algorithm to control the cluster quality. For the parameter $\mu$, we vary it from 2 to 8 with a default value of 5. We vary $\tau$ from 2 to 8 with a default value of 3, and vary $\epsilon$ from 0.2 to 0.8 with a default value of 0.5. Unless otherwise specified, the values of the other parameters are set to their default values when varying a parameter.

**Data descriptions.** The size of temporal data in real world is growing in a rapid speed. Also, the distributions of a wide variety of social networks approximately follow a power law over a wide range of magnitudes. The degree in the temporal graphs also follows the power law distribution. Figure 7 describes those two properties of DBLP. Figure 7(a) shows that the number of nodes and edges in DBLP are growing exponentially since the y-axis is in log scale. It also means that the number of publications in computer science are growing exponentially by year. Figure 7(b) demonstrates that the distributions of degree in temporal graph and de-temporal graph follow a power law distribution. We can find that the degree of most nodes will be no more than 300 in the de-temporal graph and 500 in the temporal graph. It means that one researcher is vary difficult to co-operate with more than 300 people in his whole life. All the results above are consistent with our common sense.

### 4.1 Effectiveness Testing

Since most existing metrics (e.g., modularity) for measuring the cluster quality are tailored for traditional graphs, we introduce four goodness metrics for clusters to evaluate the cluster quality of different clustering algorithms for temporal graphs. Those metrics are motivated by *separability*, *density*, *cohesiveness* and *clustering coefficient* which are used to evaluate the communities in static graphs [2].

Let $\mathcal{C}$ be a set of clusters obtained by different algorithms. Table 4 shows the brief introductions of the evaluation metrics. The full descriptions of them are as follows.

- **Average Separability (AS)** captures the intuition that good communities are well-separated from the rest of the network, meaning that they have relatively few edges pointing from $\mathcal{C}$ to rest of the network: AS $\triangleq \sum_{C_i\in\mathcal{C}}[\frac{|\{(u,v,t)\in\mathcal{E}:u\in C_i,v\in C_i\}|}{|\{(u,v,t)\in\mathcal{E}:u\in C_i,v\notin C_i\}|}]/|\mathcal{C}|$, where $C_i$ is a cluster in $\mathcal{C}$ and it measures the ratio between the internal and external number of temporal edges of $\mathcal{C}$.
- **Average Density (AD)** builds on intuition that good communities are well connected. It measures the fraction of the temporal edges that appear between the nodes in $\mathcal{C}$: AD $\triangleq \sum_{C_i\in\mathcal{C}}[\frac{\sum_{v_j\in C_i} d_{C_i}(v_j)}{|C_i|}]/|\mathcal{C}|$, where $C_i$ is a cluster in $\mathcal{C}$ and $d_{C_i}(v_j)$ denotes the number of temporal edges that are associated with $v_j$ in the cluster $C_i$.
- **Average Cohesiveness (AC)** characterizes the internal structure of the community. Intuitively, a good community should be internally well and evenly connected, i.e., it should be relatively hard to split a community into two sub communities. We characterize this by the conductance of the internal cut and adapt it into temporal graph: AC $\triangleq \sum_{C_i\in\mathcal{C}} max_{S\subseteq C_i}\phi(S)$, where $\phi(S)$ is the conductance of $S$ measured in the induced temporal subgraph by $S$. Intuitively, conductance measures the ratio of edges in $S$ that point out side the set and edges inside $S$.
- **Average Clustering Coefficient (ACC)** is based on the premise that network communities are manifestations of locally inhomogeneous distributions of edges, because pairs of nodes with common neighbors are more likely to be connected with each other: ACC $\triangleq \sum_{C_i\in\mathcal{C}}[\sum_{v_j\in C_i}\frac{\#edge(N(v_j,C_i))}{d_{C_i}(v_j)}]/|\mathcal{C}|$, where $\#edge(N(v_j,C_i))$ is the number of temporal edges in $C_i$ whose two end nodes are $v_j$'s neighbors and $d_{C_i}(v_j)$ denotes the number of temporal edges that are associated with $v_j$ in the cluster $C_i$.

Intuitively, a stable cluster should have high AS, AD, AC and ACC values. It is obvious that the final clusters in TSCAN-B and TSCAN-A are same because they both make use of StableCore for density-based clustering. So, we do effectiveness testing among PSCAN-W, TSCAN-S and TSCAN-A. Unless otherwise specified, in the following effectiveness testings, the evaluation values are normalized so each maximum single value is 1.

**Exp-1. Distributions of the evaluation value.** We test the goodness value of the computed clusters in every detemporal graph $G_i$ with $i$ varies from 1980 to 2018 in DBLP. The AS, AD, AC and ACC distributions of different algorithms in DBLP under the default parameter setting are reported in Fig. 8. Similar results can also be observed using the other parameter settings or datasets.

As can be seen, TSCAN-A significantly outperforms two baselines in terms of AS, AD and ACC metrics. Considering AC metric, the differences between PSCAN-W, TSCAN-S and TSCAN-A are not too much. This is because AS metric finds the maximum conductance of the clusters and both the algorithms can find a stable cluster which is not too bad. Both TSCAN-S and TSCAN-A perform much better than PSCAN-W. All the four goodness value are increasing by the year because the number of papers are increasing by the year (see Fig. 7(a)). It can be seen that the AS, AD and ACC value of PSCAN-W are not stable. The reason could be that PSCAN-W does not consider the stable similarities for clustering, thus the resulting clusters may not truly reflect stable clusters.

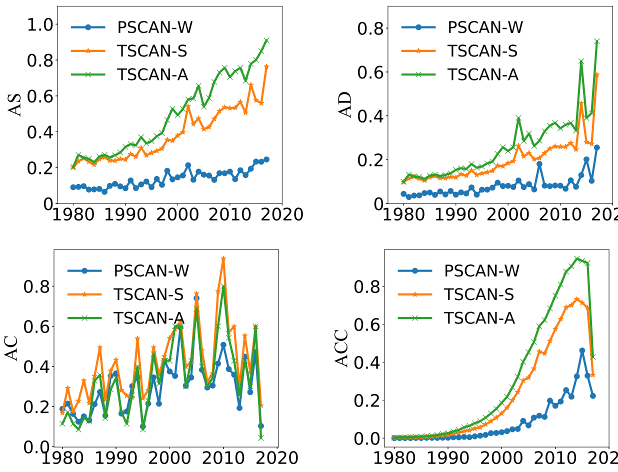


Fig. 8. Goodness values in different snapshots of DBLP

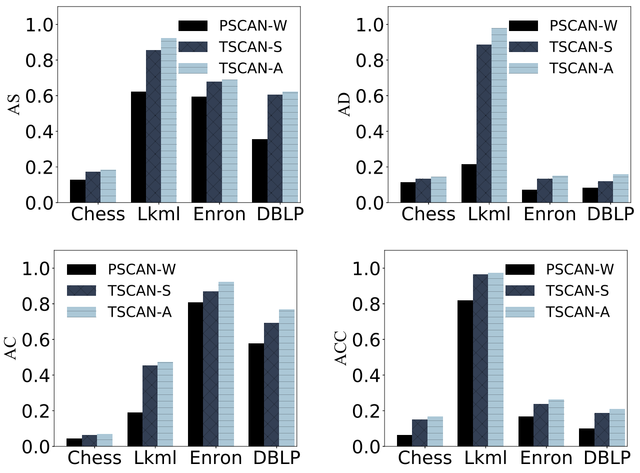


Fig. 9. Effectiveness results of various algorithms

**Exp-2. Effectiveness results of different algorithms.** The four goodness results of different algorithms under the default parameter setting are reported in Fig. 9. Similar results can also be observed using the other parameter settings. As can be seen, TSCAN-A significantly outperforms two baselines in terms of both AS, AD, AC and ACC metrics. Both TSCAN-S and TSCAN-A perform much better than PSCAN-W. The reason could be that PSCAN-W does not consider the stable similarities for clustering, thus the resulting clusters may not truly reflect stable clusters. We can see that the AS, AD and ACC values in Lkml is much larger than those in the other datasets. This is because that the average degree and the maximum degree in Lkml are largest among the four datasets. We also choose the $\tau$th largest value of AS, AD, AC

TABLE 4
Evaluation methods for the communities in temporal graph

| Metric | Formulation | Intuition |
|---|---|---|
| AS | $\sum_{C_i \in \mathcal{C}}[\frac{\lvert\{(u,v,t)\in\mathcal{E}:u\in C_i, v\in C_i\}\rvert}{\lvert\{(u,v,t)\in\mathcal{E}:u\in C_i, v\notin C_i\}\rvert}]/\lvert\mathcal{C}\rvert$ | $\text{avg}_{C_i}$( #temporal edges inside community $C_i$/ #temporal edges outside $C_i$) |
| AD | $\sum_{C_i \in \mathcal{C}}[\frac{\sum_{v_j \in C_i} d_{C_i}(v_j)}{\lvert C_i\rvert}]/\lvert\mathcal{C}\rvert$ | $\text{avg}_{C_i}$( sum of nodes' degrees inside community $C_i$ / #nodes in $C_i$) |
| AC | $\sum_{C_i \in \mathcal{C}} max_{S\subseteq C_i}\phi(S)$ | $\text{max}_{C_i}$( #edges which can split community $C_i$) |
| ACC | $\sum_{C_i \in \mathcal{C}}[\sum_{v_j \in C_i} \frac{\#edge(N(v_j,C_i))}{d_{C_i}(v_j)}]/\lvert\mathcal{C}\rvert$ | $\text{avg}_{C_i}[\text{avg}_{v_i \in C_i}$( #common neighbors of $v_i$/ #temporal degree of $v_i$ inside $C_i$)] |

TABLE 5
Multiple comparisons for effectiveness of various algorithms

| | | | Mean Difference | Std Error | Sig | 95% Confidence Interval | |
|---|---|---|---|---|---|---|---|
| | | | | | | Lower Bound | Upper Bound |
| LSD | PSCAN-W | TSCAN-S | -21.633 | 19.207 | 0.263 | -59.776 | 16.509 |
| | | TSCAN-A | -26.715 | 19.207 | 0.168 | -64.857 | 11.428 |
| | TSCAN-S | PSCAN-W | 21.633 | 19.207 | 0.263 | -16.509 | 59.776 |
| | | TSCAN-A | -5.081 | 19.207 | 0.792 | -43.224 | 33.061 |
| | TSCAN-A | PSCAN-W | 26.715 | 19.207 | 0.168 | -11.428 | 64.857 |
| | | TSCAN-S | 5.081 | 19.207 | 0.792 | -33.061 | 43.224 |
| Bonferroni | PSCAN-W | TSCAN-S | -21.633 | 19.207 | 0.789 | -68.462 | 25.195 |
| | | TSCAN-A | -26.715 | 19.207 | 0.503 | -73.544 | 20.114 |
| | TSCAN-S | PSCAN-W | 21.633 | 19.207 | 0.789 | -25.195 | 68.462 |
| | | TSCAN-A | -5.081 | 19.207 | 1.000 | -51.910 | 41.748 |
| | TSCAN-A | PSCAN-W | 26.715 | 19.207 | 0.503 | -20.114 | 73.544 |
| | | TSCAN-S | 5.081 | 19.207 | 1.000 | -41.748 | 51.910 |
| Dunnett (two sides) | TSCAN-S | PSCAN-W | 21.633 | 19.207 | 0.426 | -21.509 | 64.776 |
| | TSCAN-A | PSCAN-W | 26.715 | 19.207 | 0.283 | -16.428 | 69.858 |

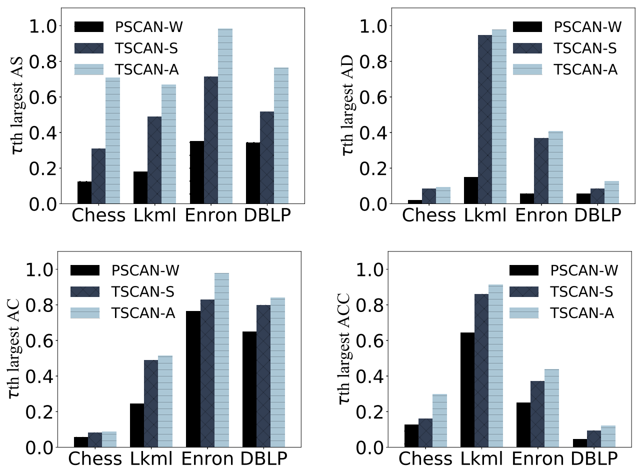
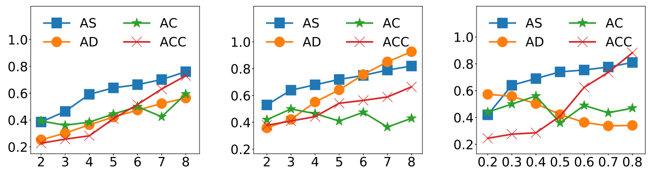


Fig. 10. The $\tau$th largest AS, AD, AC and ACC of various algorithms



(a) vary $\mu$ (DBLP) (b) vary $\tau$ (DBLP) (c) vary $\epsilon$ (DBLP)

Fig. 11. Effectiveness of TSCAN-A with varying parameters on DBLP

and ACC in all the snapshots into comparison. As can be seen in Fig. 10, The results are similar to computing them in the whole de-temporal graph.

Table. 5 shows the multiple comparisons for effectiveness of various algorithms with gathering the goodness values above. In the results of LSD method, we can find that the results of our proposed algorithms are similar, and TSCAN-A is a little better than TSCAN-S. In the results of Bonferroni method, the $p(Sig)$ value of TSCAN-A and TSCAN-S are 1.0, so the effectiveness of those two methods are similar. Considering the upper bound of the 95% confidence interval, TSCAN-A and TSCAN-S are significantly better than PSCAN-W. The results of Dunnett method show that the mean difference is 21.633 with comparing TSCAN-S to PSCAN-W, and 26.715 with comparing TSCAN-A to PSCAN-W. All the results above show that both TSCAN-S, TSCAN-A perform much better than PSCAN-W, and TSCAN-A is the best of all.

**Exp-3. Results with varying parameters.** Here we study how the parameters affect the clustering performance of our algorithm. Fig. 11 shows the results of TSCAN-A with varying parameters on DBLP. Similar results can also be observed on the other datasets.

As can be seen, both AS, AD and ACC of TSCAN-A increase with a growing $\mu$. The reason is that the number of stably similar neighbors of a stable core increases when $\mu$ increases, and therefore the stable clusters tend to be more cohesive as $\mu$ increases. Similarly, we can see that both AS, AD and ACC of TSCAN-A increase as $\tau$ increases. This is because the number of temporal edges associated with the nodes in the stable clusters increases when $\tau$ increases. We can see that the AC values in different parameter settings are not changing in a regular way. The reason is that AC calculates the conductance of the most connected cluster and there can always find one highly connected cluster in every parameter setting.

Interestingly, as shown in Fig. 11(c), ACC increases with an increasing $\epsilon$, but AD decreases when $\epsilon$ increases. The reason could be that when $\epsilon$ increases, the number of the stably similar neighbors of a stable core decreases, and therefore AD decreases. On the other hand, for a large $\epsilon$, the structural similarities between a stable core and its

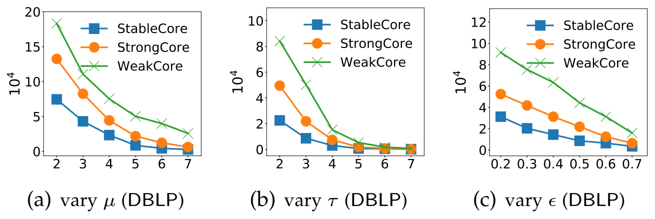


Fig. 12. Number of cores with varying parameters on DBLP

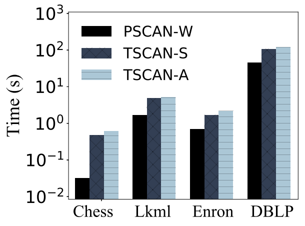


Fig. 14. Efficiency of different algorithms.

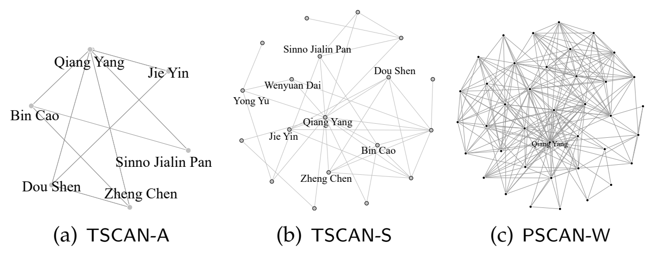


Fig. 13. Case study on DBLP

neighbors increase which give rise to a large ACC value.

**Exp-4. Seeking good parameters settings.** In Exp-3, we find that all the evaluation metrics are better if parameters $\mu$ and $\tau$ increases. But AD decreases when $\epsilon$ increases. So the higher $\mu, \tau$, the better? Fig.12 shows that the number of cores with varying parameters. As we can see, all the numbers decrease sharply while parameters $\mu, \tau, \epsilon$ increase. Although the higher $\mu$ and $\tau$ can get higher evaluation values, we also need a higher number of stable nodes into consideration. In the real applications, all the parameters must be set properly according to the datasets and requirements of the application. For example, under the default parameters settings ($\mu = 5, \tau = 3, \epsilon = 0.5$), one stable core in DBLP represents a researcher co-authored with at least 5 researchers closely ($\sigma > 0.5$, see Eq.1) in more than 3 years.

**Exp-5. Case study.** We conduct a case study on DBLP to compare the effectiveness of different algorithms. Figs. 13(a-c) shows the communities of Prof. Qiang Yang obtained by different algorithms with ($\tau = 3, \mu = 5, \epsilon = 0.3$). Figs. 13(a) shows stable cores in Definition 1 and all the edges in the figure are $(5, 0.3)$-connected edges. As desired, the resulting community of TSCAN-A contains the long-term collaborators of Prof. Qiang Yang, indicating that our algorithm can find stable communities in real-world applications. We look at the research page of Prof. Qiang Yang http://home.cse.ust.hk/~qyang/, and find that exclude Zheng Chen (he is a closely co-author of Yang at MSRA), other researchers connecting to Prof. Yang in Figs. 13(a) are all former Ph.D students of him. In Fig. 13(b), we can see that the community of TSCAN-S not only contains the stable community, but it also contains some other co-authors of Prof. Qiang Yang, which do not stably collaborate with Prof. Qiang Yang. In Figs. 13(c), PSCAN-W performs very bad, as the resulting community involves large numbers of collaborators, so it is hard to find the stable communities in the results of PSCAN-W. These results above further confirm the effectiveness of the proposed algorithm.

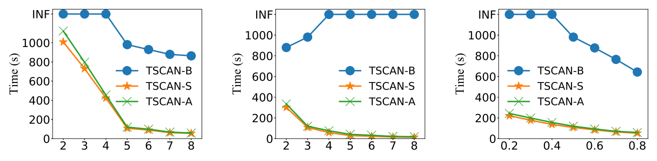
## 4.2 Efficiency Testing

Since TSCAN-B has no pruning techniques, we compare the running time among PSCAN-W, TSCAN-S and TSCAN-A. Also, we test the scalability and performance with varying parameters among TSCAN-B, TSCAN-S and TSCAN-A.

**Exp-6. Efficiency of different algorithms.** Fig. 14 shows the efficiency of different algorithms under the default parameter setting ($\mu = 5, \tau = 3, \epsilon = 0.5$). Similar results can also be observed under the other parameter settings. From Fig. 14, we can see that PSCAN-W is faster than TSCAN-S and TSCAN-A on all datasets, because PSCAN-W relies on much cheaper structural similarity and core computation [18]. Note that although PSCAN-W is very efficient, its clustering quality is much worse than those of TSCAN-S and TSCAN-A. As desired, TSCAN-S is slightly faster than TSCAN-A, because TSCAN-S does not need to invoke the time-consuming frequent pattern mining algorithm to compute the stable cores. Although TSCAN-A is more expensive than the other algorithms, it is still very efficient on large temporal graphs due to the powerful pruning techniques. For example, on DBLP, TSCAN-A only takes around 100 seconds to compute the stable clusters. These results indicate the high efficiency of the proposed algorithm.

**Exp-7. Efficiency of our algorithms with varying parameters.** Here we study how the parameter affects the efficiency of our algorithms. Fig. 15 shows the running time of our algorithms with varying parameters on DBLP. As can be seen, both TSCAN-S and TSCAN-A are much faster than TSCAN-B. The TSCAN-B algorithm is intractable under most parameter settings due to the high complexity of computing the stable cores. These results also demonstrate the high pruning performance of the strong core pruning rule. For both TSCAN-S and TSCAN-A, the running time decreases with increasing $\mu$, $\tau$, and $\epsilon$. This is because the pruning power of TSCAN-S and TSCAN-A increases as $\mu$, $\tau$, and $\epsilon$ increase. We can also see that the running time of TSCAN-S and TSCAN-A are almost the same, it is because that the StrongCore pruning technique has reduced the graph into a quite small size, which makes it easy to perform frequent mining algorithm in reality. The following experiment will demonstrate how powerful the pruning techniques are.

**Exp-8. The number of cores of our algorithms.** Table 6 shows the number of cores obtained by TSCAN-S (strong core) and TSCAN-A (stable core) on DBLP under the default parameter setting. In the columns 2 and 3 of Table 6, the left integer is the number of cores and the right value is the percentage of cores over all the nodes. As can be seen,



(a) vary $\mu$ (DBLP) (b) vary $\tau$ (DBLP) (c) vary $\epsilon$ (DBLP)

Fig. 15. Efficiency of our algorithms with varying parameters

TABLE 6
Number of cores of TSCAN-S and TSCAN-A on DBLP

| | TSCAN-S | | TSCAN-A | |
|---|---|---|---|---|
| Chess | 57 | 0.78% | 39 | 0.53% |
| Lkml | 106 | 0.39% | 49 | 0.18% |
| Enron | 198 | 0.23% | 183 | 0.21% |
| DBLP | 34,044 | 1.97% | 17,192 | 0.99% |

TABLE 7
Memory cost of TSCAN-A for storing similarity scores

| | Graph size | Memory(TSCAN-A) | Memory(PSCAN-W) |
|---|---|---|---|
| Chess | 2.7MB | 0.5MB | 0.4MB |
| Lkml | 20.1MB | 13.4MB | 10.5MB |
| Enron | 53.3MB | 10.3MB | 8.3MB |
| DBLP | 678.5MB | 695.3MB | 475.5MB |

both TSCAN-S and TSCAN-A generate a small number of cores. For example, on DBLP, only 1.97% and 0.99% nodes are strong cores and stable cores respectively. These results confirm the high effectiveness of the strong pruning technique.

**Exp-9. Memory cost of TSCAN-A for storing similarity scores.** Recall that TSCAN-A maintains all the computed similarity scores ($\sigma_i(u, v)$ and $S_\epsilon(u, v)$) in the main memory. Here we evaluate the memory overhead of TSCAN-A for storing all the computed similarity scores. Table 7 reports the results of our algorithm under the default parameter setting. As can be seen, the memory overhead of TSCAN-A is close to the graph size and a little larger than PSCAN-W, confirming that our algorithm is highly space-efficient in practice.

## 5 Related Work

**Temporal Graph Analysis.** Our work is related to the problem of temporal graph analysis, which has attracted much attention in recent years. Yang et al. [24] proposed an algorithm to detect frequent changing components in temporal graph. Huang et al. [25] investigated the minimum spanning tree problem in temporal graphs. Gurukar et al. [26] presented a model to identify the recurring subgraphs that have similar sequence of information flow. Wu et al. [27] proposed an efficient algorithm to answer the reachability and time-based path queries on temporal graphs. Yang et al. [28] studied a problem of finding a set of diversified quasi-cliques from a temporal graph. Wu et al. [29] proposed a temporal $k$-core model based on the counts of temporal edges. Ma et al. [30] investigated a dense subgraph problem in temporal graphs, in which the temporal edges are associated with positive and negative weights. To be best of our acknowledgment, our study is the first to study the problem of mining the stable communities in temporal graph.

**Community Mining.** The community detection problem in static networks has been extensively studied in the literature. Most existing optimization-based algorithms [1], [4], [5], [12] aims to formulate the community mining problem as an optimization problem and then compute an optimal solution with respect to a pre-defined objective function. Most of these algorithm are very costly to handle large graph data. Another line of research for community discovery is to identify dense subgraphs from a graph, such as clique and quasi-clique. Notable techniques for enumerating all maximal cliques and quasi-cliques were proposed in [13] and [14] respectively. Also, there are many clique-relaxed models such as $k$-core and, $k$-truss [6], [7], [15] which can be used to identify dense structures that are similar to cliques.

**Density-based Graph Clustering.** The density-based clustering algorithm for graph data, termed as SCAN, was first proposed in [3]. Shiokawa et al. [19] presented an improved algorithm named SCAN++. It is based on an intuitive idea that it is highly probable that many common neighbors exist between a node and its two-hop-away nodes. Chang et al. [18] proposed a further improved algorithm called PSCAN which can avoid unnecessary similarity computations. Wen et al. [31] proposed an index-based SCAN algorithm. Based on a pre-computed index, the algorithm can derive the structural clusterings with time complexity depending only on the result size. An approximate solution called LinkSCAN was also proposed in [17] to find overlapping clusterings.

**Community in Dynamic Graph.** There are a number of studies for mining communities on dynamic networks [32].

*(i).* Some of them build physical models that the nodes inside one dynamic community will share common statistical characteristics. Those dynamic communities will follow a statistical model (such as Stochastic Block Model), but they may not have certain structural properties (such as degree constraint). Lin et al. [33] proposed FaceNet which is the first probabilistic generative model for analyzing communities and their evolutions. Matias et al. [34] explored statistical properties that combines a stochastic block model for the evolution of the nodes groups through time. Gauvin et al. [35] investigated the use of a latent factor decomposition technique to extract the community-activity structure of temporal networks.

*(ii).* Some other researches track the common properties of nodes or edges inside one dynamic community following graph theory. Nodes or edges in the communities of dynamic graph will have certain structural properties at some timestamps. Chen et al. [36] developed an efficient algorithm for tracking community dynamics by introducing graph representatives. Agarwal et al. [37] studied how to find dense clusters efficiently for dynamic graphs in spite of rapid changes to the microblog streams. Li et al. [15] devised an algorithm which can maintain the $k$-core in the large dynamic graphs. Rossetti et al. [38] proposed an algorithm for tracking the evolution of communities following an online iterative procedure. Falkowski et al. [39] investigated an incremental community miming algorithm based on DBSCAN. DiTursi et al . [40] proposed a filter-and-verify framework for dynamic community detection which prunes the community by several structural constraints. Most community detection studies on dynamic graphs aims

to find and maintain communities that evolve over time. Unlike these studies, our work focuses mainly on detecting stable communities in temporal graphs.

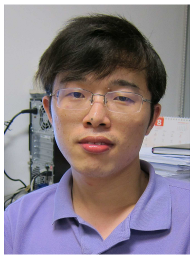
## 6 Conclusion

In this paper, we propose a new density-based graph clustering algorithm to find stable communities in temporal networks. Unlike traditional density-based graph clustering problem, our problem relies on two new concepts called $\epsilon$-stable similarity and $(\mu, \tau, \epsilon)$-stable core. Based on these concepts, we develop a novel algorithm with several powerful pruning techniques to efficiently compute the stable communities in the temporal network. We conduct comprehensive experiments using four real-life datasets to evaluate our algorithm, and the results confirm the effectiveness and efficiency of the proposed algorithm.

## References

[1] M. E. Newman, "Fast algorihtm for detecting community structure in networks," *Physical Review E*, pp. 66–133, 2004.
[2] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *ICDM*, 2012.
[3] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, "Scan: a structural clustering algorithm for networks," in *KDD*, 2007.
[4] H. Avron and L. Horesh, "Community detection using time-dependent personalized pagerank," in *ICML*, 2015.
[5] H. Zhang, T. Zhao, I. King, and M. R. Lyu, "Modeling the homophily effect between links and communities for overlapping community detection," in *IJCAI*, 2016.
[6] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," *SIGMOD*, 2014.
[7] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *ICDE*, 2011.
[8] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *SIGMOD*, 2013.
[9] S. Agreste, P. D. Meo, G. Fiumara, G. Piccione, S.Piccolo, D. Rosaci, G. M. L. Sarne, and A. V. Vasilakos, "An empirical comparison of algorithms to find communities in directed graphs and their application in web data analytics," *IEEE Transactions on Big Data*, vol. 3, no. 3, pp. 289–306, 2017.
[10] P. Vanhems, A. Barrat, C. Cattuto, J.-F. Pinton, N. Khanafer, C. Regis, B. a Kim, B. Comte, and N. Voirin, "Estimating potential infection transmission routes in hospital wards using wearable proximity sensors," *PLoS ONE*, vol. 8, p. e73970, 2013.
[11] J. Fournet and A. Barrat, "Contact patterns among high school students," *PLOS ONE*, vol. 9, p. e107878, 2014.
[12] P. Chopade and J. Zhan, "A framework for community detection in large networks using game-theoretic modeling," *IEEE Transactions on Big Data*, vol. 3, no. 3, pp. 276–288, 2017.
[13] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Trans. Database Syst.*, vol. 36, no. 4, pp. 21:1–21:34, Dec. 2011.
[14] 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*, 2013.
[15] R. H. Li, J. X. Yu, and R. Mao, "Efficient core maintenance in large dynamic graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 10, pp. 2453–2465, 2014.
[16] C. C. Aggarwal and H. Wang, "A survey of clustering algorithms for graph data," in *Managing and Mining Graph Data*, 2010, pp. 275–301.
[17] S. Lim, S. Ryu, S. Kwon, K. Jung, and J. Lee, "Linkscan*: Overlapping community detection using the link-space transformation," in *ICDE*, 2014.
[18] L. Chang, W. Li, X. Lin, L. Qin, and W. Zhang, "pSCAN: Fast and exact structural graph clustering," in *ICDE*, May 2016.
[19] H. Shiokawa, Y. Fujiwara, and M. Onizuka, "SCAN++: efficient algorithm for finding clusters, hubs and outliers on large-scale graphs," *PVLDB*, vol. 8, no. 11, pp. 1178–1189, 2015.
[20] R. Kumar, T. Calders, A. Gionis, and N. Tatti, "Maintaining sliding-window neighborhood profiles in interaction networks," in *ECML PKDD*, 2015.
[21] S. Asur, S. Parthasarathy, and D. Ucar, "An event-based framework for characterizing the evolutionary behavior of interaction graphs," in *KDD*, 2007.
[22] T. Arnoux, L. Tabourier, and M. Latapy, "Combining structural and dynamic information to predict activity in link streams," in *ASONAM*, 2017.
[23] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann, 2011.
[24] Y. Yang, J. X. Yu, H. Gao, J. Pei, and J. Li, "Mining most frequently changing component in evolving graphs," *World Wide Web*, vol. 17, no. 3, pp. 351–376, 2014.
[25] S. Huang, A. W. Fu, and R. Liu, "Minimum spanning trees in temporal graphs," in *SIGMOD*, 2015.
[26] S. Gurukar, S. Ranu, and B. Ravindran, "COMMIT: A scalable approach to mining communication motifs from dynamic networks," in *SIGMOD*, 2015.
[27] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in *ICDE*, 2016.
[28] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. S. Lui, "Diversified temporal subgraph pattern mining," in *KDD*, 2016.
[29] H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu, "Core decomposition in large temporal graphs," in *IEEE International Conference on Big Data*, 2015.
[30] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai, "Fast computation of dense temporal subgraphs," in *ICDE*, 2017.
[31] D. Wen, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Efficient structural graph clustering: An index-based approach," *PVLDB*, vol. 11, no. 3, pp. 243–255, 2017.
[32] G. Rossetti and R. Cazabet, "Community discovery in dynamic networks: A survey," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 35:1–35:37, 2018.
[33] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, "Facetnet: A framework for analyzing communities and their evolutions in dynamic networks," in *WWW*, 2008, pp. 685–694.
[34] C. Matias and V. Miele, "Statistical clustering of temporal networks through a dynamic stochastic block model," *Journal Of The Royal Statistical Society*, vol. 60, no. 4, pp. 12–31, 2015.
[35] L. Gauvin, A. Panisson, and C. Cattuto, "Detecting the community structure and activity patterns of temporal networks: A non-negative tensor factorization approach," *PLOS ONE*, vol. 9, pp. 1–13, 01 2014.
[36] Z. Chen, K. A. Wilson, Y. Jin, W. Hendrix, and N. F. Samatova, "Detecting and tracking community dynamics in evolutionary networks," in *ICDMW*, 2010, pp. 318–327.
[37] M. K. Agarwal, K. Ramamritham, and M. Bhide, "Real time discovery of dense clusters in highly dynamic graphs: Identifying real world events in highly dynamic environments," *PVLDB*, vol. 5, no. 10, 2012.
[38] G. Rossetti, L. Pappalardo, D. Pedreschi, and F. Giannotti, "Tiles: an online algorithm for community discovery in dynamic social networks," *Machine Learning*, vol. 106, no. 8, pp. 1213–1241, 2017.
[39] T. Falkowski, A. Barth, and M. Spiliopoulou, "Studying community dynamics with an incremental graph mining algorithm," in *AMCIS 2008*, vol. 5, 01 2008, p. 29.
[40] D. J. DiTursi, G. Ghosh, and P. Bogdanov, "Local community detection in dynamic networks," in *ICDM*, Nov 2017, pp. 847–852.

**Hongchao Qin** is currently a Ph.D. Candidate in Northeastern University, China. He received the B.S. degree in mathematics and M.E. degree in computer science from Northeastern University in 2013 and 2015, respectively. His current research interests include social network analysis and data-driven graph mining.



**Rong-Hua Li** received the Ph.D. degree from the Chinese University of Hong Kong in 2013. He is currently an Associate Professor at Beijing Institute of Technology, Beijing, China. His research interests include graph data management and mining, social network analysis, graph computation systems, and graph-based machine learning.

**Guoren Wang** received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, China, in 1988, 1991 and 1996, respectively. Currently, he is a Professor in the Department of Computer Science, Beijing Institute of Technology, Beijing, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management. He has published more than 100 research papers.

**Xin Huang** is currently an assistant professor in the department of computer science at the Hong Kong Baptist University. He received his BEng degree in computer science from the Xiamen University in 2010, and PhD degree in systems engineering and engineer in management from the Chinese University of Hong Kong in 2014. His research interests mainly focus on graph data management and mining.

**Ye Yuan** received the BS, MS, and PhD degrees in computer science from Northeastern University, in 2004, 2007, and 2011, respectively. He is now a professor in the Department of Computer Science, Northeastern University, China. His research interests include graph databases, probabilistic databases, and social network analysis.

**Jeffery Xu Yu** received the B.E., M.E., and Ph.D. degrees in computer science from the University of Tsukuba, Japan, in 1985, 1987, and 1990, respectively. He has held teaching positions at the Institute of Information Sciences and Electronics, University of Tsukuba, and at the Department of Computer Science, Australian National University, Australia. Currently, he is a Professor in the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong, Hong Kong. His current research interests include graph database, graph mining, keyword search in relational databases, and social network analysis.