Maximum (L, K)-Lasting Cores in Temporal Social Networks

Wei-Chun Hung and Chih-Ying Tseng(✉)

Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan

Abstract. Extracting dense structures in a social network is a fundamental task in graph mining and can find many real-world applications. The *temporal social network* augments the conventional social network with the temporal dimension, and extracting dense structures enables us to understand the period of time for which the dense structures exist. In this paper, we propose the new notion of (L, K)-*lasting core*, which is a densely connected subgraph lasting for a sufficiently long period of time in the temporal social network. We propose a polynomial-time algorithm to obtain the maximum (L, K)-lasting core with various processing strategies to boost the efficiency. We conduct extensive experiments on multiple datasets to validate the effectiveness and efficiency of the proposed approach. The experimental results show that our proposed approaches outperform the other baseline approaches in terms of solution quality and efficiency.

Keywords: Densest subgraph · Temporal social networks · Cores

1 Introduction

Extracting dense subgraphs has been actively studied in recent years. It is a basic and important work in graph analysis. There are many different definitions of dense subgroup, e.g., average degree [11,21,30], k-truss [22,40], clique [3,4,15], quasi-clique [1,27], k-core [12,24,32]. All of those different definitions refer to the same thing that vertices in the group are highly connected. Dense subgroup of different definitions have different ways to extract, such as linear programming [23,26], core decomposition [8,42], etc. It has a lot of application in practice, e.g., biological module discovery [20], story identification [2] and community detection [6,10].

Most of the previous work focus on the dense subgraph on single-layer graph which exists for just one time. However, they can not apply to multilayer graph. There were also many work finding dense subgraph on multilayer network. Some of them consider layers are different type of information [13,46]. Some of them consider that different layers represent different time [25,29] and we call them temporal networks. Each edge in temporal networks is associated with time. Densely connected vertices in a temporal network may correspond to a community. For example, in a collaboration network, dense subgroup may represent a

C. S. Jensen et al. (Eds.): DASFAA 2021 Workshops, LNCS 12680, pp. 336–352, 2021.
https://doi.org/10.1007/978-3-030-73216-5_23

research team or some researchers in the same domain publishing papers together continuously.

In this paper, we study the problem of long-lasting group in the temporal network. We not only consider the cohesiveness but also the time the group lasts. We adopt the definition of k-core, which is a group in which each member has at least k other friends also in the group. We aim to find the largest group in the temporal network with given lasting-time constraint and connectivity constraint.

To achieve above goal, we present a model, called (L, K)-lasting core, based on the well-known concept of k-core. Our model can preserve a k-core lasting for L time. For its application scenarios, we can leverage this model to find a team of researchers publishing paper every year in the same field, or a social group whose member interact with each other very frequently. We formally define the problem and propose effective algorithm to deal with it. We conduct extensive experiments to evaluate our model and study the impact of algorithm variations. In summary, the contributions of this paper are summarized as follows:

- We propose the notion of (L, K)-lasting core to integrate the social cohesiveness with the temporal dimension to identify important groups in temporal social networks.
- We develop effective algorithms and techniques to extract (L, K)-lasting cores.
- We conduct extensive experiments to evaluate the performance of our algorithms.

The rest of this paper is organized as follows. After reviewing related work on dense subgraph and temporal network in Sect. 2, we provide the notation and formulate our problem in Sect. 3. The details of the proposed algorithms are described in Sect. 4. We provide the experimental results in Sect. 5. We conclude this paper in Sect. 7.

2 Related Work

2.1 Dense Subgraph

Our work is related to the problem of extracting dense subgraphs, which has been actively studied for years. There are many measurements of dense subgraphs, including average degree [11], k-core [12], clique [4]. For example, Epasto et al. [11] proposed the method of maintaining a densest subgraph and quickly updating while an edge insertion or deletion. Sariyuce et al. [32] also studied on dynamic graph but what they maintained is the k-core decomposition. Bomze et al. [4] proposed several methods to deal with clique problem. In this paper, we propose the notion of (L, K)-lasting core, which extends the idea of k-core.

2.2 Multilayer Network

We study the problem on temporal networks, which is a special type of multilayer graph and its layers represent continuous time. There are many related work of multilayer graphs [13,25,29,46,47]. Zhang et al. [46] studied the problem on two

layer graph which is a special case of multilayer graph. One layer is friendship of entities and the other is similarity between entities. Rozenshtein et al. [29] searched several groups of vertices with different time interval to maximize the sum of group density. Galimberti et al. [13] propose a method to find a subgroup that maximizes the minimum density of selected layers. Zhu et al. [47] aimed to find several multilayer cores that cover the largest number of vertices. Li et al. [25] deals with temporal network. They aimed to find dense subgraph with three constraints, θ, τ, and k called (θ, τ)-persistent k-core. Note that τ is larger than θ. The union graph of each θ length time interval in a τ length interval contains a k-core. We give an example of this work. In 1, we give $\theta = 2$, $\tau = 3$ and $k = 3$. For $\tau = 3$, we first choose G_1, G_2 and G_3 and then intersect two consistent snapshots in them for $\theta = 2$, e.g., G_1, G_2 and G_2, G_3. Then we find k-core of each intersection graph. The k-core of G_1, G_2 is a, b, c, d and the one of G_2, G_3 is a, b, c, d, e, f. And the (2, 3)-persistent 3-core of this interval is a, b, c, d. The work we mentioned can't apply to our problem which we want to find a group lasting for a consistent time. We formulate the problem in the next section.

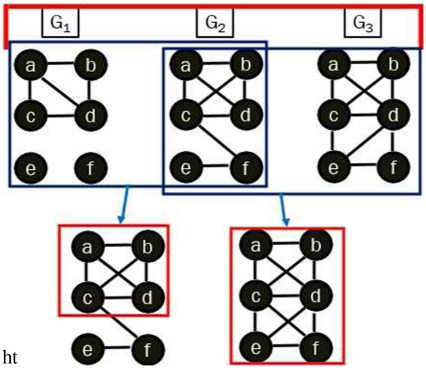


Fig. 1. An example of (θ, τ)-persistent k-core.

2.3 Community Search

Community search in social networks is an active research field [5,9,18,19,31, 41,43]. These works study various community search problems, including the enumeration of k-vertex connected components [41], extracting dense subgraphs, i.e., small-diameter k-plexes [9], identifying the maximum clique in sparse social networks [5], and proposing the UCF-Index to extract (k, η)-core in linear time for uncertain graphs [43]. In addition to finding dense communities in social networks, recent works also discuss finding sparse anti-communities in social networks [16,34,36,37], which has a wide spectrum of application scenarios.

2.4 Dense Subgraphs in Heterogeneous Social Networks

Extracting dense subgraphs in heterogeneous social networks have attracted research attentions [7,14,17,25,33,35,39,45,47]. These works propose new ideas

and enable many new applications, such as enumerating the spatial cliques in the two-dimensional space for dense subgraph extraction [45]. Moreover, a set of socio-spatial group queries aim at identifying the socially-dense groups and the corresponding meeting points [14,38,44]. In addition to social and spatial relations, SDSQ is proposed for live multi-streaming scenarios in social networks that considers both the social tightness and preference of the users, as well as the diversity of multi-streaming channels [33].

3 Problem Definition

We are given a temporal network $G = (V, E)$, where V is a set of vertices, and $E = \{(u, v, t)\}$, such that $u, v \in V$, timestamp $t \in \mathbb{N}$, indicating that edge (u, v) exists at time t. Given $t \in \mathbb{N}$, $E_t = \{(u, v, t)\}$, which contains edges in timestamp t, we call each graph associated with certain time is a snapshot: Let $G_t = (V, E_t)$, $\forall t \in \mathbb{N}$. Given a subset $C \subseteq V$, edges induced by C at timestamp t is denoted $E_t(C) = \{(u, v, t)\}$ for u, $v \in C$. Then the degree of vertex $u \in C$ at time t is denoted $d_t(u, C) = |\{u \in C | (u, v, t) \in E_t(C)\}|$.

Definition 1 *(L-lasting time).* *L-lasting time means a time sequence which has L continuous snapshot, e.g.,* $[0, 1, ..., L-1]$.

Definition 2 *((L, K)-lasting core).* *The (L, K)-lasting core of a temporal network* $G = (V, E)$ *is a non-empty set of vertices* $C_{(L,K)} \subseteq V$, *such that* $\forall u \in C_{(L,K)}$ *and* $d_t(u, C(L, K)) \geq K$, $\forall t \in [t_0, t_1, ..., t_{L-1}]$, $K \in \mathbb{N}^+$.

In other words, Definition 2 is saying that a (L, K)-lasting core is a K-core with L-lasting time. A maximum (L, K)-lasting core means it is a (L, K)-lasting core which has the most vertices. Then we formulate the first problem in this work which is to search a maximum (L, K)-lasting core.

Problem 1 (Maximum (L, K)-lasting core). Given a temporal network $G = (V, E)$, two parameters L and K, find the maximum (L, K)-lasting core of G.

We give an example of (L, K)-lasting core. In Fig. 2, we have a temporal network of 4 snapshots. Given $L = 2$ and $K = 3$, we can observe that a, b, c, d circled by red line is the maximum (2, 3)-lasting core in this temporal network. In the next section, we will propose basic algorithm for Problem 1 and how to speed up using advanced techniques.

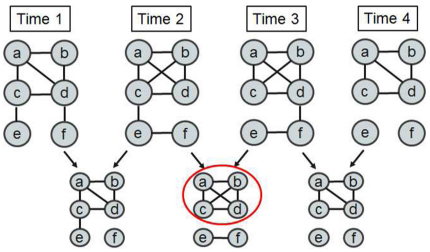


Fig. 2. An example of (L, K)-lasting core.

4 Algorithm Design

In this section, we introduce our proposed approaches and techniques to extract (L, K)-lasting core in details. First, we give a naive idea and a basic algorithm that is also the baseline in experiments. And then, we provide some simple but powerful techniques to speed up online search time. Finally, we explore the order of snapshots permutation to minimize the intersection cost.

4.1 Naive Algorithm

The naive algorithm of Problem 1 is detailed in Algorithm 1. First, we use the concept of sliding window and let L be the length of the sliding window, i.e., it contains L snapshot. The basic algorithm consists of two steps. The first step is to slide the window by changing starting time t. Let C be G_t, the snapshot of time t. Then we intersect C with the rest of $L-1$ snapshot. The second step is to find C_k, the K-core of the intersection result C. If the size of C_k is larger than the current maximum size of (L, K)-lasting core, C_k become the current maximum (L, K)-lasting core.

Algorithm 1. Naive algorithm

Input: A temporal network $G = (V, E)$, L and K
Output: The maximum (L, K)-lasting core $C_{(L,K)}$ of G

```
1: C_(L,K) ← ∅;
2: forall t ∈ [0, 1,..., t_max-L] do
3:     C ← G_t;
4:     forall i ∈ [t+1, t+2,..., t+L-1] do
5:         C ← C ∩ G_i;
6:     C_k ← find_kcore(C);
7:     if |C_k| > |C_(L,K)| then
8:         C_(L,K) ← C_k;
9: return C_(L,K);
```

Here, we omit the details of the K-core algorithm. It is to recursively remove the vertex with degree smaller than K and check its neighbors' degrees until no vertex has degree smaller than K in the subgroup C, i.e., $d_t(u, C) \geq K$. Finally, in line 9, we output the maximum (L, K)-lasting core.

4.2 Temporal Core Finding-Basic (TCFB)

In the previous subsection, the naive approach performs many redundant intersections. To tackle this issue, our idea is to reuse some parts of intersection results. Once we reuse them, we are able reduce the number intersections and improve the efficiency. For example, the current time sequence being processed is $[0, 1, 2, 3, 4]$ and the next time sequence is $[1, 2, 3, 4, 5]$. We can observe that

the timestamp $[1, 2, 3, 4]$ is repeated. If we first do intersection of $[1, 2, 3, 4]$ which means G_1, G_2, G_3 and G_4 and have the result C, then the next action for current time sequence is to intersect C and G_0 and for next time sequence is to intersect C and G_5. Thus we can reduce one time of intersection of G_1, G_2, G_3 and G_4. The detailed description is outlined in Algorithm 2.

4.3 Min-Degree Pruning (MDP)

In this subsection, we focus on reducing vertices. Given a time sequence, each vertex has a degree on different snapshot. After the process of intersection and k-core, we get the result subgraph C, the degree of each vertex in C will less than or equal to the vertex's smallest degree of all snapshots of the time sequence. In other words, given a time sequence T, $d(u, C) \leq min_t^{t \in T} d(u, G_t)$. Based on this, we can remove the vertex u which $min_t^{t \in T} d(u, G_t) < K$ before the process. Equipped with this technique, we may traverse less vertices for each time sequence to reduce the cost. Here we just eliminate the vertex which does not satisfy the K constraint before the process, then we can further execute Algorithm 2.

Algorithm 2. Temporal Core Finding-Basic (TCFB)

Input: A temporal network $G = (V, E)$, L and K
Output: The maximum (L, K)-lasting core $C_{(L,K)}$ of G

```
1: C_(L,K) ← ∅;
2: C ← ∅;
3: forall t ∈ [0, 1,..., t_max-L] do
4:    r ← t mod 2
5:    if r = 0 then
6:       C ← G_{t+1};
7:       forall i ∈ [t+2, t+3,..., t+L-1] do
8:          C ← C ∩ G_i;
9:    index ← i + (L − 1) ∗ r
10:   C ← C ∩ G_index
11:   C_k ← find_kcore(C);
12:   if |C_k| > |C_(L,K)| then
13:      C_(L,K) ← C_k;
14: return C_(L,K);
```

4.4 Reordering for Intersection Minimization (RIM)

Now if we have a time sequence, we intersect snapshots chronologically in Algorithm 2. We observed that if we disrupt the order of original time sequence, the result of intersection graph remain the same, but the number of intersection times will be different. Here we can formulate a Subproblem. If we know

the order having the least number of intersection times, then we have minimum cost. The subproblem is formulated as follows.

Problem 2 (Minimum times of intersection). Given a time sequence of L snapshot on G, we would like to find a permutation of given time sequence that minimize the sum of edge numbers of the intersection graph $[I_0, I_1, ..., I_{L-2}]$, such that $I_0 = G_0$, $I_1 = \text{intersection}(I_0, G_1)$, $I_2 = \text{intersection}(I_1, G_2)$,..., $I_{L-2} = \text{intersection}(I_{L-2}, G_{L-1})$.

Clearly, if we can solve Subproblem 1, then we can solve Problem 1 more efficiently. A straightforward method to find optimal solution of Subproblem 1 is to enumerate all the permutation of given time sequence and find the best one. However, it is time-consuming to search in L factorial number of permutation when L grows. We propose a method, which choose snapshot greedily based on the previous snapshot. The standard of choosing next snapshot is by edge difference of two snapshots. Edge difference means the number of edges of previous snapshot that do not exist in the next snapshot. The detailed description is outlined in Algorithm 3. We compute edge difference offline. In line 4, edgediff(G_i,G_j) means the number of edges of G_i that do not exist in G_j. It should be noted that edgediff(G_i,G_j) may be not same as edgediff(G_j,G_i). After the computation, we get a map $Diff$ giving us information of edge difference of each snapshot pair. We then apply it on Algorithm 2 which is Algorithm 3. In Figs. 3, 3(a) is an example of snapshots. We can compute edge difference: $Diff[(1,2)] = 3$, $Diff[(1,3)] = 2$, $Diff[(1,4)] = 1$, $Diff[(2,3)] = 4$, $Diff[(2,4)] = 1$. If we are using greedy order, and we first choose snapshot 1, and next we choose snapshot 2 because $Diff[(1,2)] = 3$ is the largest. Then we choose snapshot 3. Now we see Fig. 3(b) to compute the cost: snapshot 1 has 4 edges; Intersection of 1, 2 has 1 edge; Intersection of 1, 2, 3 has 1 edge and Intersection of 1, 2, 3, 4 has 0 edge. Cost will be $4 + 1 + 1$. Finally, we have an advanced algorithm called Temporal Core Finding (TCF) which applies MDP and RIM on TCFB.

Algorithm 3. Edge Difference

Input: A temporal network $G = (V, E)$
Output: Edge difference of each pair of snapshots
1: **forall** $i \in [0, 1, ..., t_{max}]$ **do**
2: **forall** $j \in [0, 1, ..., t_{max}]$ **do**
3: **if** $i \neq j$ **then**
4: $Diff[(i, j)]$ = edgediff(G_i,G_j);
5: **return** $Diff$;

Algorithm 4. RIM

Input: A temporal network $G = (V, E)$, L and K
Output: The maximum (L, K)-lasting core $C_{(L,K)}$ of G;

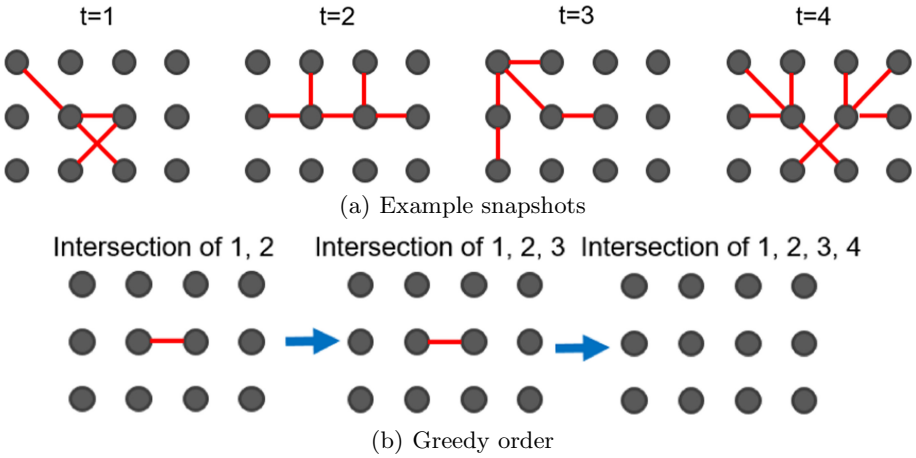
$C_{(L,K)} \leftarrow \emptyset$;
forall $t \in [0, 1, ..., t_{max}\text{-}L]$ **do**
  $C \leftarrow G_t$;
  $h \leftarrow t$;
  $S \leftarrow set(t{+}1, t{+}2, ..., t{+}L\text{-}1)$;
  forall $i \in [1, 2, ..., L\text{-}1]$ **do**
    $h \leftarrow argmax_{s \in S} Diff(h, s)$;
    $S \leftarrow S \setminus s$;
    $C \leftarrow C \cap G_h$;
  $C_k \leftarrow$ find_kcore(C);
  if $|C_k| > |C_{(L,K)}|$ **then**
    $C_{(L,K)} \leftarrow C_k$;
return $C_{(L,K)}$;

4.5 Time Complexity and Optimality

Then we analyze the time complexity of naive algorithm that we mentioned in Sect. 4.1. We use a sliding window to traverse given time. Assume the length of given time is N, then we need to process $N - L + 1$ sliding window. For each sliding window, we have L snapshots and intersect graphs by iterating edges of graph of starting time, so the time complexity of this part is O(EL). The next part is to find k-core of intersection graph. We recursively remove edges and vertices, thus the time complexity is O($V{+}E$). Then we can derive the time complexity of naive algorithm which is O(NLE^2).

Next, we analyze the time complexity of TCF. The number of sliding window is still $N - L + 1$. We can see the part of reusing intersection of overlapped snapshots can decrease intersection times by about 2 times, which do not affect time complexity. The part of MDP remove vertices which can not be in the solution, but it may not remove any vertices in the worst case. The part of RIM change the intersection order of snapshots, but it has no effect on time complexity. Therefore, the time complexity of intersection part is still O(EL). The part of finding k-core didn't change. Then we can derive the time complexity of TCF which is still O(NLE^2). Though the time complexity of TCF doesn't change, the better performance can be see in later Sect. 5.

Here we discuss the optimality of basic algorithm and TCF. The basic algorithm slide a window to search in each interval, do intersection, and find k-core. Obviously, the basic algorithm can find optimal solution. Then, in TCF, the part of reusing intersection of overlapped snapshots just reduce intersection cost, it process the same thing. MDP part eliminates the vertex not in the solution which doesn't affect the result. RIM part change the order of snapshots, but the intersection graph of them is same as non-ordering one's. Thus we can observe that both naive algorithm and TCF can find optimal solution.



(a) Example snapshots

(b) Greedy order

Fig. 3. Example of greedy order and optimal order.

5 Experiments

In this section, we present experimental results and performance comparisons of our methods on different datasets.

Datasets. We use 3 real-world datasets recording interactions with temporal information. Each dataset is with a window size which defining how much continuous time each snapshot contains. If multiple interactions between two vertices appear in the same window, they just counted as one interaction. The characteristics of the datasets are reported in Table 1.

Last.fm records the co-listening history of its streaming platform. DBLP is the co-authorship network of the authors of scientific papers from DBLP computer science bibliography. Youtube [28] is a video-sharing web site that includes a social network and we pick two window size for this dataset. Synthetic datasets are all generated by Barabási–Albert preferential attachment model. Synthetic2000 is generated as 2000 vertices with 100 snapshots and the average degree of each snapshot ranges from 50 to 700. Synthetic5000 is generated as 5000 vertices and the average degree ranges from 80 to 900. Synthetic10W is generated as 100000 vertices and the average degree ranges from 60 to 300.

Method. We compare our approach (TCFB, TCF and two techniques) with brute force finding order of least intersection times (BF), greedy density method (Jethava) in [23] and maximal-span-cores algorithm (Galimberti) in [12]. Techniques are MDP and RIM. TCFB is in Sect. 4.2 and TCF is our best method applying MDP and RIM.

Implementation. All methods are implemented in C++. The experiments run on a machine equipped with Intel CPU at 3.7 GHz and 64 GM RAM.

Table 1. Datasets.

Dataset	$\|V\|$	$\|E\|$	$\|T\|$	Window size	Type
Last.fm	992	4M	77	21 days	Co-listening
DBLP	1M	11M	78	1 year	Co-authorthip
Youtube_1	3M	12M	78	1 day	Friendship
Youtube_2	3M	11M	78	2 day	Friendship
synthetic2K	2K	39M	100	X	Synthetic
synthetic5K	5K	166M	100	X	Synthetic
synthetic10W	10W	636M	50	X	Synthetic

5.1 Comparison of BF and TCF

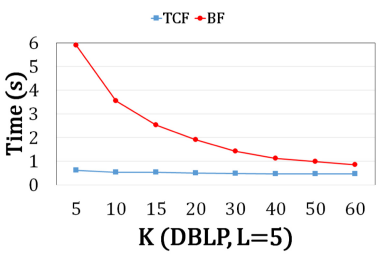
First, we try to find optimal solution of Subproblem 1, finding the permutation of minimum number of times of intersection. We test all permutation of snapshots for each given time sequence. The running time is proportional to L factorial. Then we conduct experiments with small L on DBLP dataset to compare TCF and BF. In Fig. 4(a), L remains the same, and when K becomes larger, BF's running time decreases because the part of finding k-core removes most of the vertices that do not satisfy K constraint and TCF's changes not much because most of the vertices are removed before processing and intersection time dominates the running time. In Fig. 4(b), we can see BF's running time grows with L as expected. Figures 4(c) and (d) show that TCF has the smallest number of times of intersection in all conditions.

6 Methods with Different Techniques

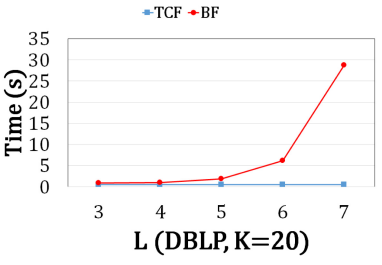
We compare our methods applying different techniques in this subsection. In Fig. 5(a), the running time of intersection part decreases when K grows, but the part of finding k-core dominates the running time, thus the total running time seems to remain the same. In Fig. 5(b), all the total running time of four methods decreases when L grows because more vertices are removed before processing and more edges do not satisfy the L constraint. Therefore, both running time of intersection part and finding k-core part decrease. Moreover, Figs. 5(c) indicates that TCFB+RIM's intersection times is smaller than TCFB's because reordering snapshots has effect on reducing the cost. They all remain the same because L is fixed. Both TCFB+MDP's and TCF's intersection times decrease when K grows because more vertices that not satisfy constraint K are removed.

6.1 Synthetic Datasets

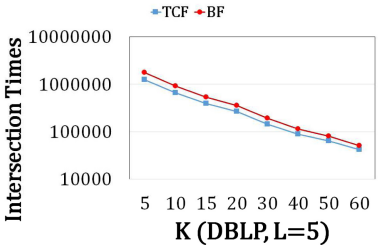
Then we conduct experiments on small and big synthetic datasets. For small synthetic datasets, in Figs. 6 and 7, we can observe that TCFB+MDP's and TCF's time and intersection times decrease when K is larger than 60 in Figs. 6(a)



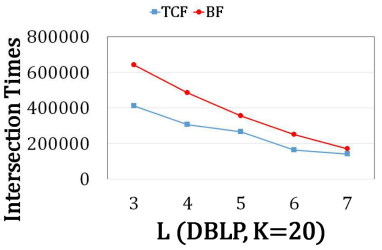
(a) Time vs K



(b) Time vs L

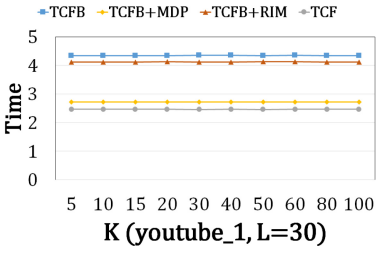


(c) Number of times of intersection vs K

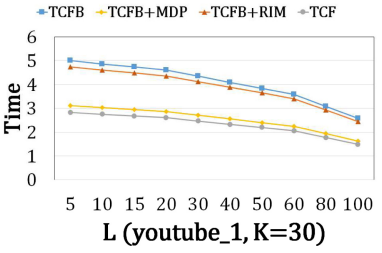


(d) Number of times of intersection vs L

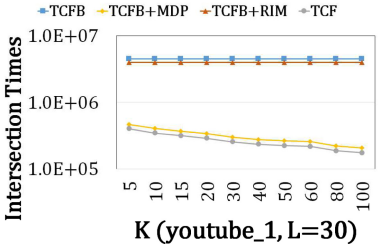
Fig. 4. Comparison of baseline and exhaustive - DBLP.



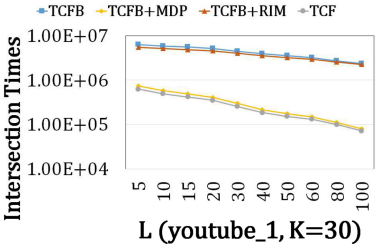
(a) Time vs K



(b) Time vs L

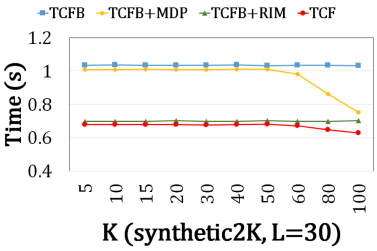


(c) Number of times of intersection vs K

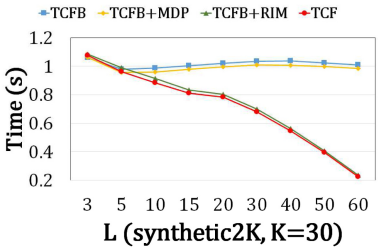


(d) Number of times of intersection vs L

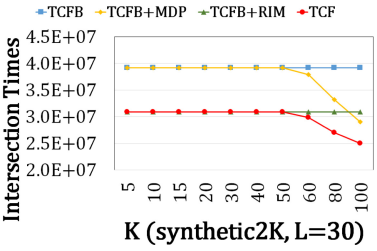
Fig. 5. Methods with different techniques - Youtube #1.



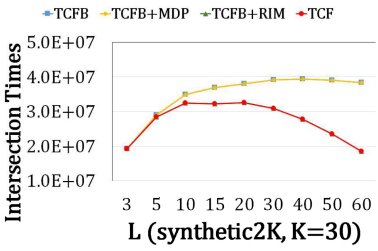
(a) Time vs L - synthetic2000



(b) Time vs L - synthetic2000

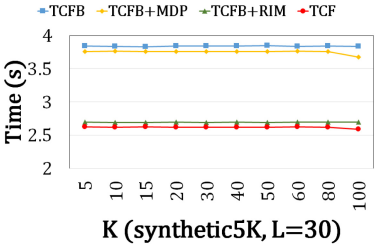


(c) Number of times of intersection vs K

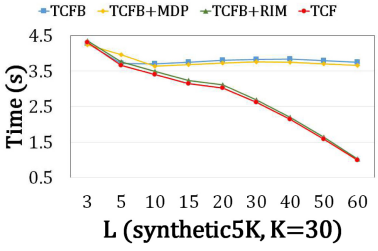


(d) Number of times of intersection vs L

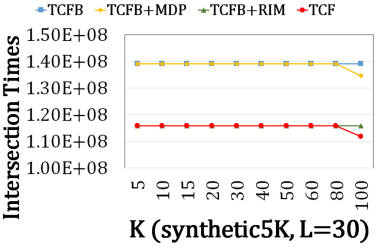
Fig. 6. Methods with different techniques - synthetic datasets #1.



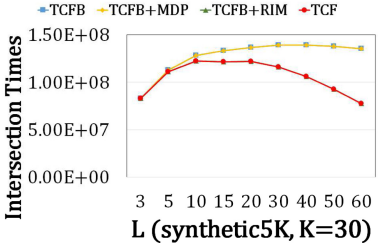
(a) Time vs L - synthetic5000



(b) Time vs L - synthetic5000

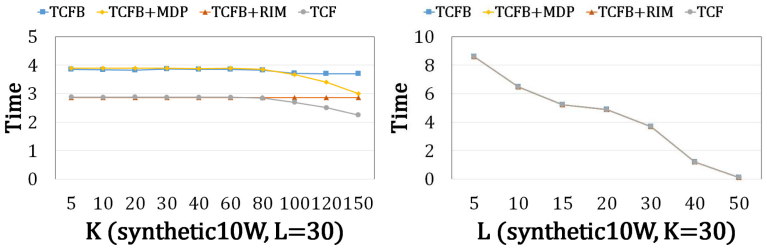


(c) Number of times of intersection vs K



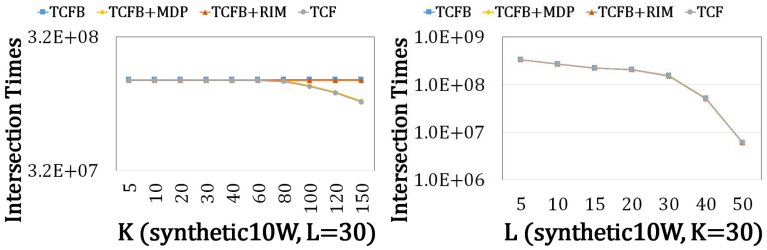
(d) Number of times of intersection vs L

Fig. 7. Methods with different techniques - synthetic datasets #2.



(a) Number of times of intersection vs K

(b) Number of times of intersection vs L



(c) Number of times of intersection vs K

(d) Number of times of intersection vs L

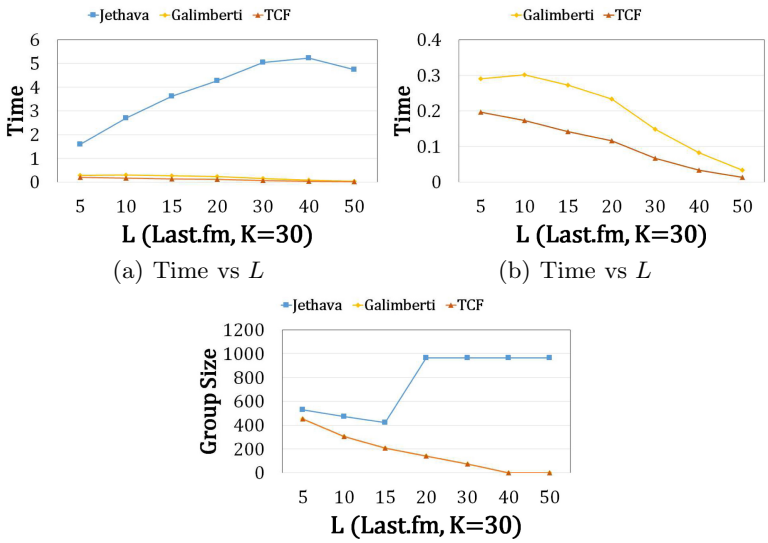
Fig. 8. Methods with different techniques - big synthetic datasets.

and (c) and when K is larger than 80 in Figs. 7(a) and (c). That is because synthetic2K's vertex smallest degree is 60 and synthetic5K's is 80 and MDP only can remove vertices when K is larger than each dataset's smallest degree. Other methods do not apply MDP so their running time and intersection times remain almost the same. In Figs. 6(b) and 7(b), TCFB and TCFB+MDP are lines almost the same, and TCFB+RIM and TCF are lines almost the same. We can see methods with RIM get large improvement and MDP have no effect on this condition because of fixed K 30 is too small to remove any vertices. Figures 6(d) and 7(d) have same condition that methods with RIM are lines almost the same and methods without RIM are other lines almost the same. They first ascend and then decline, it is because RIM is less effective when L is small. On the other hand, when L is large, RIM reduces a lot of cost. For big synthetic dataset, in Figs. 8(a) and (c), MDP only decreases intersection times when K is large enough. In Figs. 8(b) and (d), RIM is less effective because this dataset is too sparse to make great change on intersection times, so all lines seems almost the same.

6.2 Comparison of Other Work

In this subsection, we implement greedy density method (Jethava) in [23] and maximal-span-cores algorithm (Galimberti) in [12] and compare with our methods on Last.fm dataset. In Fig. 9(a), we can see Jethava takes more time than

Galimberti and TCF. For Galimberti and TCF, we can see more clear in Fig. 9(b) that Galimberti takes more time than TCF. Then we turn to judge the solution each method found. In Fig. 9(c), we can see Jethava's solution is always the biggest size and Galimberti's solution is same as TCF which is the optimal solution. Jethava find the group with largest density according to L, which do not consider K. And its solution is too big to find meaningful or interesting pattern. Although Galimberti finds optimal solution, TCF is more effective which has less running time. Therefore, our method TCF has better efficiency and effectiveness.



(a) Time vs L (b) Time vs L

(c) Group Size vs L

Fig. 9. Comparison of other work.

Finally, we make a conclusion. MDP can reduce graph size when K becomes larger thus reduce number of times of intersection and then running time. When L becomes larger, RIM can intensively strengthen the performance. In other words, our methods can give good speedup in running time.

7 Conclusions

In this paper, we introduced a model called (L, K)-lasting core to detect the lasting group in temporal networks. We proposed efficient algorithms and applied advance techniques to solve this problem. Experiments in different datasets show us the efficiency and scalability. We can find interesting group by using our algorithms. In future work, we can extend our algorithms to temporal hypergraphs, which edges in the graph are arbitrary sets of nodes.

References

1. Abello, J., Resende, M.G.C., Sudarsky, S.: Massive quasi-clique detection. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 598–612. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45995-2_51
2. Angel, A., Sarkas, N., Koudas, N., Srivastava, D.: Dense subgraph maintenance under streaming edge weight updates for real-time story identification. Proc. VLDB Endowment **5**(6), 574–585 (2012)
3. Balasundaram, B., Butenko, S., Hicks, I.V.: Clique relaxations in social network analysis: the maximum k-plex problem. Oper. Res. **59**(1), 133–142 (2011)
4. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: Handbook of Combinatorial Optimization, pp. 1–74. Springer (1999). https://doi.org/10.1007/978-1-4757-3023-4_1
5. Chang, L.: Efficient maximum clique computation over large sparse graphs. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 529–538. ACM (2019)
6. Chen, J., Saad, Y.: Dense subgraph extraction with application to community detection. IEEE Trans. Knowl. Data Eng. **24**(7), 1216–1230 (2010)
7. Chen, Y.-L., Yang, D.-N., Shen, C.-Y., Lee, W.-C., Chen, M.-S.: On efficient processing of group and subsequent queries for social activity planning. IEEE Trans. Knowl. Data Eng. **31**(12), 2364–2378 (2018)
8. Cheng, J., Ke, Y., Chu, S., Özsu, M.T.: Efficient core decomposition in massive networks. In: 2011 IEEE 27th International Conference on Data Engineering, pp. 51–62. IEEE (2011)
9. Conte, A., De Matteis, T., De Sensi, D., Grossi, R., Marino, A., Versari, L.: D2k: scalable community detection in massive networks via small-diameter k-plexes. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1272–1281. ACM (2018)
10. Dourisboure, Y., Geraci, F., Pellegrini, M.: Extraction and classification of dense communities in the web. In: Proceedings of the 16th International Conference on World Wide Web, pp. 461–470. ACM (2007)
11. Epasto, A., Lattanzi, S., Sozio, M.: Efficient densest subgraph computation in evolving graphs. In: Proceedings of the 24th International Conference on World Wide Web, pp. 300–310. International World Wide Web Conferences Steering Committee (2015)
12. Galimberti, E., Barrat, A., Bonchi, F., Cattuto, C., Gullo, F.: Mining (maximal) span-cores from temporal networks. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pp. 107–116. ACM (2018)
13. Galimberti, E., Bonchi, F., Gullo, F.: Core decomposition and densest subgraph in multilayer networks. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 1807–1816. ACM (2017)
14. Ghosh, B., Ali, M.E., Choudhury, F.M., Apon, S.H., Sellis, T., Li, J.: The flexible socio spatial group queries. Proc. VLDB Endowment **12**(2), 99–111 (2018)
15. Himmel, A.-S., Molter, H., Niedermeier, R., Sorge, M.: Enumerating maximal cliques in temporal graphs. In: Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, pp. 337–344. IEEE Press (2016)
16. Hsu, B.-Y., Lan, Y.-F., Shen, C.-Y.: On automatic formation of effective therapy groups in social networks. IEEE Trans. Comput. Soc. Syst. **5**(3), 713–726 (2018)

17. Hsu, B.-Y., Shen, C.-Y.: On extracting social-aware diversity-optimized groups in social networks. In: 2018 IEEE Global Communications Conference (GLOBECOM), pp. 206–212. IEEE (2018)
18. Hsu, B.-Y., Shen, C.-Y.: Willingness maximization for ego network data extraction in online social networks. In: Proceedings of the 29th ACM International Conference on Information and Knowledge Management, pp. 515–524 (2020)
19. Hsu, B.-Y., Tu, C.-L., Chang, M.-Y., Shen, C.-Y.: Crawlsn: community-aware data acquisition with maximum willingness in online social networks. Data Mining Knowl. Disc. **34**(5), 1589–1620 (2020)
20. Hu, H., Yan, X., Huang, Y., Han, J., Zhou, X.J.: Mining coherent dense subgraphs across massive biological networks for functional discovery. Bioinformatics **21**(suppl–1), i213–i221 (2005)
21. Hu, S., Wu, X., Chan, T.: Maintaining densest subsets efficiently in evolving hypergraphs. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 929–938. ACM (2017)
22. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 1311–1322. ACM (2014)
23. Jethava, V., Beerenwinkel, N.: Finding dense subgraphs in relational graphs. In: Appice, A., Rodrigues, P.P., Santos Costa, V., Gama, J., Jorge, A., Soares, C. (eds.) ECML PKDD 2015. LNCS (LNAI), vol. 9285, pp. 641–654. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23525-7_39
24. Li, R.-H., Qin, L., Yu, J.X., Mao, R.: Influential community search in large networks. Proc. VLDB Endowment **8**(5), 509–520 (2015)
25. Li, R.-H., Su, J., Qin, L., Yu, J.X., Dai, Q.: Persistent community search in temporal networks. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE), pp. 797–808. IEEE (2018)
26. Luenberger, D.G., Ye, Y., et al.: Linear and Nonlinear Programming, vol. 2. Springer (1984). https://doi.org/10.1007/978-3-319-18842-3
27. Pei, J., Jiang, D., Zhang, A.: On mining cross-graph quasi-cliques. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, pp. 228–238. ACM (2005)
28. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: AAAI (2015)
29. Rozenshtein, P., Bonchi, F., Gionis, A., Sozio, M., Tatti, N.: Finding events in temporal networks: segmentation meets densest-subgraph discovery. In: 2018 IEEE International Conference on Data Mining (ICDM), pp. 397–406. IEEE (2018)
30. Rozenshtein, P., Tatti, N., Gionis, A.: Finding dynamic dense subgraphs. ACM Trans. Knowl. Disc. Data (TKDD) **11**(3), 27 (2017)
31. Sanei-Mehri, S-V., Das, A., Tirthapura, S.: Enumerating top-k quasi-cliques. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 1107–1112. IEEE (2018)
32. Saríyüce, A.E., Gedik, B., Jacques-Silva, G., Wu, K.-L., Çatalyürek, Ü.V.: Streaming algorithms for k-core decomposition. Proc. VLDB Endowment **6**(6), 433–444 (2013)
33. Shen, C.-Y., Fotsing, C.K., Yang, D.-N., Chen, Y.-S., Lee, W.-C.: On organizing online soirees with live multi-streaming. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)

34. Shen, C.-Y., Huang, L.-H., Yang, D.-N., Shuai, H.-H., Lee, W.-C., Chen, M.-S.: On finding socially tenuous groups for online social networks. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 415–424. ACM (2017)
35. Shen, C.-Y., Shuai, H.-H., Hsu, K.-F., Chen, M.-S.: Task-optimized group search for social internet of things. In: EDBT, pp. 108–119 (2017)
36. Shen, C.-Y., et al.: Forming online support groups for internet and behavior related addictions. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 163–172 (2015)
37. Shen, C.-Y., et al.: On extracting socially tenuous groups for online social networks with k-triangles. IEEE Tran. Knowl. Data Engineering (2020)
38. Shen, C.-Y., Yang, D.-N., Huang, L.-H., Lee, W.-C., Chen, M.-S.: Socio-spatial group queries for impromptu activity planning. IEEE Trans. Knowl. Data Eng. **28**(1), 196–210 (2015)
39. Shen, C.-Y., Yang, D.-N., Lee, W.-C., Chen, M.-S.: Activity organization for friend-making optimization in online social networks. IEEE Trans. Knowl. Data Eng. (2020)
40. Wang, J., Cheng, J.: Truss decomposition in massive networks. Proc. VLDB Endowment **5**(9), 812–823 (2012)
41. Wen, D., Qin, L., Zhang, Y., Chang, L., Chen, L.: Enumerating k-vertex connected components in large graphs. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 52–63. IEEE (2019)
42. Wu, H., et al.: Core decomposition in large temporal graphs. In: 2015 IEEE International Conference on Big Data (Big Data), pp. 649–658. IEEE (2015)
43. Yang, B., Wen, D., Qin, L., Zhang, Y., Chang, L., Li, R.-H.: Index-based optimal algorithm for computing k-cores in large uncertain graphs. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 64–75. IEEE (2019)
44. Yang, D.-N., Shen, C.-Y., Lee, W.-C., Chen, M.-S.: On socio-spatial group query for location-based social networks. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 949–957 (2012)
45. Zhang, C., Zhang, Y., Zhang, W., Qin, L., Yang, J.: Efficient maximal spatial clique enumeration. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 878–889. IEEE (2019)
46. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: When engagement meets similarity: efficient (k, r)-core computation on social networks. Proc. VLDB Endowment **10**(10), 998–1009 (2017)
47. Zhu, R., Zou, Z., Li, J.: Diversified coherent core search on multi-layer graphs. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE), pp. 701–712. IEEE (2018)