Durable Community Search on Temporal Graphs

Jianhua Wang[1], Jianye Yang[2,3]✉, Wu Yao[4], Ziyi Ma[5,6], Zhaoquan Gu[7], and Chengyuan Zhang[4]

[1] Inner Mongolia University, Hohhot, China
[2] Department of New Networks, PengCheng Laboratory, Shenzhen, China
[3] Guangzhou University, Guangzhou, China
[4] Hunan University, Changsha, China
[5] Hebei University of Technology, Tianjin, China
[6] Wuzhou University, Wuzhou, China
[7] Harbin Institute of Technology (Shenzhen), Shenzhen, China
yishiliucaihua@163.com, jyyang@gzhu.edu.cn, {wyao,cyzhangcse}@hnu.edu.cn, zyma@hebut.edu.cn, guzhaoquan@hit.edu.cn

Abstract. This paper studies the problem of durable community search on temporal graphs. Given a temporal graph $G_{[s,e]}$, a positive integer k, and a keyword set Q, we attempt to detect all connected k-trusses H of $G_{[s,e]}$ with (1) the keywords of H cover Q, (2) H has the largest existence interval. (3) there is no such k-truss $H' \supseteq H$ while also satisfying (1) and (2). This problem has many applications, such as bio-network analysis and anomaly detection. However, there is no efficient solution in the literature. In this paper, we first analyze the existence of durable communities among related time intervals and then devise a binary search-based method, namely BinaryDCS, which can skip fruitless intervals correctly. Besides, we optimize the intersection of snapshots by the segment tree. After that, we develop a novel framework, i.e., IncrementDCS, to enhance the pruning capacity by exploring the subintervals more orderly. Comprehensive performance studies on 3 real datasets show that our proposals outperform the baselines by up to 2 orders of magnitude.

Keywords: graph analysis · temporal graph · keyword search · cohesive subgraph · efficient algorithm.

1 Introduction

As a fundamental task to analyze graphs, community detection has received a lot of research efforts [5, 6]. In this paper, we focus on the k-truss model, which can reveal the cohesive connections in a graph [12]. For many real-world applications, the network is not static, e.g., on the Internet, links among terminals change over time. In addition, vertices should associate with keywords to make the detected subgraphs more meaningful [16].

Motivation. To integrate temporal attributes and keywords into the k-truss, we introduce the concept of durable community. Given a temporal graph $G_{[s,e]}$, a positive integer k, and a keyword set Q, a maximal connected k-truss $H \subseteq G_{[s,e]}$

is called a durable community if its keywords cover Q and the maximum interval of existence is largest. This paper studies the problem of Durable Community Search on temporal graphs (DCS for short). Below is a small list of applications.

(1) Analysis of PPI Networks. In a protein-protein interaction network about different ages [11], nodes contain metadata, which can be treated as keywords. After providing expected labels and k, we resolve DCS and collect the results. These communities have enough durability and cohesiveness, which reflects the most significant relationships of the network.

(2) Collaboration of Researchers. DBLP [1] is a dynamic co-authorship network, where nodes and edges represent authors and collaborations, respectively. To better analyze DBLP, we can assign each author several labels, which are from the biography. Given a set of keywords, a durable community in DBLP indicates that a group of researchers has durable yet deep cooperation.

Challenges. For solving DCS on $G_{[s,e]}$, a simple idea is to check all subintervals of $[s, e]$ and collect desired communities. However, in the worst case, the number of touched intervals is proportional to the square of $e - s$, which makes this approach too expensive for datasets with large time gaps. To the best of our knowledge, we are the first to investigate this problem thoroughly. Note that, [16] and [8] are both k-truss-related works. Unfortunately, they cannot support DCS because of different search targets.

Contributions. We first propose a competitive solution, i.e., BinaryDCS, which considers time intervals with fixed start time at the same time. In each step, it applies the binary search to update durable communities. Besides, we accelerate BinaryDCS by the optimization for calculating the intersection of graph snapshots. The main drawback of BinaryDCS is that the pruning power is poor. To enhance the performance, we develop a new framework, i.e., IncrementDCS, which improves the time complexity drastically. Empirical studies show that IncrementDCS significantly outperforms its competitors by up to 2 orders of magnitude. Our principal contributions are summarized as follows.

- This is the first work to study the DCS problem. We propose a binary search-based algorithm, i.e., BinaryDCS.
- We design a new search framework, i.e., IncrementDCS, to improve the performance of BinaryDCS.
- Extensive performance studies on 3 real datasets demonstrate the superiority of the proposed techniques.

2 Preliminaries

We consider a temporal graph $G_{[s,e]} = [G_s, G_{s+1}, ..., G_e]$, where G_t is a (simple and undirected) graph snapshot on time t. In this paper, we suppose the vertex set of $G_{[s,e]}$ is stable, i.e., V, while this work can be generalized to dynamic vertices easily. For each snapshot $G_t \in G_{[s,e]}$, $E(G_t) \subseteq V \times V$ denotes its edge set. Note that, a triangle $\triangle_{uvw} \in G_t$ implies that $(u, v), (v, w), (w, u) \in E(G_t)$.

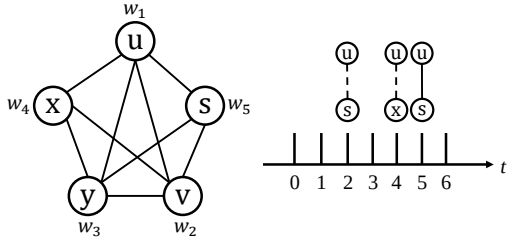


Fig. 1: Running example.

L is a function that assigns each node $v \in V$ a set of keywords, i.e., $L(v) \subseteq \Sigma$, where Σ is the keyword set of $G_{[s,e]}$. Given a subgraph $H \subseteq G_{[s,e]}$, we use $L(H)$ to represent the union of the keywords of vertices in H. An example temporal graph is shown in Figure 1. G_0 is a graph with 5 vertices (left part). For simplicity, when the graph changes, we only show the removed and inserted edges with dashed and solid lines, respectively (right part), e.g., (u, s) is removed when $t = 2$ and appears again when $t = 5$.

Definition 1 (Edge Support). *Given a graph G_t and an edge $e = (u, v) \in E(G_t)$, the support of e, denoted by $sup_e(G_t)$, is the number of triangles containing e in G_t, i.e., $sup_e(G_t) = |\{\triangle_{uvw} \in G_t \mid w \in V\}|$.*

Definition 2 (Connected k-truss). *Given a graph G_t and a parameter k, a connected k-truss is a connected subgraph $H \subseteq G_t$, such that $\forall e \in E(H)$, $sup_e(H) \geq k - 2$.*

Definition 3 (Duration). *Given a temporal graph $G_{[s,e]}$ and a subgraph H, we can organize all the time t with $H \subseteq G_t$ as several time intervals, i.e., $I = \{[s_1, e_1], [s_2, e_2], ..., [s_p, e_p]\}$, where p is the number of intervals and $s_1 \leq e_1 < s_2 \leq e_2 < s_p \leq e_p$. Then, the duration of H in $G_{[s,e]}$, denoted by $D_H(G_{[s,e]})$, is $\max_{i \in [1,p]} |e_i - s_i + 1|$.*

We call a subgraph $H \subseteq G_{[s,e]}$ dominates a keyword set Q if $Q \subseteq L(H)$ holds. Then, combined with the definition of duration, we introduce a cohesive yet significant subgraph model for temporal networks.

Definition 4 (Durable Community). *Given a temporal graph $G_{[s,e]}$, a positive integer k, and a set of keywords, i.e., $Q = \{w_1, w_2, ..., w_{|Q|}\}$, a subgraph $H \subseteq G_{[s,e]}$ is said to be a Durable Community (DC for short), if*
(1) H is a connected k-truss that dominates Q;
(2) H has the maximum duration; and
(3) there does not exist a subgraph $H' \supset H$, which also satisfies (1) and (2).

Problem Statement. Given a temporal graph $G_{[s,e]}$, a keyword set Q, and a positive integer k, we investigate the problem of durable communities search (this may output a set of results) on $G_{[s,e]}$ w.r.t. k and Q.

Example 1. Consider the temporal graph in Figure 1, i.e., $G_{[0,6]}$. Assume H_0, H_1, and H_2 are the induced subgraphs by vertex sets $\{u, x, y, v, s\}$, $\{u, x, y, v\}$, and

$\{u, y, v, s\}$, respectively. When $k = 4$ and $Q = \{w_1, w_2, w_3\}$, DCS will output H_1. Specifically, in $G_{[0,6]}$, H_0, H_1, and H_2 are all 4-trusses that dominate Q. Based on Definition 4, we have that $I_{H_0} = \{[0, 1]\}$, $I_{H_1} = \{[0, 3]\}$, and $I_{H_2} = \{[0, 1], [5, 6]\}$. Thus, $D_{H_0}(G_{[0,6]}) = D_{H_2}(G_{[0,6]}) = 2$, $D_{H_1}(G_{[0,6]}) = 4$, and H_1 is the DC.

k-truss Extraction. For solving DCS, we need to calculate the connected k-truss on certain graphs. To this end, we first adopt the ImprovedTrussDecomposition algorithm proposed in [12] to conduct truss decomposition. Then, we collect all the edges with enough support to obtain the k-truss. The details are omitted due to space limitations.

3 Baseline

3.1 Naive Approach

Main Idea. We use $G_{\cap}^{[i,j]}$ to denote the intersection of snapshots for time interval $[i, j]$ in $G_{[s,e]}$, i.e., $G_{\cap}^{[i,j]} = \cap_{t \in [i,j]} G_t$. Then, a straightforward approach, i.e., NaiveDCS, is first enumerating all time intervals of $G_{[s,e]}$, and for each of them, computing $G_{\cap}^{[i,j]}$. After that, it invokes ImprovedTrussDecomposition on $G_{\cap}^{[i,j]}$ and obtains the connected k-trusses that dominates Q. Finally, we collect all k-trusses with the maximum duration. In the implementation, we process time intervals in non-increasing order of the length. Thus, some intervals can be ignored, if its length is less than the current maximum duration, i.e., D_{max}. The detailed algorithm is omitted due to its simplicity.

Analysis. Let $T = e - s + 1$, it can be verified that the time (Resp. space) complexity of NaiveDCS is bounded by $O(T^2|E(G_{max})|^{1.5})$ (Resp. $O(|E(G_{[s,e]})|)$), where G_{max} is one snapshot $G_t \in G_{[s,e]}$ with the most edges. Obviously, NaiveDCS is too expensive for long periods.

3.2 Binary Search-based Approach

Motivation. NaiveDCS may consider all continuous time intervals of $G_{[s,e]}$ in the worst case, which brings redundant computations inevitably. To cope with this problem, we design a binary search-based algorithm, i.e., BinaryDCS. Specifically, given a temporal graph $G_{[s,e]}$, we first generate T groups with fixed start time $t \in [s, e]$. Then, by the binary search, for each of them, we can get the largest time $x \in [t, e]$, where there are connected k-trusses in $G_{\cap}^{[t,x]}$ that dominate Q. After processing all the T groups, the most durable k-truss is obtained.

Algorithm. Before detailing the algorithm, we first propose a lemma, which ensures the correctness of the binary search.

Lemma 1. *Given a temporal graph $G_{[s,e]}$, a positive integer k, and a keyword set Q. For a time interval $[i, j]$, we have the following two rules:*
(a) if there exists a connected k-truss that dominates Q in $G_{\cap}^{[i,j]}$, $G_{\cap}^{[i,x]}$ with $j > x \geq i$ cannot contain any DC;
(b) if there is no answer in $G_{\cap}^{[i,j]}$, then $G_{\cap}^{[i,y]}$ with $y > j$ cannot contain any DC

Algorithm 1 BinaryDCS

Require: $G_{[s,e]}$: a temporal graph; Q: a keyword set; k: a positive integer
Ensure: $\mathcal{R}_{D_{max}}$: all the durable communities

```
1: R ← ∅
2: D_max ← 0
3: for t ← s, e do
4:     if D_max > e − t + 1 then
5:         break
6:     end if
7:     l ← t
8:     r ← e
9:     while r ≥ l do
10:        m ← ⌊(r+l)/2⌋
11:        H ← ImprovedTrussDecomposition(G_∩^[t,m])
12:        R_temp ← ∅
13:        for each connected component H ∈ H do
14:            if IsDomination(H, Q) then
15:                R_temp ← R_temp ∪ {H}
16:            end if
17:        end for
18:        if R_temp = ∅ then
19:            r ← m − 1
20:            if D_max > r − t + 1 then
21:                break
22:            end if
23:        else
24:            if D_max ≤ m − t + 1 then
25:                D_max ← m − t + 1
26:                R_{D_max} ← R_{D_max} ∪ R_temp
27:            end if
28:            l ← m + 1
29:        end if
30:    end while
31: end for
32: return R_{D_max}
```

Proof. According to the definition of $G_{\cap}^{[i,j]}$, $G_{\cap}^{[i,y]} \subseteq G_{\cap}^{[i,j]} \subseteq G_{\cap}^{[i,x]}$ must hold. First, if there is an answer in $G_{\cap}^{[i,j]}$, the duration of the connected k-trusses that dominates Q in $G_{\cap}^{[i,x]}$ is not maximum (even if the size is larger). Second, if there is no result in $G_{\cap}^{[i,j]}$, $G_{\cap}^{[i,y]}$ cannot contain any DC (otherwise it leads to a contradiction). This completes the proof.

Algorithm 1 shows the details of BinaryDCS. It initializes R as an empty set and the maximum duration as 0 (Lines 1-2). Then, for each $t \in [s, e]$, it first inspects whether the length of $[t, e]$ is less than D_{max} (Line 4), if so, it can be terminated immediately, since the length of the following time intervals can not exceed D_{max}. Next, BinaryDCS conducts the binary search on $[t, e]$ (Lines 7-30). Specifically, for the median (say m) of the current interval, it performs ImprovedTrussDecomposition on $G_{\cap}^{[t,m]}$ and collects all of the connected components that dominate Q to $\mathcal{R}_{temp}$ (Lines 11-17). The details of IsDomination are omitted due to their simplicity. If $\mathcal{R}_{temp}$ is empty, it updates the interval as $[t, m-1]$ and tries to break early (Lines 19-22, Lemma 1(b)). Otherwise, it stores necessary subgraphs and updates the interval accordingly (Lines 24-28, Lemma 1(a)). After completing the binary search for all necessary time intervals, BinaryDCS simply returns the $\mathcal{R}_{D_{max}}$ (Line 32).

Intersection of Snapshots. BinaryDCS calculates $G_{\cap}^{[i,j]}$ for various time intervals, and a brute-force method may deteriorate the performance. The reason is that there are overlaps among intervals. In this paper, we use the segment tree to calculate $G_{\cap}^{[i,j]}$ efficiently. Given a temporal graph $G_{[s,e]}$, it takes $O(|E(G_{[s,e]})|)$ building time and extra space. Besides, the time complexity for computing $G_{\cap}^{[i,j]}$ is $O(logT|E(G_{max})|)$ [15].

Remark. All methods introduced in this paper are equipped with the segment tree. In real applications, $logT|E(G_{max})| \ll |E(G_{max})|^{1.5}$ often holds. Thus, the time cost for calculating $G_{\cap}^{[i,j]}$ does not affect the analysis of these algorithms.

Theorem 1. *The time complexity and space complexity of BinaryDCS are bounded by $O(TlogT|E(G_{max})|^{1.5})$ and $O(|E(G_{[s,e]})|)$, respectively.*

Proof. In the worst case, there are T iterations in BinaryDCS (Line 3). For each iteration, the binary search has $O(logT)$ loops and the bottleneck of each loop is ImprovedTrussDecomposition, which spends $O(|E(G_{\cap}^{[i,j]})|^{1.5})$ time. Thus, the overall time complexity of BinaryDCS is bounded by $O(TlogT|E(G_{max})|^{1.5})$. Note that, once $\mathcal{R}$ is updated in Line 26, we can discard all subgraphs whose duration is smaller than $m-t+1$. As a result, $\mathcal{R}$ only keeps all k-trusses with a certain duration, and the space cost of $\mathcal{R}$ is bounded by $|E(G_{[s,e]})|$. In addition, we simply reuse the memory for ImprovedTrussDecomposition. Thus, the space complexity of BinaryDCS is $O(|E(G_{[s,e]})|)$.

4 A New Method

Motivation. The search process in BinaryDCS is not conducted orderly, which reduces its pruning capacity. Thus, we develop a more efficient algorithm, i.e., IncrementDCS. It initializes the search from time interval $[s,s]$, and then enumerates intervals with larger end time until no k-truss dominates Q (say $[s,x]$). Now, based on Lemma 1(b), we prune all time intervals $[s,y]$ with $y \geq x$ and update D_{max} accordingly. Next, we calculate the target interval, i.e., $[s', \max(s', s' + D_{max} - 1)]$, where $s' = s+1$. The reason is that the following intervals with end time less than $\max(s', s' + D_{max} - 1)$ cannot contain any DC. Repeat the above operations, we obtain the solution of DCS.

Algorithm. The details of IncrementDCS are presented in Algorithm 2. It first reserves necessary variables (Lines 1-4). Then, while the current time interval (say $[l,r]$) is valid (Line 5), IncrementDCS conducts ImprovedTrussDecomposition on $G_{\cap}^{[l,r]}$ and collects all needed subgraphs to $\mathcal{R}_{temp}$ (Lines 6-11). Next, it updates D_{max} and r (Lines 17-21). Finally, the algorithm returns $\mathcal{R}_{D_{max}}$. It should be remarked that once $\mathcal{R}_{temp}$ is empty, we should generate a new interval to avoid fruitless efforts (Lines 12-16).

Theorem 2. *The time complexity and space complexity of IncrementDCS are $O(T|E(G_{max})|^{1.5})$ and $O(|E(G_{[s,e]})|)$, respectively.*

Algorithm 2 IncrementDCS

Require: $G_{[s,e]}$: a temporal graph; Q: a keyword set; k: a positive integer
Ensure: $\mathcal{R}_{D_{max}}$: all the durable communities

1: $\mathcal{R} \leftarrow \emptyset$
2: $D_{max} \leftarrow 0$
3: $l \leftarrow s$
4: $r \leftarrow s$
5: **while** $r \leq e$ **and** $D_{max} \leq e - l + 1$ **do**
6: $\quad \mathcal{H} \leftarrow$ ImprovedTrussDecomposition$(G_{\cap}^{[l,r]})$
7: $\quad$ **for each** connected component $H \in \mathcal{H}$ **do**
8: $\quad\quad$ **if** IsDomination(H, Q) **then**
9: $\quad\quad\quad \mathcal{R}_{temp} \leftarrow \mathcal{R}_{temp} \cup \{H\}$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: $\quad$ **if** $\mathcal{R}_{temp} = \emptyset$ **then**
13: $\quad\quad l \leftarrow l + 1$
14: $\quad\quad r \leftarrow \max(l, l + D_{max} - 1)$
15: $\quad\quad$ **continue**
16: $\quad$ **end if**
17: $\quad$ **if** $D_{max} \leq r - l + 1$ **then**
18: $\quad\quad D_{max} \leftarrow r - l + 1$
19: $\quad\quad \mathcal{R}_{D_{max}} \leftarrow \mathcal{R}_{D_{max}} \cup \mathcal{R}_{temp}$
20: $\quad$ **end if**
21: $\quad r \leftarrow r + 1$
22: **end while**
23: **return** $\mathcal{R}_{D_{max}}$

Table 1: Characteristics of real-life datasets

Dataset	**Abbreviation**	**Category**	$\|V\|$	$\|E\|$	$\overline{d}$
DNC Emails	DE	Email	2,029	39,264	38.7
UC Irvine Messages	UI	Message	1,899	59,835	63.0
Linux Kernel Mailing List Replies	LK	Reply	63,399	1,096,440	34.6

Proof. IncrementDCS starts the search from $[s, s]$ (Lines 3-4). In each iteration, it increases l (Line 13) or r (Line 21) by 1. Since l and r cannot exceed e, the number of iterations is $O(T)$. Besides, ImprovedTrussDecomposition takes $O(|E(G_{\cap}^{[i,j]})|^{1.5})$ time, so the overall time complexity of IncrementDCS is $O(T|E(G_{max})|^{1.5})$. The space complexity can be proved the same as Theorem 1.

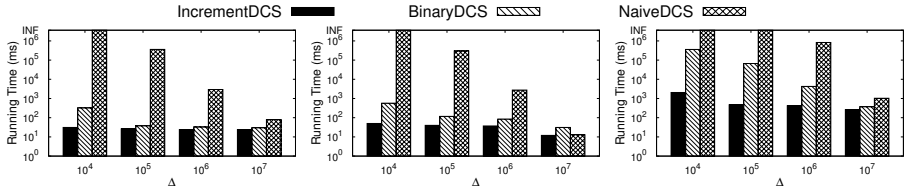
5 Experiment

All experiments are conducted on PCs with Intel Xeon 2 × 2.4GHz CPU containing 40 cores and 128GB RAM running Ubuntu 22.04.2 LTS. A process is terminated if it runs more than 1 hour, which is denoted as INF.

5.1 Experimental Setup

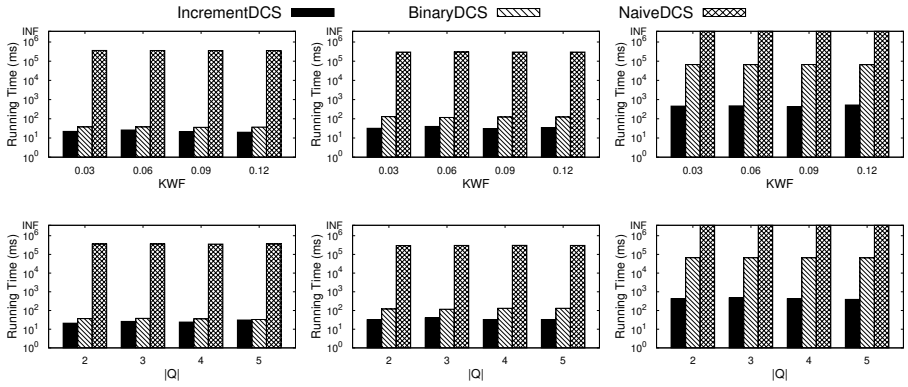
Algorithms. In the experiments, we consider the following three algorithms.

- NaiveDCS. The naive idea introduced in Section 3.1.
- BinaryDCS. The binary search-based method proposed in Section 3.2.
- IncrementDCS. The new search framework derived in Section 4.



(a) DE (b) UI (c) LK

Fig. 2: Varying Δ ($\mathsf{KWF} = 0.06$, $|Q| = 3$).



(a) DE (b) UI (c) LK

Fig. 3: Varying KWF and $|Q|$ ($\Delta = 10^5$).

Datasets. We focus on 3 real-life datasets [7], and the detailed characteristics of them are reported in Table 1. DE is obtained from a collection of emails. UI is about the messages among the students of an online community. LK is the communication network of the Linux kernel mailing list. Note that, the periods of these datasets are too large. To increase the stability of the results, we split the whole span as several intervals by the time gap Δ. Besides, there are no keywords within these datasets. Inspired by the methods in [16], we generate keywords based on the keyword frequency (KWF).

Parameter settings. For time gap Δ, we employ 10^4, ..., 10^7, where 10^5 is the default. For KWF, 0.03, ..., 0.12 is used, where 0.06 is the default. For query size $|Q|$, $2-5$ is adopted, and 3 is the default. Unless otherwise specified, experiments are conducted with the default settings.

5.2 Performance Evaluation

We consider 3-trusses and only compare the running time of the three algorithms, since they have the same space complexity and our early experiments show that there is indeed no significant difference.

Exp-1: Varying time gap. We first compare the efficiency of these algorithms by varying Δ. The experimental results are presented in Figure 2. It is observed that in most cases, the performance of NaiveDCS is poor. BinaryDCS is much

faster because of the effectiveness of the pruning techniques. However, IncrementDCS conducts the search procedure more orderly, and thus outperforms BinaryDCS on all datasets, e.g., on LK with $\Delta = 10^5$, it achieves 2 orders of magnitude speedup.

Exp-2: Varying KWFand query size. We then show the influence of KWF and $|Q|$ for all methods. Figure 3 reports the experimental results. It is noticed that KWF and $|Q|$ do not have a great influence on the performance. The reason is that these algorithms all conduct ImprovedTrussDecomposition and domination verification successively, while the former takes the most time. Besides, IncrementDCS also outperforms its competitors (by up to 2 orders on LK).

6 Related Works

k-truss-based subgraph mining. For general graphs, [2] first introduces the concept of k-truss to analyze social networks. [14] integrates the signed information to ensure the significance and stability of the communities. For temporal graphs, [13] develops the novel EquiTree-index to support fast k-truss community search. It should be remarked that [8] also employs the temporal attribute. However, it only focuses on user-given durability (say Δ), which differs from DCS. For labeled graphs, [16] solves the dense truss problem and uses the KT-index to speed up queries.

Other cohesive subgraph models. k-core is a well-known density-based model. [4] utilizes the keyword cohesiveness of k-core to retrieve more meaningful communities. [9] applies the k-ECC to community search and employs an effective index to accelerate computation. [10] employs the concept of "pivot" to improve the worst-case time complexity for maximal clique enumeration. [3] introduces the quasi-clique model and conducts sufficient experiments to illustrate how to tune the parameters to retrieve desired results.

7 Conclusion

In this paper, we study the DCS problem, which is computationally challenging. To resolve DCS efficiently, we propose a binary search-based algorithm and utilize the segment tree to speed up the intersection of snapshots. To further improve the performance, we develop a search framework with more pruning capacity. Comprehensive experiments on 3 real datasets demonstrate the superiority of our techniques compared with the baselines.

Acknowledgments. This work was funded in part by the Major Key Project of PCL (PCL2024A05), in part by the Guangdong Basic and Applied Basic Research Foundation (2023A1515011655), in part by the National Natural Science Foundation of China (62372137 and 62472161), in part by the Hunan Provincial Natural Science Foundation of China (2023JJ30169), in part by the Science Research Project of the Hebei Education Department (BJK2024172), in part by the High-level Talents Introduction Project

of Inner Mongolia University (10000-23112101/270), and in part by the Guangxi Key Laboratory of Machine Vision and Intelligent Control (2023B03).

References

1. Biryukov, M., Dong, C.: Analysis of computer science communities based on dblp. In: Research and Advanced Technology for Digital Libraries: 14th European Conference. pp. 228–235 (2010)
2. Cohen, J.: Trusses: Cohesive subgraphs for social network analysis. National Security Agency Technical Report **16**(3.1), 1–29 (2008)
3. Cui, W., Xiao, Y., Wang, H., Lu, Y., Wang, W.: Online search of overlapping communities. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. pp. 277–288 (2013)
4. Fang, Y., Cheng, R., Luo, S., Hu, J.: Effective community search for large attributed graphs. Proceedings of the VLDB Endowment **9**(12), 1233–1244 (2016)
5. 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 (2014)
6. Jin, D., Yu, Z., Jiao, P., Pan, S., He, D., Wu, J., Philip, S.Y., Zhang, W.: A survey of community detection approaches: From statistical modeling to deep learning. IEEE Transactions on Knowledge and Data Engineering **35**(2), 1149–1170 (2021)
7. Kunegis, J.: Konect: the koblenz network collection. In: the 22nd International Conference on World Wide Web. pp. 1343–1350 (2013)
8. Lantian, X., Ronghua, L., Guoren, W., Biao, W.: Research on k-truss community search algorithm for temporal networks. Journal of Frontiers of Computer Science and Technology **14**(9), 1482 (2020)
9. Meng, S., Yang, H., Liu, X., Chen, Z., Xuan, J., Wu, Y.: Personalized influential community search in large networks: A k-ecc-based model. Discrete Dynamics in Nature and Society **2021**, 1–10 (2021)
10. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science **363**(1), 28–42 (2006)
11. Vazquez, A., Flammini, A., Maritan, A., Vespignani, A.: Global protein function prediction from protein-protein interaction networks. Nature Biotechnology **21**(6), 697–700 (2003)
12. Wang, J., Cheng, J.: Truss decomposition in massive networks. Proceedings of the VLDB Endowment **5**(9), 812–823 (2012)
13. Xu, T., Lu, Z., Zhu, Y.: Efficient triangle-connected truss community search in dynamic graphs. Proceedings of the VLDB Endowment **16**(3), 519–531 (2022)
14. Zhao, J., Sun, R., Zhu, Q., Wang, X., Chen, C.: Community identification in signed networks: a k-truss based model. In: Proceedings of the 29th ACM International Conference on Information and Knowledge Management. pp. 2321–2324 (2020)
15. Zheng, C., Shen, G., Li, S., Shenker, S.: Distributed segment tree: Support of range query and cover query over dht. In: Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS) (2006)
16. Zhu, Y., Zhang, Q., Qin, L., Chang, L., Yu, J.X.: Cohesive subgraph search using keywords in large networks. IEEE Transactions on Knowledge and Data Engineering **34**(1), 178–191 (2022)