QTCS: Efficient Query-Centered Temporal Community Search

Longlong Lin
Southwest University,
College of Computer and
Information Science
longlonglin@swu.edu.cn

Pingpeng Yuan
Huazhong University of
Science and Technology
ppyuan@hust.edu.cn

Rong-Hua Li
Beijing Institute of
Technology
lironghuabit@126.com

Chunxue Zhu
Huazhong University of
Science and Technology
cxzhu@hust.edu.cn

Hongchao Qin
Beijing Institute of
Technology
qhc.neu@gmail.com

Hai Jin
Huazhong University of
Science and Technology
hjin@hust.edu.cn

Tao Jia
Southwest University
tjia@swu.edu.cn

ABSTRACT

Temporal community search is an important task in graph analysis, which has been widely used in many practical applications. However, existing methods suffer from two major defects: (i) they only require that the target result contains the query vertex q, leading to the temporal proximity between q and other vertices being ignored. Thus, they may find many temporal irrelevant vertices (these vertices are called query-drifted vertices) concerning q for satisfying their objective functions; (ii) their methods are NP-hard, incurring high costs for exact solutions or compromised qualities for approximate/heuristic algorithms. In this paper, we propose a new problem named query-centered temporal community search to overcome these limitations. Specifically, we first present a novel concept of Time-Constrained Personalized PageRank to characterize the temporal proximity between q and other vertices. Then, we introduce a model called β-temporal proximity core, which can seamlessly combine temporal proximity and structural cohesiveness. Subsequently, our problem is formulated as an optimization task that finds a β-temporal proximity core with the largest β. We theoretically prove that our problem can circumvent these query-drifted vertices. To solve our problem, we first devise an exact and near-linear time greedy removing algorithm that iteratively removes unpromising vertices. To improve efficiency, we then design an approximate two-stage local search algorithm with bound-based pruning techniques. Finally, extensive experiments on eight real-life datasets and nine competitors show the superiority of the proposed solutions.

PVLDB Reference Format:
Longlong Lin, Pingpeng Yuan, Rong-Hua Li, Chunxue Zhu, Hongchao Qin, Hai Jin, and Tao Jia. QTCS: Efficient Query-Centered Temporal Community Search. PVLDB, 17(6): 1187 - 1199, 2024.
doi:10.14778/3648160.3648163

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/longlonglin/QTCS.

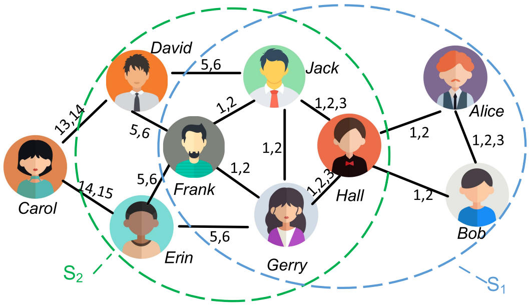


Figure 1: Motivation Example. *Frank* **is the query vertex.** S_1 **is the answer of [12],** S_2 **is our answer.**

1 INTRODUCTION

Real-life graphs exhibit rich community structures that are defined as densely connected subgraphs. Community mining is a significant vehicle for analyzing network organization. In general, community mining can be divided into community detection [2, 7, 25, 33, 38] and community search [3, 6, 44, 55]. The former aims to find *all* communities by some predefined criteria (e.g., Modularity or Conductance [25]), resulting in that it is time-consuming and not customized for user-specified query requests. To alleviate these defects, the latter identifies the specific community containing the user-initiated query vertex, which is more efficient and personalized.

Despite the significant success of community search, most existing approaches are tailored to static networks, which ignore the potential time interaction information among vertices. For example, in e-commerce or social media, the connection between two parties was made at a specific time. Such networks are named temporal networks [13]. It would be possible to use static community search methods to temporal networks by ignoring the time information of edges, but such solutions have been shown to be sub-optimal [4, 12, 23, 35, 46]. For example, Figure 1 shows a sample money transfer network, in which the timestamps of each edge indicate when the two individuals make transactions. We assume *Frank* is the query vertex. By using 2-core as the community model (i.e., each vertex has at least 2 neighbors in the community. Note that 3-core does not exist in Figure 1), the whole graph $\mathcal{G}$ is the answer [6, 44]. However, the occurrence time of the transactions among $\mathcal{G}$ differs greatly. Thus, $\mathcal{G}$ is an unpromising temporal community. Recently, some studies have been done on temporal community search. For example, Galimberti et al. [12] proposed the span-core

Proceedings of the VLDB Endowment, Vol. 17, No. 6 ISSN 2150-8097.
doi:10.14778/3648160.3648163

model, which persistently maintains a k-core in every timestamp of a small time window. As a result, S_1 is the answer of [12] (because S_1 is 2-core in every timestamp of $[1, 2]$). However, we observe that *Frank* cannot walk to *Alice* and *Bob* in chronological order, resulting in that they have poor temporal proximity with respect to (w.r.t.) *Frank* (see Section 2.2 and Definition 2.1 for details).

In this paper, we introduce a new problem named query-centered temporal community search (*QTCS*), which aims at identifying a community with good temporal proximity w.r.t. the query vertex. Thus, the vertices of S_2 may be the target community. This is because *Frank* has many walks to the vertices of S_2 in chronological order, resulting in that S_2 has good temporal proximity w.r.t. *Frank*. Note that *Carol* is not included in S_2 because the time difference of the walk from *Frank* to *Carol* is too big (e.g., the time difference of $\{(Frank, David, 5), (David, Carol, 14)\}$ is $14 - 5 = 9$), resulting in that *Carol* has also poor temporal proximity w.r.t. *Frank* (Section 2.2 and Definition 2.3). For applications such as anti-money-laundering, we would like to search the community that contains a known suspicious account (e.g., the query node $Frank$), and investigate the associated counts (e.g., nodes of S_2) [21]. Besides, on temporal collaboration networks, *QTCS* may be the research group initiated by the given query vertex. So, detecting *QTCS* enables us to reveal interesting communities that revolve around the query vertex.

Unfortunately, existing temporal community search methods suffer from two major challenges in terms of *QTCS* [4, 12, 23, 35, 46]. First, in the community search problem, the vertices of the target community should have good proximity w.r.t. the query vertex [18, 52]. However, *all existing methods* do not consider the temporal proximity between the query vertex q and other vertices but simply require the result including q. Namely, they underestimate the importance of q in the target community. Thus, they may find many temporal irrelevant vertices (e.g., *Alice* and *Bob* in Figure 1) concerning q for satisfying their objective functions. We denote such temporal irrelevant vertices as *query-drifted vertices* (Section 3.2). Second, most existing methods are NP-hard, incurring either prohibitively high costs for exact solutions or severely compromised results for approximate/heuristic algorithms. For example, [23, 46] cannot obtain results within two days in our experiments, which is clearly impractical for online interactive graph explorations.

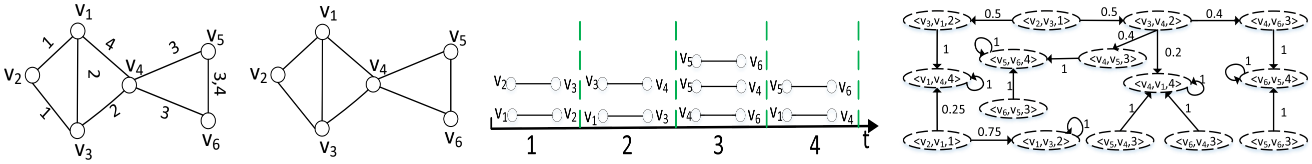
For the first challenge, we extend the well-known proximity metric Personalized PageRank [24, 30, 48, 50] to temporal networks and define the corresponding Time-Constrained Personalized PageRank (*TPPR*) by non-trivially integrating temporal constraint, which can more properly capture the temporal proximity between the query vertex and other vertices. Equipped with *TPPR*, we then propose β-temporal proximity core to model the preference of user-specified query vertex by combining seamlessly the temporal proximity and structural cohesiveness. Consequently, by maximizing the value of β, we proved theoretically that these *query-drifted vertices* are removed (Section 3.2 and 6). Besides, the temporal proximity core has only one parameter (i.e., the teleportation probability α in Section 2.3), which is user-friendly. In contrast, [4, 23, 35] have many parameters which are heavily dependent on datasets and are often hard-to-tune. For the second challenge, we propose two efficient algorithms. Specifically, we first develop an exact and near-linear time greedy removing algorithm called *EGR*. *EGR* first computes *TPPR* for every vertex and then greedily selects out the vertices with the minimum query-biased temporal degree (Definition 2.5). Clearly, the core of *EGR* is how to compute *TPPR* quickly. A baseline solution for *TPPR* is to apply the existing push-based methods [1, 47] or power iteration-based methods [34, 50], but they require prohibitively high time costs. Based on in-depth observations, we find that the state transition graph of a temporal graph is acyclic. Thus, we leverage the acyclic property to devise an efficient dynamic programming algorithm to compute *TPPR* with one pass over all temporal edges. To boost efficiency, we develop an approximate two-stage local search algorithm named *ALS* with several powerful pruning techniques. The high-level idea of *ALS* is to adopt the expanding and reducing paradigm. The expanding stage directly starts from the query vertex and progressively adds qualified vertices with proposed bound-based pruning techniques. Until it touches the termination condition with theoretical guarantees. The reducing stage iteratively removes unqualified vertices to satisfy the approximation ratio. Our main contributions are as follows.

- **Novel Model.** We formulate the *query-centered* temporal community search problem in Section 2, which can reduce irrelevant vertices and provide more accurate results.
- **Theoretical Analysis.** We introduce the concept of *query-drifted vertices* to analyze the limitations of the existing solutions in Section 3. We show that most existing methods contain many *query-drifted vertices.* However, our model can circumvent these *query-drifted vertices.*
- **Efficient Algorithms.** To solve our problem quickly, we propose two practical algorithms in Section 4 and Section 5. One of them is the exact greedy removing algorithm *EGR* with near-linear time complexity. The other is the approximate two-stage local search algorithm *ALS.*
- **Comprehensive Experiments.** Experiments (Section 6) on eight datasets with different domains and sizes demonstrate our proposed solutions indeed are more efficient, scalable, and effective than the nine competitors. For instance, on a million-vertex DBLP dataset, *ALS* consumes about 13 seconds while *EGR* takes 47 seconds. However, some competitors cannot get the results within two days on some datasets. Besides, our model is much denser and more separable in terms of temporal features than the competitors. Our model can find high-quality *query-centered* temporal communities by eliminating *query-drifted vertices* that the competitors cannot identify.

2 PROBLEM FORMULATION

2.1 Notations

Let $\mathcal{G}(V, \mathcal{E})$ be an undirected temporal graph, in which V (resp., $\mathcal{E}$) indicates the vertex set (resp., the temporal edge set). Let $(u, v, t) \in \mathcal{E}$ be any temporal edge that indicates an interaction was made between u and v at timestamp t. Note that (u, v, t_1) and (u, v, t_2) are regarded as two different temporal edges if $t_1 \neq t_2$. That is, u and v may be connected at different timestamps. Let $|V| = n$ and $|\mathcal{E}| = m$ be the number of vertices and the number of temporal edges, respectively. For example, Figure 2(a) illustrates a sample temporal graph $\mathcal{G}$ with 6 vertices and 9 temporal edges. More generally, temporal graphs can also be modeled as an edge stream [13], which is a sequence of all temporal edges ordered by timestamps. Figure 2(c) shows the edge stream representation for Figure 2(a). We use $G(V, E)$ to denote the de-temporal graph of $\mathcal{G}$, in which $E = \{(u, v) | (u, v, t) \in \mathcal{E}\}$ and $|E| = \bar{m}$. That is, G is a static graph that ignores the timestamps of $\mathcal{G}$. Figure 2(b) shows a de-temporal



(a) Temporal graph $\mathcal{G}$ (b) De-temporal graph G (c) Edge stream representation for $\mathcal{G}$ (d) State transition graph

Figure 2: De-temporal graph, Edge stream and State transition of an example temporal graph

graph G. Let $G_S = (S, E_S)$ be the subgraph induced by S if $S \subseteq V$ and $E_S = \{(u, v) \in E | u, v \in S\}$. Let $N_S(v) = \{u \in S | (u, v) \in E\}$ be the neighbors of v in S.

2.2 Time-Constrained Personalized PageRank

Proximity is a fundamental concept in static graph analysis, which aims to characterize the similarity between two nodes by summarizing all walks between them [53]. Since community search aims to find a vertex set similar to the given query node, proximity can be applied to improve the effectiveness of community search [1, 44]. In this paper, we extend the proximity to temporal networks and study the corresponding temporal proximity. In the literature, several studies have been done on temporal proximity. For example, [14, 37] first converted the temporal graph into a weighted graph and then applied the traditional method over the weighted graph to define the temporal proximity. These methods, however, only consider the temporal information of two directly-connected vertices, missing higher-order temporal and structural information. [31] adopted the tensor to represent the temporal network and calculated the eigenvector of the tensor to capture the temporal proximity, which is inefficient for handling large graphs. [40] is most relevant to our work. We discuss the main differences between [40] and us in Table 1 and Section 6. A high-level definition of temporal proximity is as follows.

Definition 2.1. **[Temporal Proximity [13]]** Given a temporal graph $\mathcal{G}(V, \mathcal{E})$ and two vertices $u, q \in V$. u has poor (resp., good) temporal proximity w.r.t q if (1) q has few (resp., many) walks to u in chronological order; (2) The probability of these walks being visited is small (resp., large).

Inspired by Definition 2.1, we propose the Time-Constrained Personalized PageRank to effectively capture the temporal proximity between two vertices. Before proceeding further, we review the most representative proximity model Personalized PageRank (*PPR*) [30, 48]. Essentially, *PPR* models a random walk process that has a unique stationary distribution and solves the equation:

$$\mathbf{x} = \alpha \mathbf{s} + (1 - \alpha)\mathbf{x}\mathbf{W} \tag{1}$$

$\mathbf{x}$ is the stationary *PPR* distribution, α is the teleportation probability, and $\mathbf{s}$ is a start distribution named the teleportation vector. $\mathbf{W}$ is the state transition matrix, where each entry W_{vu} indicates the transition probability from vertex v to vertex u.

Although *PPR* has achieved significant success in static networks, how to preserve the rich temporal information in *PPR* still faces two challenges as follows. First, how to design an effective walk for temporal networks. In real-world scenarios, the information transmission follows the chronological order. For example, (v_2, v_1, v_4, v_5) is a walk in Figure 2 (b), but (v_2, v_1, v_4, v_5) in Figure 2 (a) is clearly problematic w.r.t. chronological order. Such chronological order constraints frequently occur in temporal networks, such as temporal money transfer networks, transportation transfer networks, and disease transmission networks [13, 49]. Second, how to design an effective state transition matrix in temporal networks. Intuitively, the preference for an interaction decreases as time goes by [54] (i.e., the tie between two vertices becomes stronger if the interaction between them happens in a more current time). For instance, in Figure 2(a), when the walker walks to v_1 through temporal edge $(v_2, v_1, 1)$, the probability that the walker chooses $(v_1, v_3, 2)$ to walk is higher than $(v_1, v_4, 4)$. This is because the time difference between $(v_2, v_1, 1)$ and $(v_1, v_3, 2)$ (i.e., 2-1=1) is smaller than the difference between $(v_2, v_1, 1)$ and $(v_1, v_4, 4)$ (i.e., 4-1=3). Another example is in temporal money transfer networks, the money transaction $\{(a, b, 1), (b, c, 2)\}$ has a better transaction experience than $\{(a, b, 1), (b, d, 2000)\}$ because $(b, c, 2)$ responds more promptly. However, the traditional state transition matrix $\mathbf{W}$ cannot distinguish such edge relationships. Additionally, more than one interaction may occur between two vertices in temporal networks. So, $\mathbf{W}$ is not applicable for modeling temporal proximity.

For ease of description, we convert each temporal edge to two *ordered* temporal edges of opposing directions. For example, (u, v, t) converts to $< u, v, t >$ and $< v, u, t >$ (to avoid confusion, we use () and <> to represent the temporal edge and *ordered* temporal edge, respectively.). Moreover, we use $\vec{e}$ to denote any *ordered* temporal edge. Let $head(\vec{e})$, $tail(\vec{e})$ and $time(\vec{e})$ be the head vertex, tail vertex and timestamp of $\vec{e}$, $N^{>}(\vec{e}) = \{< u, v, t > | u = head(\vec{e}), t > time(\vec{e})\}$, $\vec{e}^{\,out}_u = \{\vec{e} | tail(\vec{e}) = u\}$, $\vec{e}^{\,in}_u = \{\vec{e} | head(\vec{e}) = u\}$. Based on these symbols, we present the following definition to overcome the challenges discussed above.

Definition 2.2. **[Temporal transition matrix]** Given a temporal graph $\mathcal{G}(V, \mathcal{E})$, the temporal transition matrix $\mathbf{P} \in R^{m \times m}$ on two *ordered* temporal edges $\vec{e}_i$ and $\vec{e}_j$ can be computed as

$$P(\vec{e}_i \rightarrow \vec{e}_j) = \begin{cases} \frac{g(time(\vec{e}_j) - time(\vec{e}_i))}{\sum\limits_{\vec{e}_k \in N^{>}(\vec{e}_i)} g(time(\vec{e}_k) - time(\vec{e}_i))}, & \vec{e}_j \in N^{>}(\vec{e}_i) \\ 0, & \vec{e}_j \notin N^{>}(\vec{e}_i) \end{cases} \tag{2}$$

$P(\vec{e}_i \rightarrow \vec{e}_j)$ indicates the temporal transition probability from $\vec{e}_i$ to $\vec{e}_j$ and $g(a - b)$ is a decaying function to capture the dependency between interactions. Here, we apply a linear decaying function $g(a - b) = \frac{1}{a - b}$, which is often used in temporal settings [20, 51]. Our proposed solutions can trivially accommodate different functions (e.g., exponential or logarithmic function). In the case that $\sum_{\vec{e}_j} P(\vec{e}_i \rightarrow \vec{e}_j) = 0$, we call $\vec{e}_i$ a dangling state as [30, 48]. For simplicity, we set $P(\vec{e}_i \rightarrow \vec{e}_i) = 1$ to handle these dangling states. By doing so, we can guarantee that $\mathbf{P}$ is a stochastic matrix, that is, $\sum_{\vec{e}_j} P(\vec{e}_i \rightarrow \vec{e}_j) = 1$ for any $\vec{e}_i$ holds.

Definition 2.3. **[Time-Constrained Personalized PageRank (*TPPR*)]** Given a temporal graph $\mathcal{G}(V, \mathcal{E})$, a query vertex q and a teleportation probability α, the Time-Constrained Personalized PageRank of vertex u is denoted by $tppr_q(u) = \sum_{\vec{e} \in \vec{e}_u^{in}} \widetilde{ppr}(\alpha, \widetilde{\chi_q})(\vec{e})$.

$$\widetilde{ppr}(\alpha, \widetilde{\chi_q}) = \alpha \widetilde{\chi_q} + (1 - \alpha)\widetilde{ppr}(\alpha, \widetilde{\chi_q})\mathbf{P} \qquad (3)$$

$\widetilde{\chi_q} \in R^{1 \times m}$ is a vector with $\widetilde{\chi_q}(\vec{e}) = 1/|\vec{e}_q^{out}|$ for $\vec{e} \in \vec{e}_q^{out}$.

We explain the intuition behind Definition 2.3 as follows: (i) Equation 3 is also a random walk process analogous to Equation 1, except that each state in Equation 3 is an *ordered* temporal edge instead of a vertex. Thus, $\widetilde{ppr}(\alpha, \widetilde{\chi_q})(\vec{e})$ can reflect the proximity of each *ordered* temporal edge $\vec{e}$ w.r.t. q. (ii) Since $\mathbf{P}$ is a stochastic matrix, $\widetilde{ppr}(\alpha, \widetilde{\chi_q})$ is a probability distribution [30, 48]. Thus, $\sum_u tppr_q(u) = \sum_u \sum_{\vec{e} \in \vec{e}_u^{in}} \widetilde{ppr}(\alpha, \widetilde{\chi_q})(\vec{e}) = 1$. That is, $tppr_q$ is also a probability distribution. (iii) $tppr_q(u)$ is equivalent to the probability sum of the temporal walks from q to u (Lemma 4.3), which conforms to the intuition of Definition 2.1. So, it is reasonable to use $tppr_q(u)$ to describe the temporal proximity of u w.r.t. q. For simplicity, we use $tppr(u)$ to denote $tppr_q(u)$ if the context is clear.

2.3 Problem Statement

As mentioned in Section 2.2, *TPPR* can be used to measure the temporal proximity between the query vertex and other vertices. Thus, a naive way is to find a connected subgraph containing the query vertex and has optimal *TPPR* score. Unfortunately, it ignores the fact that a perfect temporal community should also have strong structural cohesiveness. Thus, another potential approach is to adopt the cohesive subgraph model k-core to model the structural cohesiveness of the community [6, 44]. We call this model *QTCS_Baseline*, which serves as a baseline model for experimental comparison.

Definition 2.4. **[*QTCS_Baseline*]** For a temporal graph $\mathcal{G}(V, \mathcal{E})$, a teleportation probability α, a query vertex q and a parameter k, *QTCS_Baseline* finds a vertex set S, satisfying (i) $q \in S$; (ii) G_S is a connected k-core (i.e., $|N_S(v)| \geq k$ for any $v \in S$); (iii) $\min\{tppr(u)|u \in S\}$ is maximum.

However, *QTCS_Baseline* considers separately structural cohesiveness and temporal proximity, resulting in that it may identify a sub-optimal result. For example, *QTCS_Baseline* may remove many vertices with good temporal proximity under the structural constraints of the k-core. Conversely, it may contain many vertices with poor temporal proximity to satisfy the structural cohesiveness. Thus, we propose the following novel metrics to combine seamlessly structural cohesiveness and temporal proximity.

Definition 2.5. **[Query-biased temporal degree]** Given a vertex set C, the query-biased temporal degree of vertex u w.r.t. C is defined as $\rho_C(u) = \sum_{v \in N_C(u)} tppr(v)$.

By Definition 2.5, we know that the query-biased temporal degree measures the quality (i.e., temporal proximity w.r.t. the query vertex) of neighbors rather than quantity. For example, u has 10^5 neighbors and each neighbor has a *TPPR* value of 10^{-10}. As a result, the query-biased degree of u is 10^{-5}. On the other hand, suppose u has only 10 neighbors, but each neighbor has a *TPPR* value of 10^{-2}. In this case, the query-biased degree of u is 10^{-1}. So, the higher the query-biased temporal degree of u, u may have more neighbors with good temporal proximity w.r.t. the query vertex.

Definition 2.6. **[β-temporal proximity core]** The β-temporal proximity core is a vertex set C, satisfying (i) G_C is connected; (ii) $\min\{\rho_C(u)|u \in C\} \geq \beta$.

By maximizing the value of β of a β-temporal proximity core, we can detect a community in which each vertex of the community has many neighbors with good temporal proximity w.r.t. the query vertex. As a result, it ensures that the detected community is very related to the query vertex, which makes it is easier to interpret why the community is formed (see case studies of Section 6).

Problem 1 (*QTCS*). Given a temporal graph $\mathcal{G}(V, \mathcal{E})$, a teleportation probability α and a query vertex q, ***query-centered*** **t**emporal **c**ommunity **s**earch aims to identify a vertex set C, satisfying (i) $q \in C$; (ii) C is a β-temporal core with the largest β; (iii) there does not exist another community $C' \supseteq C$ meets the above conditions.

3 PROBLEM ANALYSIS

3.1 Comparison with *CSM*

The community search by maximizing the minimum degree (*CSM*) [6, 44] does have many similarities with our methods, but there are also pivotal differences. Firstly, a key concept in *CSM* is the degree of each vertex. So, we can simply adapt the *CSM* model to solve the temporal community search problem by using a concept of temporal degree. Specifically, the temporal degree of a vertex u is the number of temporal edges that u participates in. Such a simple adaption, however, has some serious defects. For example, the temporal degree is a local metric used to measure the absolute importance of vertices in the network. However, for the community search problem, it may be more appropriate to consider the relative importance between the query vertex and other vertices [18, 52]. Unlike *CSM*, our solution is based on a new definition of query-biased temporal degree (Definition 2.5) which can capture the relative importance of temporal community search. Secondly, in *CSM*, the (temporal) degree of a vertex can be obtained by simply checking the number of neighbors. However, the proposed query-biased temporal degree is a global metric and needs more complicated techniques to calculate it. Finally, the technologies of *CSM* are very hard to handle massive temporal networks. This is because their technologies are tailored to static networks. Even if a temporal network can be approximately transformed into a static network by existing methods, the size of the static network is often much larger than the original temporal network (e.g., [49]), resulting in prohibitively computational costs. However, our technologies are directly oriented to temporal networks which are very efficient as shown in our experiments. Besides, we have also empirically demonstrated the superiority of our approach by comparing it with *CSM* in terms of community quality (Section 6).

3.2 Query Drift Issue

Here, we want to prove that most existing methods may identify many temporal irrelevant vertices to the query vertex q for optimizing their objective functions. For simplicity, we assume that $f(.)$ is an objective function, and the larger the value of $f(C)$, the better the quality of the community C. Let $C^*(f)$ be any optimal community based on $f(.)$, and C_q be any community containing q.

Definition 3.1. **[Query-drifted vertices]** Given an objective function $f(.)$, we say $C^*(f) - C_q \neq \emptyset$ is *query-drifted vertices* and $f(.)$ suffers from the *query drift* issue iff the following two conditions hold: (i) $f(C^*(f) \cup C_q) \geq f(C_q)$; (ii) $\min\{\rho_{C^*(f) \cup C_q}(u)|u \in C^*(f) \cup C_q\} \leq \min\{\rho_{C_q}(u)|u \in C_q\}$.

By Definition 3.1, we know that adding *query-drifted vertices* $C^*(f) - C_q$ to C_q can improve its objective function score (i.e., condition (i)), but reduce the query-biased temporal degree (i.e., condition (ii)). In other words, if an objective function $f(.)$ finds many temporal irrelevant vertices to the query vertex (i.e., condition (ii)) for optimizing $f(.)$ (i.e., condition (i)), then we say that $f(.)$ suffers from the *query drift* issue.

Remark. Surprisingly, the condition (i) of Definition 3.1 is also called the free rider issue, which has been widely considered in static community search [18, 52]. Specifically, if an objective function $f(.)$ has the free rider issue (i.e., condition (i)), $f(.)$-based community search methods tend to include some redundant vertices (e.g., $C^*(f) - C_q$) in the detected community. However, the free rider issue cannot measure the temporal proximity between the query vertex and the redundant vertices. Thus, we introduce condition (ii) to further measure how these redundant vertices affect the temporal proximity of the detected community. As a result, our proposed *query drift* issue is more strict than the free rider issue. That is, if $f(.)$ suffers from the *query drift* issue, then $f(.)$ must have the free rider issue, and vice versa is not necessarily true.

Proposition 3.2. *Given a temporal graph $\mathcal{G}$ and a query vertex q, QTCS does not suffer from the query drift issue.*

Proposition 3.3. *Given a temporal graph $\mathcal{G}$ and a query vertex q, [4, 12, 23, 35] suffer from the query drift issue.*

4 EXACT GREEDY REMOVING FOR *QTCS*

In this section, we devise an exact greedy removing algorithm *EGR* to address our problem *QTCS*. The main idea of *EGR* is first to calculate the *TPPR* of each vertex and then greedily remove the vertices with the minimum query-biased temporal degree.

4.1 Edge Stream For *TPPR* Computation

We focus on calculating the *TPPR* of every vertex. Straightforwardly, we can use the push-based methods [1, 47] or power iteration-based methods [34, 50] to solve Equation 3. However, such methods have a high time overhead when handling temporal networks. The reasons are as follows. First, the time complexity of these methods is $O(MN)$ [50], in which M is the number of non-zero elements in the state transition matrix and N is the number of iterations. For temporal graphs, since each state in our model is an *ordered* temporal edge instead of a vertex, $M = O(m^2)$ (m is the number of temporal edges). So, the worst-case time complexity of existing methods is O(m^2N). Second, the state transition graph has a highly imbalanced weighted distribution. As a result, they would spend significant time on computing a small probability mass, resulting in high time overhead in practice. So, the main technical contribution of this section is to design efficient algorithms for computing *TPPR*.

Definition 4.1. **[l-hop temporal walk]** A l-hop temporal walk from vertex i to vertex j is ordered temporal edges $\{\vec{e}_1, \vec{e}_2, ..., \vec{e}_l\}$, satisfying $head(\vec{e}_1) = i$, $tail(\vec{e}_l) = j$, $tail(\vec{e}_i) = head(\vec{e}_{i+1})$ and $time(\vec{e}_i) \leq time(\vec{e}_{i+1})$ for all $1 \leq i \leq l-1$. For simplicity, we denote tw_l and $TW_l^{u \rightsquigarrow v}$ as the l-hop temporal walk and the set of l-hop temporal walk from u to v, respectively.

Definition 4.2. **[l-hop temporal transition probability]** Given a l-hop temporal walk $tw_l = \{\vec{e}_1, \vec{e}_2, ..., \vec{e}_l\}$, the l-hop temporal transition probability of tw_l, denoted by $P(tw_l)$, is $P(tw_l) = P(\vec{e}_1 \rightarrow \vec{e}_2) * P(\vec{e}_2 \rightarrow \vec{e}_3) * ... * P(\vec{e}_{l-1} \rightarrow \vec{e}_l)$. For completeness, we set $P(tw_0) = 0$, $P(tw_1) = 1/|\vec{e}_u^{out}|$ if $tw_1 = \{< u, v, t >\}$.

Lemma 4.3. *Given a temporal graph $\mathcal{G}(V, \mathcal{E})$, a query vertex q and a teleportation probability α, we have $tppr(u) = \sum_{i=0}^{\infty} \alpha(1-\alpha)^i \sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u}} P(tw_{i+1})$.*

Table 1: Comparison of *PPR* [1], *TPP*[40], and our *TPPR*

	Graph	State unit	State transition graph
PPR **[1]**	Static	Node	Cyclic
TPP**[40]**	Temporal	Node	Acyclic
TPPR	Temporal	*Ordered* temporal edge	**Acyclic (self-loops are allowed)**

Remark. Rozenshtein et al. [40] proposed temporal Personalized PageRank (*TPP*) $tpp_q(u)$ to model the temporal proximity of u w.r.t q. Specifically, $tpp_q(u) = \sum_{i=0}^{t(\mathcal{G})} \alpha(1-\alpha)^i \frac{Pr_i(q \rightsquigarrow u)}{\sum_{v \in V} Pr_i(q \rightsquigarrow v)}$, in which $t(\mathcal{G})$ is the maximum timestamp of $\mathcal{G}$ and $Pr_i(q \rightsquigarrow u) = \sum_{tw_i \in TW_i^{q \rightsquigarrow u}} P^{TPP}(tw_i)$ ($P^{TPP}(.)$ is the temporal transition probability of [40]). Our *TPPR* differs from *TPP* in two ways. Firstly, *TPP* limits the length of the temporal walk to at most $t(\mathcal{G})$. Thus, in *TPP*, the temporal edges associated with $t(\mathcal{G})$ do not have any chance to connect with other temporal edges, which contradicts the original definition of Personalized PageRank. However, our *TPPR* allows the length of the temporal walk to be infinite, as stated in Lemma 4.3. Thus, *TPPR* can capture more structure and temporal information than *TPP*. Secondly, the temporal transition probability of *TPP* (i.e., $\frac{Pr_i(q \rightsquigarrow u)}{\sum_{v \in V} Pr_i(q \rightsquigarrow v)}$) is differ from our *TPPR* (i.e., $\sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u}} P(tw_{i+1})$). However, the former takes more time to calculate because it needs to consider all cases where q walks to the remaining nodes (i.e., the denominator part). Besides, Table 1 summaries the differences among *PPR* [1], *TPP*[40], and our *TPPR*. We know that a striking feature of our proposed *TPPR* is that the state transition graph is acyclic (due to temporal walks occurring in chronological order), which helps us to design efficient algorithms. Note that the self-loops only appear in dangling states.

A failed attempt. By Lemma 4.3, a naive solution is first to enumerate all temporal walks from q to any vertex u. Then, it computes the l-hop temporal transition probability from q to u by previous temporal walks, and finally obtains $tppr(u)$ by Lemma 4.3. Unfortunately, it is impossible to calculate exactly the $tppr(u)$ as the summation goes to infinity. So, it is very challenging to directly apply Lemma 4.3 to compute $tppr(u)$. To tackle this challenge, we observe that the state transition graph of a temporal graph is acyclic (self-loops are allowed, see Definition 2.2 or Table 1), leading any temporal walk has a finite length unless it hits a self-loop state (i.e., dangling state). We state the observation as the following lemma.

Lemma 4.4. *By Definition 2.2 and 4.2, for $tw_\infty = \{\vec{e}_1, \vec{e}_2, ...\}$, we observe that $P(tw_\infty) \neq 0$ iff there is an integer l such that (1) $time(\vec{e}_i) < time(\vec{e}_{i+1})$ and $\vec{e}_i$ is not a dangling state for $1 \leq i \leq l-1$; (2) $\vec{e}_l$ is a dangling state and $\vec{e}_l = \vec{e}_{l+k}$ for any integer k.*

We denote a α-discount temporal walk of q as follows: (1) it starts from an order temporal edge $\vec{e}$ sampled from the distribution $\widetilde{\chi_q}$; (2) at each step it stops in the current order temporal edge with probability α, or it continues to walk according to Equation 2 with probability 1-α. Let u^t be any ordered temporal edge $\vec{e}$ with $tail(\vec{e}) = u$ and $time(\vec{e}) = t$. Let $D[u][t]$ be the probability that a α-discount temporal walk stops in u^t given the α-discount temporal walk at most one dangling state u^t if any.

Algorithm 1 *Compute_tppr* $(\mathcal{G}, q, \alpha)$

```
Input: temporal graph $\mathcal{G}$; query vertex $q$; teleportation probability $\alpha$
Output: the TPPR for every vertex.
1:  $tppr(u) \leftarrow 0$, $D[u] \leftarrow \{\}$ for any $u \in V$
2:  for $(u, v, t)$ in the edge stream of $\mathcal{G}$ do
3:      for $t_1 \in D[u]$ do
4:          $D[v][t] = D[v][t] + \frac{D[u][t_1]}{\alpha}(1-\alpha)P(u^{t_1} \rightarrow < u, v, t >)\alpha$
5:      if $u == q$ then
6:          $D[v][t] = D[v][t] + \frac{\alpha}{|\vec{e}_q^{out}|}$
7:      for $t_2 \in D[v]$ do
8:          $D[u][t] = D[u][t] + \frac{D[v][t_2]}{\alpha}(1-\alpha)P(v^{t_2} \rightarrow < v, u, t >)\alpha$
9:      if $v == q$ then
10:         $D[u][t] = D[u][t] + \frac{\alpha}{|\vec{e}_q^{out}|}$
11: for $u \in D$ do
12:     for $t \in D[u]$ do
13:         if $u^t$ is a dangling state then
14:             $D[u[t] = D[u][t]/\alpha$
15:         $tppr[u] = tppr[u] + D[u][t]$
16: return tppr
```

Lemma 4.5. *Given a temporal graph $\mathcal{G}(V, \mathcal{E})$, a query vertex q, and a teleportation probability α, we have $tppr(u) = \sum_{t \in T_1} D[u][t] + \sum_{t \in T_2} D[u][t]/\alpha$, in which $T_1 = \{t|u^t$ is not a dangling state$\}$ and $T_2 = \{t|u^t$ is a dangling state$\}$.*

Implication of Lemma 4.5. We can interpret *tppr*(u) as the sum of probabilities that α-discount temporal walk stops in u^t with different t. In particular, when u^t is a dangling state, the α-discount temporal walk can be of infinite length and form a geometric sequence with $(1 - \alpha)$ common ratio. As a result, the termination probability of α-discount temporal random walks with infinite length can be calculated directly without simulating random walks.

Inspired by the above theoretical observations, we devise an efficient dynamic programming approach (Algorithm 1) to compute *TPPR* with one pass over all temporal edges. Algorithm 1 first initializes *tppr*(u) as 0 and $D[u]$ as a dictionary structure for every vertex $u \in V$ (Line 1). In Line 2, we represent the temporal graph as an edge stream to ensure the time of temporal edges is non-decreasing, which can facilitate the $D[u][t]$ calculation. Thus, for each temporal edge (u, v, t), we update the dictionary structures $D[u][t]$ and $D[v][t]$ accordingly (Lines 3-10). So, the *TPPR* of u is the sum of $D[u][t]$ for different t by Lemma 4.5 (Lines 11-15).

Example 4.6. Consider the edge stream representation in Figure 2(c). Let v_5 be the query vertex and $\alpha = 0.2$, after the edge set $\{(v_1, v_2, 1), (v_2, v_3, 1), (v_1, v_3, 2), (v_3, v_4, 2), (v_4, v_6, 3)\}$ is traversed in Line 2 of Algorithm 1, D is still empty. This is because the edge set does not contain the query vertex v_5. Then, $\{(v_5, v_4, 3), (v_5, v_6, 3)\}$ is traversed, $D[v_4][3] = D[v_6][3] = \frac{\alpha}{|\vec{e}_{v_5}^{out}|} = \frac{0.2}{3}$ by Lines 2-10. Finally, $\{(v_1, v_4, 4), (v_5, v_6, 4)\}$ is traversed, $D[v_1][4] = \frac{D[v_4][3]}{\alpha}(1-\alpha)P(v_4^3 \rightarrow < v_4, v_1, 4 >)\alpha = \frac{\alpha}{|\vec{e}_{v_5}^{out}|} = \frac{0.16}{3}$, $D[v_6][4] = \frac{\alpha}{|\vec{e}_{v_5}^{out}|} = \frac{0.2}{3}$, and $D[v_5][4] = \frac{D[v_6][3]}{\alpha}(1-\alpha)P(v_6^3 \rightarrow < v_6, v_5, 4 >)\alpha = \frac{\alpha}{|\vec{e}_{v_5}^{out}|} = \frac{0.16}{3}$ by Lines 2-10. In Lines 11-15, since v_1^4, v_5^4, and v_6^4 are dangling states, $D[v_1][4] = \frac{D[v_1][4]}{\alpha} = \frac{0.8}{3}$, $D[v_5][4] = \frac{D[v_5][4]}{\alpha} = \frac{0.8}{3}$, and $D[v_6][4] = \frac{D[v_6][4]}{\alpha} = \frac{1}{3}$. So, $tppr[v_1] = tppr[v_5] = \frac{0.8}{3}$, $tppr[v_2] = tppr[v_3] = 0$, $tppr[v_4] = \frac{0.2}{3}$, $tppr[v_6] = \frac{0.2}{3} + \frac{1}{3} = \frac{1.2}{3}$.

Algorithm 2 *EGR* $(\mathcal{G}, q, \alpha)$

```
Input: temporal graph $\mathcal{G}$; query vertex $q$; teleportation probability $\alpha$
Output: the exact QTCS
1:  $tppr \leftarrow$ Compute_tppr $(\mathcal{G}, q, \alpha)$
2:  $temp \leftarrow V; R \leftarrow V; \beta^* \leftarrow 0$
3:  $\rho(u) \leftarrow \sum_{v \in N_V(u)} tppr(v)$ for each vertex $u \in V$.
4:  while $temp \neq \emptyset$ do
5:      $u \leftarrow \arg\min\{\rho(u)|u \in temp\}$
6:      if $u == q$ then
7:          break
8:      if $\rho(u) \geq \beta^*$ then
9:          $R \leftarrow temp; \beta^* \leftarrow \rho(u)$
10:     $temp \leftarrow temp \setminus \{u\}$
11:     for $v \in N_V(u) \cap temp$ do
12:         $\rho(v)$=$\rho(v) - tppr(u)$
13: return $CC(R, q)$, in which $CC(R, q)$ is the vertex set from the maximal connected component of $G_R$ containing $q$
```

Theorem 4.7. *Algorithm 1 can compute TPPR for each vertex. The time complexity of Algorithm 1 is $O(\mathcal{T}_{max} \cdot (m + n))$, where $\mathcal{T}_{max} = \max\{\mathcal{T}_u | u \in V\}$, $\mathcal{T}_u = |\{t|(u, v, t) \in \mathcal{E}\}|$.*

4.2 The *EGR* Algorithm

Below, we show that the query-biased temporal degree satisfies a monotonic property, which supports an exact greedy removing algorithm to solve our problem.

Lemma 4.8. ***[Monotonic property]*** *Given two vertex sets S and H and $S \subseteq H$, we have $\rho_S(u) \leq \rho_H(u)$ for any vertex $u \in S$ holds.*

By Lemma 4.8, we know that the larger the vertex set, the greater the query-biased temporal degree of vertex u. Inspired by this, we devise an exact greedy removing algorithm called *EGR* (Algorithm 2). Algorithm 2 first calls Algorithm 1 to calculate *TPPR* of every vertex (Line 1). Then, it initializes the current search space *temp* as V, candidate result R as V, the optimal value β^* of *QTCS* as 0, and the query-biased temporal degree $\rho(u)$ for every vertex $u \in V$ according to Definition 2.5 (Lines 2-3). Subsequently, it executes the greedy removing process in each round to improve the quality of the target community (Lines 4-12). Specifically, in each round, it obtains one vertex u with the minimum query-biased temporal degree (Line 5). Lines 8-12 update the candidate result R, the optimal value β^*, the search space *temp*, and the query-biased temporal degree. The iteration terminates once the current search space is empty (Line 4) or the query vertex q is removed (Lines 6-7). Finally, Line 13 returns $CC(R, q)$ as the exact *query-centered* temporal community.

Example 4.9. Consider the temporal graph in Figure 2(a). Let v_5 be the query vertex and $\alpha = 0.2$, the ordering $(v_1, v_2, v_3, v_6, v_5, v_4)$ is the order of vertices selected in Line 5 of Algorithm 2, which is illustrated in Figure 3. This is because that v_1 has the minimum query-biased temporal degree (i.e., 0.07) in $\{v_1, v_2, v_3, v_6, v_5, v_4\}$. Similarly, v_2 has the minimum query-biased temporal degree (i.e., 0) in $\{v_2, v_3, v_6, v_5, v_4\}$. Following this ordering, we can derive that (v_4, v_5, v_6) is the resultant community returned by Algorithm 2. This is because the minimum query-biased temporal degree in (v_4, v_5, v_6) is 0.34, which is the maximum value among all subgraphs generated by Algorithm 2 (Figure 3).

Theorem 4.10. *Algorithm 2 identify the exact* QTCS. *The time complexity and space complexity of Algorithm 2 are $(\mathcal{T}_{max} \cdot (m + n) + n \log n + \bar{m})$ and $O(\mathcal{T}_{max} \cdot n + m)$, respectively.*

In most real-life temporal graphs, $n \log n \leq m$ and $\bar{m} \leq m$ as stated in Section 6. Thus, the time complexity of Algorithm 2 can

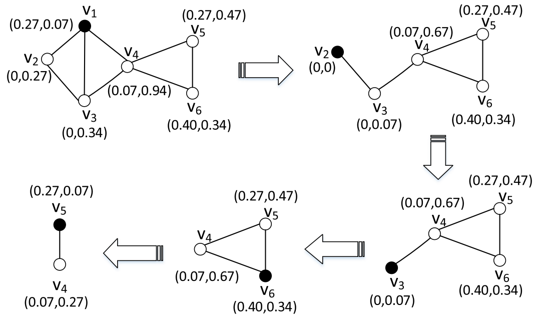


Figure 3: Illustration of Example 4.9 for the *EGR* algorithm. The black circle refers to the currently deleted vertices. The numbers in parentheses next to each vertex correspond to the *tppr* value (obtained by Example 4.6) and query-biased temporal degree, respectively.

be further reduced to $O(\mathcal{T}_{max} \cdot m)$. Moreover, Algorithm 2 is even near-linear in practice because $\mathcal{T}_{max}$ is usually small (Section 6). Clearly, the time complexity of *QTCS* is $\Omega(m)$ because it has to visit the whole graph at least once to calculate the exact *TPPR* of each vertex. Therefore, Algorithm 2 is nearly optimal.

5 APPROXIMATE LOCAL SEARCH FOR *QTCS*

In this section, we develop an approximate two-stage local search algorithm named *ALS* for solving our problem *QTCS*. *ALS* adopts the expanding and reducing paradigm. The expanding stage estimates the *TPPR* for some vertices, which essentially reduces unnecessary computation. Besides, it also obtains a small vertex set (say C) covering all target community members with theoretical guarantees. The reducing stage identifies an approximate solution directly from C instead of the original large graph, reducing the search space.

5.1 The Expanding Stage

Inspired by the problem of estimating *PPR* [1], we devise a local expanding algorithm. Before proceeding further, we briefly review the simple but efficient algorithm named *Forward_Push* proposed by Andersen et.al [1]. *Forward_Push* starts from the source state s and propagates information. The procedure iteratively updates two variables for each state v: its reserve $\pi(s, v)$ and residue $r(s, v)$. $\pi(s, v)$ indicates the approximate *PPR* value of v w.r.t. s and $r(s, v)$ indicates the information that will be propagated to other states from state v. In each iteration, for each state v that needs to propagate information, *Forward_Push* propagates $\alpha r(s, v)$ to $\pi(s, v)$ and the remaining $(1 - \alpha)r(s, v)$ is propagated along its neighbors. After finishing the propagation, *Forward_Push* sets $r(s, v)$ to zero. *Forward_Push* has the following equation [1].

$$PPR(s, v) = \pi(s, v) + \sum_{w} r(s, w)PPR(w, v) \tag{4}$$

Where $PPR(s, v)$ (resp. $PPR(w, v)$) is the *PPR* value of v w.r.t. s (resp. w). Our proposed expanding stage is built upon *Forward_Push*, but incorporates more novel strategies to adapt to ordered temporal edges (because each state in *TPPR* is an ordered temporal edge instead of a vertex). We first propose one key sub-algorithm in Algorithm 3, which will be invoked later to estimate the *TPPR* for some vertices. The process is similar to *Forward_Push*, except that the propagation is executed on *ordered* temporal edges.

Algorithm 3 *Propagation*($\vec{e}$)

```
if $r(\vec{e}) \geq 1/m$ then
    for each $\vec{e}_1 \in N^{>}(\vec{e})$ do
        $r(\vec{e}_1) \leftarrow r(\vec{e}_1) + (1-\alpha)r(\vec{e})P(\vec{e} \rightarrow \vec{e}_1)$
    $\pi(\vec{e}) \leftarrow \pi(\vec{e}) + \alpha r(\vec{e})$, $\widehat{tppr}(tail(\vec{e})) \leftarrow \widehat{tppr}(tail(\vec{e})) + \alpha r(\vec{e})$
    $r(\vec{e}) \leftarrow 0$
```

Algorithm 4 *Expanding* $(\mathcal{G}, q, \alpha)$

```
Input: temporal graph $\mathcal{G}$; query vertex $q$; teleportation probability $\alpha$
Output: expanded vertex set $C$, $r$ and $\widehat{tppr}$
1:  $r \leftarrow \{\}$; $\pi \leftarrow \{\}$; $\widehat{tppr} \leftarrow \{\}$
2:  $r(\vec{e}) \leftarrow 1/|\vec{e}_q^{\,out}|$ for all $\vec{e} \in \vec{e}_q^{\,out}$
3:  $C \leftarrow \emptyset$; $\widehat{\beta} \leftarrow 0$; $Q \leftarrow \{q\}$; $D \leftarrow \{q\}$
4:  while $Q \neq \emptyset$ do
5:      $u \leftarrow Q.pop()$; $C \leftarrow C \cup \{u\}$
6:      for $\vec{e} \in \vec{e}_u^{\,out}$ do
7:          $Propagation(\vec{e})$
8:      if $\min\{\sum_{v \in N_C(w)} \widehat{tppr}(v) | w \in C\} > \widehat{\beta}$ then
9:          $\widehat{\beta} \leftarrow \min\{\sum_{v \in N_C(w)} \widehat{tppr}(v) | w \in C\}$
10:     for $v \in N_V(u)$ and $v \notin D$ do
11:         $D \leftarrow D \cup \{v\}$
12:         if $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in N_V(v)} \widehat{tppr}(w) \geq \widehat{\beta}$ then
13:             $Q.push(v)$
14:     if $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in Q} \widehat{tppr}(w) < \widehat{\beta}$ then
15:         $C \leftarrow C \cup Q$
16:         break
17: return $C$, $r$ and $\widehat{tppr}$
```

Lemma 5.1. *For any vertex set H and any vertex $u \in H$, we have $\sum_{v \in N_H(u)} \sum_{\vec{e}_i \in \vec{e}_v^{\,in}} \pi(\vec{e}_i) \leq \rho_H(u) \leq \sum_{v \in N_H(u)} \sum_{\vec{e}_i \in \vec{e}_v^{\,in}} \pi(\vec{e}_i) + \sum_{\vec{e}} r(\vec{e})$.*

Implication of Lemma 5.1. The lemma indicates that the additive errors of $\rho_H(u)$ are negligible when the residue sum $\sum_{\vec{e}} r(\vec{e})$ is small enough. Therefore, we set $r(\vec{e}) \geq 1/m$ in Algorithm 3 to speed up the propagation, resulting in $\sum_{\vec{e}} r(\vec{e})$ decreases rapidly.

Based on Lemma 5.1, we present powerful pruning techniques to delete some unqualified vertices or terminate the expanding stage with theoretical guarantees. We denote C as the expanded vertex set for the following reducing stage, Q as the candidate vertices that are neighbors of C and not in C, $\widehat{\beta}$ as the best estimate of the minimum query-biased temporal degree so far, D as the visited vertices to avoid repeated visits. Let $\widehat{tppr}(v) = \sum_{\vec{e}_i \in \vec{e}_v^{\,in}} \pi(\vec{e}_i)$ be the lower bound of *TPPR* for vertex v by Lemma 5.1.

Lemma 5.2. ***[bound-based pruning]*** *For a vertex v, we can prune the vertex v if $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in N_V(v)} \widehat{tppr}(w) < \widehat{\beta}$.*

Lemma 5.3. ***[stop expanding-I]*** *Given the currently expanded vertices C and candidate vertices Q, we can safely terminate the expanding stage if $Q = \emptyset$.*

Lemma 5.4. ***[stop expanding-II]*** *Given the current expanded vertices C and candidate vertices Q, we can set $C = C \cup Q$ and safely terminate the expanding stage if $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in Q} \widehat{tppr}(w) < \widehat{\beta}$.*

With these powerful pruning techniques, we introduce Algorithm 4 to implement the expanding stage. Specifically, in Lines 1-2, the algorithm first initializes r and π for ordered temporal edges, which are used to estimate the query-biased temporal degree (Lemma 5.1). In Lines 4-16, it executes the expanding process. In particular, it pops a vertex u from queue Q to execute the propagation process and adds u into the expanded vertex set C (Lines 5-7). After the propagation, it updates the estimate of the minimum

Algorithm 5 *Reducing* $(C, r, \widehat{tppr}, q, \alpha)$

```
Input: expanded vertex set $C$, $r$ and $\widehat{tppr}$ from Algorithm 4; query vertex $q$; teleportation probability $\alpha$
Output: the $\epsilon$-approximate QTCS
1: $R \leftarrow C; \widehat{\rho} \leftarrow \{\}; flag \leftarrow True$
2: for $u \in C$ do
3:     $\widehat{\rho}(u) \leftarrow \sum_{v \in N_C(u)} \widehat{tppr}(v)$
4: $temp \leftarrow \max\{\widehat{\rho}(u) | u \in C\} + \sum_{\vec{e}} r(\vec{e}); \overline{\epsilon} \leftarrow \frac{temp}{\min\{\widehat{\rho}(u)|u \in C\}}$
5: while $flag$ do
6:     $Q \leftarrow \emptyset; D \leftarrow \emptyset$
7:     for $u \in R$ do
8:         if $\overline{\epsilon}\widehat{\rho}(u) \leq temp$ then
9:             $Q.push(u)$
10:            if $u == q$ then
11:                $flag \leftarrow False; Q \leftarrow \emptyset$
12:    while $Q \neq \emptyset$ do
13:        $u \leftarrow Q.pop$ and $D \leftarrow D \cup \{u\}$
14:        for $v \in N_R(u)$ and $v \notin D$ do
15:            $\widehat{\rho}(v) = \widehat{\rho}(v) - \widehat{tppr}(u)$
16:            if $\overline{\epsilon}\widehat{\rho}(v) \leq temp$ then
17:                $Q.push(v)$
18:                if $v == q$ then
19:                    $flag \leftarrow False; Q \leftarrow \emptyset$
20:    if $flag$ then
21:        $\epsilon \leftarrow \overline{\epsilon}; R \leftarrow R \setminus D; \overline{\epsilon} \leftarrow \overline{\epsilon}/2$
22: return $(\epsilon, CC(R, q))$, in which $CC(R, q)$ is the vertex set from the maximal connected component of $G_R$ containing $q$ and $\epsilon$ is the corresponding approximation ratio
```

query-biased temporal degree (Lines 8-9). In Lines 10-13, for each neighbor vertex v of u, it uses the bound-based pruning technique (Lemma 5.2) to remove unqualified vertices. Once the queue Q becomes the empty set or $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in Q} \widehat{tppr}(w) < \widehat{\beta}$, the algorithm stops expanding according to stop expanding pruning techniques in Lemma 5.3 and Lemma 5.4. Clearly, the vertex set C returned by Algorithm 4 covers all target community members.

THEOREM 5.5. *The time complexity and space complexity of Algorithm 4 are $O(\sum_{u \in C} \sum_{\vec{e} \in \vec{e}_u^{out}} |N^{>}(\vec{e})|)$ and $O(n + m)$ respectively.*

5.2 The Reducing Stage

In the reducing stage, we identify an approximate *query-centered* temporal community directly from the subset C found by the previous expanding stage. At a high level, this stage progressively removes the vertices in C that are not contained in the approximate solution. Until the remaining vertices meet the given approximation ratio. Choosing which vertices to remove is a significant challenge. Thus, we devise the following definition and lemma to guarantee the quality of the search.

Definition 5.6. For a vertex set H and $\epsilon \geq 1$, if $\min\{\rho_H(u) | u \in H\} \leq \beta^* \leq \epsilon \cdot \min\{\rho_H(u) | u \in H\}$, we say H is an ϵ-approximate *QTCS*, where β^* is the optimal value for *QTCS*.

LEMMA 5.7. *For the current search space R and $\epsilon \geq 1$, we can safely prune $u \in R$ without losing any ϵ-approximate QTCS if $\epsilon \cdot \sum_{v \in N_R(u)} \widehat{tppr}(v) < \max\{\sum_{w \in N_C(v)} \widehat{tppr}(w) | v \in C\} + \sum_{\vec{e}} r(\vec{e})$.*

Unfortunately, ϵ does not know in advance. Thus, to obtain a high-quality estimation error ϵ, we use a binary search to continuously refine ϵ. The idea of the reducing stage is outlined in Algorithm 5. Specifically, it first initializes the current search space R as vertex set C found by the previous expanding stage and the estimated query-biased temporal degree $\widehat{\rho}(u)$ by the lower bound of *TPPR* (Lines 1-3). Subsequently, in Line 4, it computes $\overline{\epsilon}$ as the upper bound of the approximation ratio. In Lines 5-21, it proceeds by continuously refining $\overline{\epsilon}$ and iteratively removing the unpromising vertices in each round to meet the current approximation ratio $\overline{\epsilon}$ by Lemma 5.7. In particular, in each round, it first initializes a queue Q to collect vertices to be deleted and a set D to maintain all deleted vertices (Line 6). Then it applies Lemma 5.7 to push those unpromising vertices into Q in Lines 7-9 and processes iteratively the vertices in Q to remove more unpromising vertices in Lines 12-17. The algorithm use a $flag$ to indicate whether query vertex q is removed or not. If the $flag$ is $True$, it updates the target approximation ratio ϵ, search space R and $\overline{\epsilon}$ (in Lines 20-21). The iteration terminates once query vertex q is removed. Finally, the algorithm returns $CC(R, q)$ as the ϵ-approximate *query-centered* temporal community (Line 22). Algorithm 5 can correctly find an ϵ-approximate *query-centered* temporal community based on Lemma 5.7.

THEOREM 5.8. *The time complexity and space complexity of Algorithm 5 are $O(|G_C| \log m)$ and $O(|G_C|)$ respectively, where $G_C = \{(u, v) \in E | u, v \in C\}$.*

6 EXPERIMENTAL EVALUATION

We conduct extensive experiments to evaluate our solutions. All experiments are executed on a server with an Intel (R) Xeon (R) E5-2680 v4@2.40GHZ CPU and 256GB RAM running Ubuntu 18.04.

6.1 Experimental setup

Datasets. We evaluate our solutions on eight graphs which are used in recent work [4, 23, 26, 27, 35, 36, 59] as benchmark datasets (Table 2). Reality Mining (Rmin for short), Lyonschool (Lyon), and Thiers13 (Thiers) are temporal face-to-face networks, in which a vertex represents a person, and a temporal edge indicates when the corresponding persons had physical contact. Facebook and Twitter are temporal social networks, in which vertices represent users and temporal edges indicate when they had online interactions. Lkml and Enron are temporal communication networks in which a vertex indicates an ID and a temporal edge signifies when the corresponding IDs had a message. DBLP is a temporal collaboration network, in which each temporal edge denotes when the authors coauthored a paper.

Algorithms. Several state-of-the-art baselines are complemented. Specifically, *CSM* [6] identifies the maximal k-core containing the query vertex with the largest k. *TCP* [15] applies the triangle connectivity and k-truss to model the higher-order truss community. *PPR_NIBBLE* [1] adopts the conductance as the criterion of a community and uses static Personalized PageRank to obtain the result. Note that *CSM*, *TCP*, and *PPR_NIBBLE* are static community search methods. *MPC* [35] extends the concept of clique to adapt to the temporal setting. *PCore*[23] maintains persistently a k-core structure. *DBS* [4] uses the density and duration to model bursting communities. But *MPC*, *PCore*, and *DBS* address the problem of temporal community detection. Thus, to fit our problem, we first find all possible communities by the predefined criteria[4, 23, 35], and then select the target community containing the query vertex from these communities. *MTIS* [46] and *MSCS* [12] are temporal community search methods. *MTIS* and *MSCS* model the temporal cohesiveness of the community by extending the network-inefficiency and k-core to the temporal setting, respectively. *TPP_CS* calculates the *TPP* values proposed by [40] to replace Line 1 of Algorithm 2. *QTCS_Baseline* is an intuitive variant model (Definition 2.4). *EGR* and *ALS* are our methods.

Effectiveness metrics. **Evaluating the utility of a temporal community is more difficult than a static community since there**

Table 2: **Dataset statistics.** TS **is the time scale of the timestamp**

Dataset	$\|V\|$	$\|\mathcal{E}\|$	$\|E\|$	$\mathcal{T}_{max}$	TS
Rmin	96	76,551	2,539	2,478	Hour
Lyon	242	218,503	26,594	20	Hour
Thiers	328	352,374	43,496	49	Hour
Facebook	45,813	585,743	183,412	552	Day
Twitter	304,198	464,653	452,202	7	Day
Lkml	26,885	547,660	159,996	2,663	Day
Enron	86,978	912,763	297,456	765	Day
DBLP	1,729,816	12,007,380	8,546,306	49	Year

are no ground-truth communities for temporal networks yet [23, 26, 27, 35, 36, 59]. However, we can adopt the following two widely used effectiveness metrics [4, 26, 27, 43, 59] for qualitative testing: temporal density (*TD*) and temporal conductance (*TC*). Specifically, let S be the target community, the two metrics are defined as follows. $TD(S) = 2*|\{(u,v,t) \in \mathcal{E}|u,v \in S\}|/|S|(|S|-1)|T_S|$, in which $T_S = \{t|(u,v,t) \in \mathcal{E}, u,v \in S\}$. Clearly, *TD* computes the average density of the internal structure of the temporal community. $TC(S) = |Tcut(S, V \setminus S)|/\min\{|Tvol(S)|, |Tvol(V \setminus S)|\}$, where $Tcut(S, V \setminus S) = \{(u,v,t) \in \mathcal{E}|u \in S, v \in V \setminus S\}$, $Tvol(S) = \sum_{u \in S}\{(u,v,t) \in \mathcal{E}\}$. Clearly, *TC* measures the separability of the temporal community. Thus, the larger the value of *TD*(*S*), the denser *S* is in the temporal network. The smaller the value of *TC*(*S*), the farther *S* is away from the rest of the temporal network. In addition, we also report the value of our proposed objective function. Let *MD*(*S*)=$\min\{\rho_C(u)|u \in S\}$ be the minimum query-biased temporal degree within *S*. So, the larger the value of *MD*(*S*), the better the quality of *S* in terms of *query-centered* temporal community search.

Parameters. Unless otherwise stated, the teleportation probability α is set to 0.2 in all experiments as [30, 48]. For other methods, we take their corresponding default parameters. To be more reliable, we randomly select 50 vertices as query vertices and report the average running time and quality.

6.2 Efficiency testing

Exp-1: Running time of various temporal methods. From Table 3, we can see that *ALS* is consistently faster than other methods on most datasets. For example, *ALS* takes 3.038 seconds and 191.889 seconds to obtain the result from Facebook and Lkml, respectively, while *PCore* and *MTIS* cannot get the result within two days. Moreover, our methods (i.e., *QTCS_Baseline*, *EGR*, and *ALS*) are more efficient than existing methods. The reasons can be explained as follows. (1) *MPC*, *PCore*, and *DBS* need to enumerate all possible temporal communities in advance and then select the target community containing the query vertex from these communities, resulting in very high time overheads. (2) *MTIS* and *MSCS* first perform the very time-consuming Steiner tree procedure to identify a tree T containing all query vertices, and then greedily add some desirable vertices to T to derive the final result. (3) *TPP_CS* takes more time to calculate the *TPP* values because it needs to consider all cases where q walks to the remaining nodes (see Table 1 and the corresponding *Remark* for details). Furthermore, *ALS* is faster than *EGR* on all datasets. For example, *ALS* only consumes about 13 seconds to identify the result from DBLP, while *EGR* consumes over 47 seconds. These results give some preliminary evidence that the proposed pruning strategies (Section 5) are efficient in practice.

Exp-2: Running time of various ***QTCS*** **algorithms with varying parameters.** In this experiment, we investigate how the parameter α affects the running time of different *QTCS* algorithms. Additionally, we also study the effect of the temporal occurrence rank of query vertices. Let $\mathcal{T}_u = |\{t|(u,v,t) \in \mathcal{E}\}|$ be the temporal occurrence of the vertex u, which indicates how many timestamps are associated with u. Thus, we denote the temporal occurrence rank of a vertex as 0.1 if its temporal occurrence is in the bottom 1%- 10%, and the temporal occurrence ranks 0.2, . . ., 0.9 are defined accordingly. For *EGR* algorithm, we know that the search time is composed of Algorithm 1 and the greedy removing process. We denote t(TPPR) as the time spent in Algorithm 1. Figure 4 (a-h) show the results with varying rank and α on Rmin, Facebook, Enron, and DBLP. Other datasets can also obtain similar results. As can be seen, t(TPPR) dominates the time of *EGR* on all datasets except for DBLP. This is because the size of DBLP is relatively large, so it needs more time to perform the greedy removing process. Moreover, as shown in Figure 4 (a-d), the running time decreases first and then increases as rank increases, and the optimal time is taken when rank=0.5. Thus, we recommend users set the vertex with rank 0.5 as the query vertex for faster performance. On the other hand, by Figure 4 (e-h), we know that the running time of *ALS* decreases with increasing α. An intuitive explanation is that when α increases, the vertices have a higher probability of running a temporal random walk around the query vertex, resulting in the locality of *ALS* being stronger. As a result, the techniques of bound-based pruning and stop expanding are enhanced with increasing α, thus more search spaces or vertices are pruned (Section 5.1). Note that the running time of t(TPPR) and *EGR* are stable with varying α. This is because the time complexity of t(TPPR) and *EGR* is independent of α.

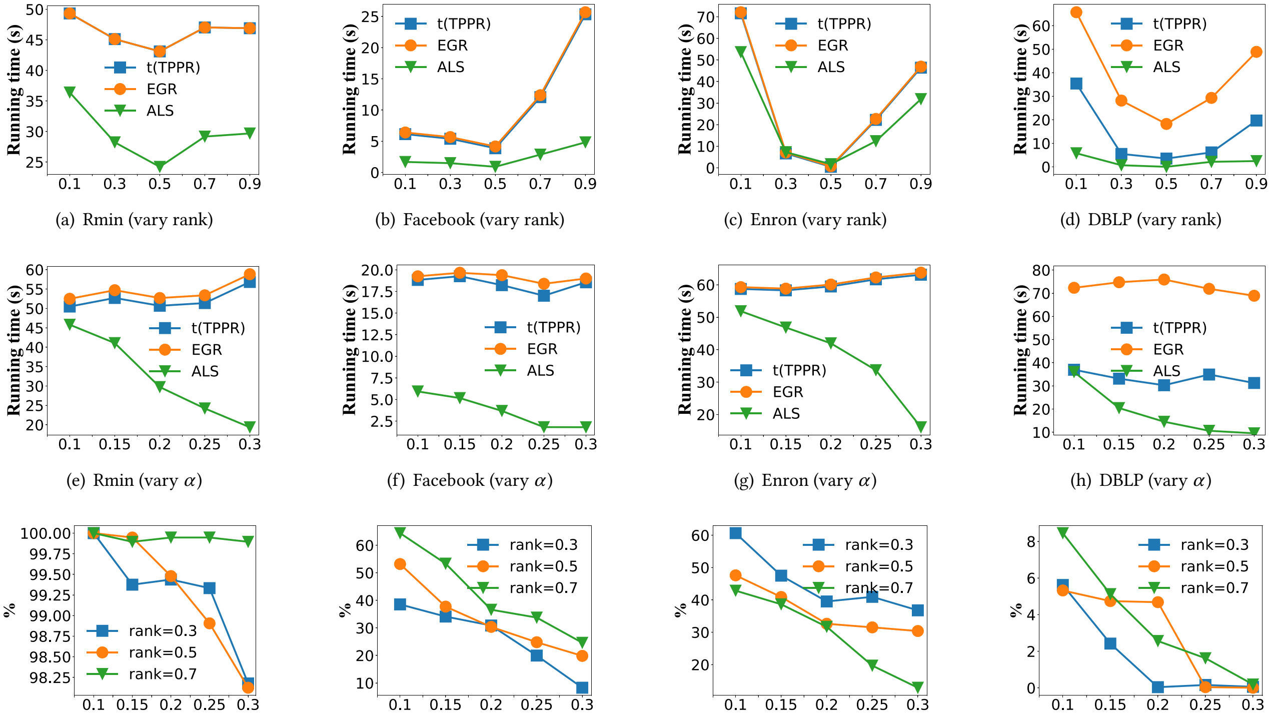
Exp-3: The size of the expanded graph with varying parameters. Figure 4 (i-l) shows the size of the expanded graph obtained by the expanding stage (i.e., $|C|$ in Section 5.1), divided by the size of the original graph, with varying rank and α. We can see that the expanding stage obtains a very small graph. For instance, on Enron and DBLP, the number of vertices obtained by the expanding stage are only about 35% and 4% of the original graph, respectively. And the size of the expanded graph decreases with increasing α. This is because the power of both bound-based pruning and stop expanding are enhanced when α increases. These results give some preliminary evidence that the proposed expanding algorithm (Section 5.1) is very effective when handling real-life temporal graphs. Moreover, we also observe that the size of the expanded graph is irregular as the rank increases.

Exp-4: Scalability testing on synthetic datasets. To test the scalability of *EGR* and *ALS*, we first artificially generate eight temporal subgraphs by selecting randomly 20%, 40%, 60% and 80% vertices or edges from DBLP. Subsequently, we test the runtime of *EGR* and *ALS* on these temporal subgraphs. Figure 5 shows the results. As can be seen, *EGR* and *ALS* scale near-linear w.r.t. the size of the temporal subgraphs. These results indicate that our proposed algorithms can handle massive temporal networks.

Exp-5: Memory overhead of ***EGR*** **and** ***ALS*****.** From Table 4, we can see that the memory overhead of *EGR* and *ALS* is less than twice that of the original graph. Moreover, we can also see that the memory overhead of *ALS* is less than *EGR* in six of the eight datasets. This is because *ALS* is a local search algorithm, thus fewer vertices may be visited (Exp-3 also confirms this), which further

Table 3: Running time of various temporal methods (second). AVG.RANK is the average rank of each method across testing datasets.

Temporal methods	Rmin	Lyon	Thiers	Facebook	Twitter	Lkml	Enron	DBLP	AVG.RANK
MPC	2133.440	6.153	59.746	3.987	1.318	47563.571	729.380	2605.572	5
PCore	35913.248	28561.989	>48h	>48h	148.447	>48h	21221.338	24.493	8
DBS	47.363	1722.200	2150.320	48.792	33179.300	**91.411**	614.998	2462.040	6
MTIS	>48h	42.339	154.161	>48h	152.064	>48h	>48h	78252.764	9
MSCS	241.613	25.204	28.786	753.186	42.699	859.255	1290.521	3083.327	7
TPP_CS	73.129	4.724	9.794	28.123	6.269	302.821	112.479	83.572	4
QTCS_Baseline	47.283	1.879	6.703	16.107	1.800	226.457	82.66	45.391	2
EGR	47.293	1.881	6.711	16.067	2.604	224.592	83.168	47.259	3
ALS	**28.326**	**1.030**	**3.049**	**3.038**	**1.257**	191.889	**30.557**	**13.707**	**1**



(a) Rmin (vary rank) (b) Facebook (vary rank) (c) Enron (vary rank) (d) DBLP (vary rank)

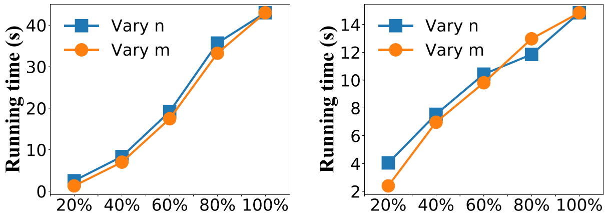
(e) Rmin (vary α) (f) Facebook (vary α) (g) Enron (vary α) (h) DBLP (vary α)

(i) Rmin (vary α) (j) Facebook (vary α) (k) Enron (vary α) (l) DBLP (vary α)

Figure 4: The efficiency of various algorithms with varying parameters

Table 4: Memory overhead of *EGR* and *ALS* (MB)

	Graph in memory	Memory of *EGR*	Memory of *ALS*
Rmin	9.291	12.871	16.669
Lyon	34.780	35.236	35.072
Thiers	62.381	63.917	63.430
Facebook	149.538	162.873	159.564
Twitter	311.206	393.152	331.207
Lkml	131.514	148.0143	182.439
Enron	244.577	272.900	247.764
DBLP	5190.925	5758.229	5302.925



(a) *EGR* (b) *ALS*

Figure 5: Scalability testing on synthetic datasets

results in less space used to store reserve and residue for estimating the *TPPR* values. However, *EGR* is a global algorithm, which needs to store $D[u]$ for computing the exact *TPPR* values. These results show that *EGR* and *ALS* can achieve near-linear space cost, which is consistent with our analysis in Section 4 and Section 5.

6.3 Effectiveness testing

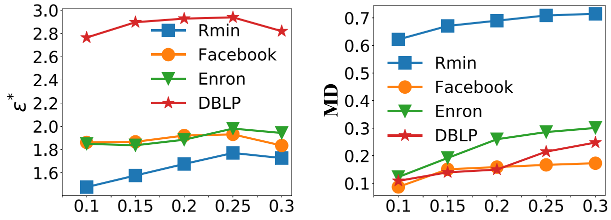
Exp-6: Effectiveness of different methods. Table 5 reports our results. For the *TC* metric, we have: (1) our model achieves the best scores on seven of the eight datasets. This is because our model can mitigate the *query drift* issue (Section 3.2), resulting in that it can keep good temporal separability by removing many temporal irrelevant vertices to the query vertex (i.e., *query-drifted vertices*). (2) *PPR_NIBBLE* and *QTCS_Baseline* are the runner-up and third-place, respectively, which shows that these random walk methods can also obtain better temporal separability. (3) *TPP_CS* is slightly better than static methods *CSM* and *TCP* but very worse than our model. This is because although *TPP_CS* also uses temporal walks

Table 5: Effectiveness of different methods. AVG.RANK is the average rank of each method across the testing datasets.

TC/TD/MD	Rmin	Lyon	Thiers	Facebook	Twitter	Lkml	Enron	DBLP	AVG.RANK
CSM	0.33/0/0.35	0.87/0.42/0.76	0.92/0.14/0.49	0.43/0.08/0	0.71/0.04/0	0.68/ 0.06/0.07	0.48/ 0.02/0	0.72/ 0.30/0.01	5/10/3
TCP	0.92/0/0.10	1/0.38/0.55	1/0.13/0.32	0.50/0.28/0.03	0.71/0.52/0.03	0.36/0.08/0	0.40/0.09/0	0.68/0.40/0	6/9/4
PPR_NIBBLE	0.48/0/0.07	0.50/0.51/0.28	0.44/0.17/0.17	0.17/0.01/0	**0.11**/0/0	0.07/0/0	**0.27**/0.01/0	0.09/0/0	2/11/10
MPC	0.71/**0.29**/0.03	0.79/0.76/0.13	0.82/**0.64**/0.02	0.50/**0.50**/0	1/**0.79**/0	0.96/**0.22**/0	0.94/**0.44**/0	0.84/**0.59**/0	10/**1**/9
PCore	0.75/0/0.24	0.55/0.52/0.30	0.62/0.58/0.11	0.72/0.09/0	0.94/0.03/0	0.76/0.02/0.11	0.76/0.06/0.04	0.60/0.08/0	8/4/6
DBS	0.66/0.18/0.21	0.72/**0.77**/0.18	0.52/0.56/0.07	0.67/0.41/0	0.95/0.66/0	0.95/0.21/0.15	0.92/0.33/0.09	0.70/0.43/0	9/2/8
MTIS	0.67/0.02/0.13	0.98/0.43/0.02	0.98/0.27/0	1/0.32/0	1/0.26/0	1/0/0	1/0/0	1/0/0	11/8/11
MSCS	0.53/0.08/0.38	0.58/0.54/0.49	0.31/0.29/0.54	0.72/0.18/0	0.72/0.12/0	0.72/0/0.01	0.59/0/0	0.60/0/0	7/7/7
TPP_CS	0.31/0.01/0.18	0.62/0.48/0.43	0.57/0.22/0.26	0.46/0.15/0.01	0.69/0.06/0.02	0.57/ 0.02/0.07	0.53/ 0.12/0.03	0.64/ 0.02/0	4/6/5
QTCS_Baseline	0.30/0.01/0.43	0.56/0.52/0.58	0.45/0.17/0.46	0.49/0.07/0	0.68/0/0	0.53/0.03/0.06	0.54/0.20/0.04	0.55/0.05/0	3/5/2
our model	**0.01**/0.18/**0.73**	**0.44**/0.73/**0.81**	**0.16**/0.56/**0.67**	**0.11**/0.46/**0.15**	**0.11**/0.57/**0.08**	**0.02**/0.20/**0.25**	0.32/0.33/**0.26**	**0.03**/0.40/**0.15**	**1**/3/**1**

Table 6: Quality comparison between *EGR* and *ALS*

	ϵ	ϵ^*	Precision	Recall	F1-Score
Rmin	3.350	1.657	0.646	0.984	0.780
Lyon	2.745	1.302	0.848	1.000	0.918
Thiers	3.439	1.489	0.772	1.000	0.871
Facebook	7.410	1.751	0.504	0.977	0.665
Twitter	5.160	1.584	0.266	0.983	0.419
Lkml	7.601	1.937	0.477	0.995	0.645
Enron	8.580	1.863	0.575	0.964	0.720
DBLP	13.024	3.279	0.224	0.950	0.362



(a) ϵ^* (vary α)

(b) *MD* (vary α)

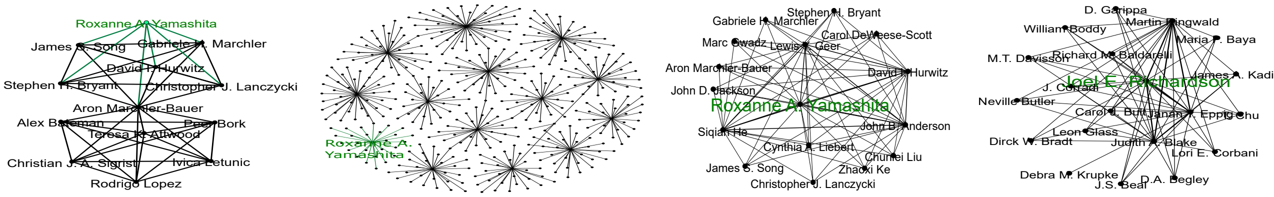
Figure 6: The quality of *ALS* with various α.

to measure the temporal proximity, but it limits the length of these walks to $t(\mathcal{G})$ at most, resulting in temporal information not being fully utilized and performance degradation. (4) *MPC*, *PCore*, *DBS*, *MTIS*, and *MSCS* have the worst performance. This is because they focus on internal temporal cohesiveness but ignore the separability from the outside. For the *TD* metric, we have: (1) *MPC* and *DBS* outperform other methods (but they have poor *TC*), and our model is the third-place and slightly worse than *MPC* and *DBS*. This is because *MPC* and *DBS* respectively adopt the clique and density as the criteria of the community, which has a strong density in itself. (2) *CSM*, *TCP*, and *PPR_NIBBLE* have the worst performance. This is because they are static methods that ignore the temporal dimension of the graph. For the *MD* metric, we have: (1) our model achieves the best scores on all datasets while other models are almost zero on large datasets. (2) The gap between other models and our model is smaller on small datasets (i.e., Rmin, Lyon, and Thiers) than on large datasets. These results indicate that baselines cannot optimize our proposed objective function well, and our model is much denser and more separable in terms of temporal features than baselines.

Exp-7: Quality comparison between *EGR* and *ALS*. Here, we compare the community identified by the approximate local search algorithm *ALS* with that identified by the exact greedy removing algorithm *EGR*. Specifically, we use the community derived by *EGR* as the ground-truth for evaluating the quality of *ALS*. Table 6 reports the results. Here, ϵ is the theoretical approximation ratio of *ALS* (Algorithm 5) and $\epsilon^* = \min\{\rho_{H_1}(u)|u \in H_1\}/\min\{\rho_{H_2}(u)|u \in H_2\}$ is the *true* approximation ratio, where H_1 and H_2 are the communities identified by *EGR* and *ALS*, respectively. We have the following observations. (1) *ALS* obtains better results than the theoretical ϵ-approximation ratio. In particular, the *true* approximate ratio of *ALS* is between 1 and 4. (2) *ALS* obtains a good recall value, which indicates the community found by *ALS* covers almost all members of the ground-truth. (3) *ALS* obtains relatively high scores of precision and F1-Score, which implies the size of the community returned by *ALS* is close to the ground-truth. In summary, the approximate algorithm *ALS* can find high-quality communities in practice.

Exp-8: The quality of *ALS* with various α. Figure 6 shows the *true* approximation ratio ϵ^* and the minimum query-biased temporal degree *MD* with various α. Due to the space limit, we only report the results on Rmin, Facebook, Enron, and DBLP. Other datasets can also obtain similar results. As shown in Figure 6(a), ϵ^* increases first and then decreases as α increases. The reasons are: (1) when α is small, the target community is closer to the query vertex, and the locality of *ALS* is stronger. As a result, the community found by *ALS* matches the target community. (2) When α is large, the target community may be very small. Thus, once the community identified by *ALS* is slightly different from the target community, it will cause ϵ^* to drop rapidly. From Figure 6(b), we can observe that *MD* increases with increasing α. This is because when α increases, the *TPPR* value tends to be concentrated near the query vertex and these *TPPR* values are large, which leads to a larger *MD* by Definition 2.5.

Exp-9: Case studies on DBLP. Here, we further show that our model can eliminate the *query drift* issue (Section 3.2) while other models cannot eliminate it. Due to the space limit, we mainly report the results on *PCore*, *MSCS*, *QTCS_Baseline*, and our model *QTCS*. Similar results can also be obtained by the other models. Specifically, we choose Prof. Roxanne A. Yamashita or Joel E. Richardson as the query vertex. The community identified by *QTCS_Baseline* contains more than 1,000 authors (since it is too large to show in a figure, we do not visualize the community) that come from diverse research domains. This is because *QTCS_Baseline* considers structural cohesiveness and temporal proximity separately, which forces the result to include many vertices with poor temporal proximity to satisfy the structural cohesiveness. As shown in Figure 7 (c) (the timestamps of edges are ignored for visualizing), the community obtained by *QTCS* is a meaningful *query-centered* temporal community and does not cause the *query drift* issue. This is because Roxanne A. Yamashita has many temporal walks with small intervals to other researchers (see the temporal information in http://snap.stanford.edu/). Besides, these researchers worked closely and frequently with Roxanne A. Yamashita in conserved



(a) *PCore* (b) *MSCS* (c) *QTCS* (d) *QTCS*

Figure 7: Case studies on DBLP. (a-c) (resp. (d)) are the communities of Prof. Roxanne A. Yamashita (resp. Joel E. Richardson)

sequence, amino acid sequence, and proteins during 2015-2021 (see their homepages for details). Thus, we can explain that this community is formed by their shared research interests and long-term cooperation with Roxanne A. Yamashita. However, from Figure 7 (a), we can see that the members on the upper and lower parts are connected by the hub vertex Aron Marchler-Bauer. Besides, Roxanne A. Yamashita has very few temporal walks to the lower part (see the temporal information at http://snap.stanford.edu/). Thus, the lower part is *query-drifted vertices*. Additionally, by looking at the homepages of these researchers, we find that they come from different research backgrounds. Moreover, several important collaborators of Roxanne A. Yamashita in Figure 7 (c) do not appear in Figure 7 (a). Such as Stephen H. Bryant, Gabriele H. Marchler, and David I. Hurwitz (we can also see the importance of these three researchers to Roxanne A. Yamashita from https://www.aminer.cn/). By Figure 7 (b), we can see that the community obtained by *MSCS* is a connected subgraph composed of multiple stars. Furthermore, Figure 7 (b) contains many *query-drifted vertices*, which come from various backgrounds. Similar trends can also be observed in the community of Prof. Joel E. Richardson (due to the space limit, we only visualize the result of *QTCS* in Figure 7 (d)). Since *PCore* and *MSCS* only consider the temporal cohesiveness but ignore the temporal proximity with the query vertex, they may find many temporal irrelevant vertices to the query vertex for satisfying their cohesiveness. In summary, these case studies further indicate that our model *QTCS* is indeed more effective than the other models in searching *query-centered* temporal communities.

7 RELATED WORK

Community detection. Existing studies mainly rely on structure-based approaches to identify all communities from graphs, including modularity optimization [33], spectral analysis [7], hierarchical clustering [38] and cohesive subgraph discovering [2]. However, all these methods do not consider the temporal dimension of networks. Until recently, some studies have been done on community detection over temporal networks [4, 23, 26, 27, 32, 35, 39, 56, 57, 59]. For instance, Lin et al. [27] proposed the stable quasi-clique to capture the stability of cohesive subgraphs. Ma et al. [32] studied the heavy subgraphs for detecting traffic hotspots.

Community search. As a meaningful counterpart, community search has recently become a focal point of research in network analysis [10, 17]. For simple graphs, they aim to identify the subgraphs that contain the given query vertices and satisfy a specific community model such as k-core [6, 44], k-truss [15, 28], clique [5, 58], density [52], connectivity [41, 42, 45] and conductance [1, 25, 55]. For instance, Sozio et al. [44] introduced a framework of community search, which requires the target community to be a connected subgraph containing query vertices and has a good score w.r.t. the proposed quality function. In particular, they used the k-core as the quality function. Since the k-core is not necessarily dense, Huang et al. [15] adopted a more cohesive subgraph model k-truss to model the community. Recently, Wu et al. [52] observed the above approaches have the free rider issue, that is, the returned community often contains many redundant vertices. However, our proposed *query drift* issue (Definition 3.1) is more strict than the free rider issue. That is, if an objective function $f(.)$ suffers from the *query drift* issue, then $f(.)$ must have the free rider issue, and vice versa is not necessarily true (Section 3.2). Besides, graph diffusion-based local clustering methods have also been considered. For example, Tong et al. [45] applied random walk with restart to measure the goodness score of any vertex w.r.t. the query vertices. Andersen et al. [1] used Personalized PageRank to sort vertices and then executed a sweep cut procedure to obtain the local optimal conductance. However, the random walk used in these works is mainly tailored to static networks. Besides simple graphs, more complicated attribute information associated with vertices or edges also has been investigated, such as keyword-based graphs [9, 16, 29], location-based social networks [3, 8], multi-valued graphs [22] and heterogeneous information networks [11, 19]. However, they ignore the temporal properties of networks that frequently appear in applications. Recently, two studies have been done on temporal community search [12, 46]. However, they suffer from several defects stated in Sections 1, 3.2 and 6).

8 CONCLUSION

In this work, we are the first to introduce and address the *query-centered* temporal community search problem. We first develop the Time-Constrained Personalized PageRank to capture the temporal proximity between the query vertex and other vertices. Then, we introduce β-temporal proximity core to combine seamlessly structural cohesiveness and temporal proximity. Subsequently, we formulate our problem as an optimization task, which returns a β-temporal proximity core with the largest β. To query quickly, we first devise an exact and near-linear time greedy removing algorithm *EGR*. To further boost efficiency, we then propose an approximate two-stage local search algorithm *ALS*. Finally, extensive experiments on eight real-life temporal networks and nine competitors show the superiority of the proposed solutions.

ACKNOWLEDGMENTS

The work was supported by (i) the National Natural Science Foundation of China under Grant Nos. 62072205 and 61932004, (ii) National Key Research and Development Program of China 2021YFB3301301, (iii) NSFC Grants U2241211, 62072034, (iv) Fundamental Research Funds for the Central Universities under Grant SWU-KQ22028. Pingpeng Yuan and Rong-Hua Li are the corresponding authors of this paper.

REFERENCES

[1] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *FOCS*. 475–486.

[2] Lijun Chang and Lu Qin. 2019. Cohesive Subgraph Computation Over Large Sparse Graphs. In *ICDE*. 2068–2071.

[3] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, Jeffrey Xu Yu, and Jianxin Li. 2020. Finding Effective Geo-social Group for Impromptu Activities with Diverse Demands. In *KDD*. 698–708.

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

[5] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *SIGMOD*. 277–288.

[6] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *SIGMOD*. 991–1002.

[7] Luca Donetti and Miguel A Munoz. 2004. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment* 2004, 10 (2004), 10012.

[8] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective Community Search over Large Spatial Graphs. *Proc. VLDB Endow.* 10, 6 (2017), 709–720.

[9] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective Community Search for Large Attributed Graphs. *Proc. VLDB Endow.* 9, 12 (2016), 1233–1244.

[10] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *VLDB J.* 29, 1 (2020), 353–392.

[11] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and Efficient Community Search over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 13, 6 (2020), 854–867.

[12] Edoardo Galimberti, Martino Ciaperoni, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2021. Span-core Decomposition for Temporal Networks: Algorithms and Applications. *ACM Trans. Knowl. Discov. Data* 15, 1 (2021), 2:1–2:44.

[13] Petter Holme. 2015. Modern temporal network theory: A colloquium. *CoRR* abs/1508.01303 (2015).

[14] Weishu Hu, Haitao Zou, and Zhiguo Gong. 2015. Temporal PageRank on Social Networks. In *WISE*. 262–276.

[15] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*. 1311–1322.

[16] Xin Huang and Laks V. S. Lakshmanan. 2017. Attribute-Driven Community Search. *Proc. VLDB Endow.* 10, 9 (2017), 949–960.

[17] Xin Huang, Laks V. S. Lakshmanan, and Jianliang Xu. 2017. Community Search over Big Graphs: Models, Algorithms, and Opportunities. In *ICDE*. 1451–1454.

[18] Xin Huang, Laks V. S. Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate Closest Community Search in Networks. *PVLDB* 9, 4 (2015), 276–287.

[19] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and Efficient Relational Community Detection and Search in Large Dynamic Heterogeneous Information Networks. *Proc. VLDB Endow.* 13, 10 (2020), 1723–1736.

[20] Jian-Huang Lai, Chang-Dong Wang, and Philip S. Yu. 2013. Dynamic Community Detection in Weighted Graph Streams. In *SDM*. 151–161.

[21] Michael Levi and Peter Reuter. 2006. Money laundering. *Crime and justice* 34, 1 (2006), 289–375.

[22] Rong-Hua Li, Lu Qin, Fanghua Ye, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2018. Skyline Community Search in Multi-valued Networks. In *SIGMOD*. 457–472.

[23] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent Community Search in Temporal Networks. In *ICDE*. 797–808.

[24] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. 2022. Efficient Personalized PageRank Computation: A Spanning Forests Sampling Based Approach. In *SIGMOD*. 2048–2061.

[25] Longlong Lin, Ronghua Li, and Tao Jia. 2023. Scalable and Effective Conductance-Based Graph Clustering. In *AAAI*. 4471–4478.

[26] Longlong Lin, Pingpeng Yuan, Rong-Hua Li, and Hai Jin. 2022. Mining Diversified Top-r Lasting Cohesive Subgraphs on Temporal Networks. *IEEE Trans. Big Data* 8, 6 (2022), 1537–1549.

[27] Longlong Lin, Pingpeng Yuan, Rong-Hua Li, Jifei Wang, Ling Liu, and Hai Jin. 2022. Mining Stable Quasi-Cliques on Temporal Networks. *IEEE Trans. Syst. Man Cybern. Syst.* 52, 6 (2022), 3731–3745.

[28] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based Community Search over Large Directed Graphs. In *SIGMOD*. 2183–2197.

[29] Qing Liu, Yifan Zhu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. VAC: Vertex-Centric Attributed Community Search. In *ICDE*. 937–948.

[30] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *WSDM*. 163–172.

[31] Laishui Lv, Kun Zhang, Ting Zhang, Dalal Bardou, Jiahui Zhang, and Ying Cai. 2019. PageRank centrality for temporal networks. *Physics Letters A* 383, 12 (2019), 1215–1222.

[32] Shuai Ma, Renjun Hu, Luoshu Wang, Xuelian Lin, and Jinpeng Huai. 2017. Fast Computation of Dense Temporal Subgraphs. In *ICDE*. 361–372.

[33] Mark EJ Newman. 2004. Fast algorithm for detecting community structure in networks. *Physical review E* 69, 6 (2004), 066133.

[34] L. Page, S. Brin, R. Motwani, and T. Winograd. 1999. The PageRank Citation Ranking : Bringing Order to the Web. In *WWW*.

[35] Hongchao Qin, Rong-Hua Li, Guoren Wang, Lu Qin, Yurong Cheng, and Ye Yuan. 2019. Mining Periodic Cliques in Temporal Networks. In *ICDE*. 1130–1141.

[36] Hongchao Qin, Ronghua Li, Ye Yuan, Guoren Wang, Lu Qin, and Zhiwei Zhang. 2022. Mining Bursting Core in Large Temporal Graph. *Proc. VLDB Endow.* 15, 13 (2022), 3911–3923.

[37] Luis E C Rocha and Naoki Masuda. 2014. Random walk centrality for temporal networks. *New Journal of Physics* 16, 6 (2014), 063023.

[38] Lior Rokach and Oded Maimon. 2005. Clustering methods. In *Data mining and knowledge discovery handbook*. 321–352.

[39] Polina Rozenshtein, Francesco Bonchi, Aristides Gionis, Mauro Sozio, and Nikolaj Tatti. 2018. Finding Events in Temporal Networks: Segmentation Meets Densest-Subgraph Discovery. In *ICDM*. 397–406.

[40] Polina Rozenshtein and Aristides Gionis. 2016. Temporal PageRank. In *ECML-PKDD*. 674–689.

[41] Natali Ruchansky, Francesco Bonchi, David García-Soriano, Francesco Gullo, and Nicolas Kourtellis. 2015. The Minimum Wiener Connector Problem. In *SIGMOD*. 1587–1602.

[42] Natali Ruchansky, Francesco Bonchi, David Garcia-Soriano, Francesco Gullo, and Nicolas Kourtellis. 2017. To Be Connected, or Not to Be Connected: That is the Minimum Inefficiency Subgraph Problem. In *CIKM*. 879–888.

[43] Arlei Silva, Ambuj K. Singh, and Ananthram Swami. 2018. Spectral Algorithms for Temporal Graph Cuts. In *WWW*. 519–528.

[44] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *KDD*. 939–948.

[45] Hanghang Tong and Christos Faloutsos. 2006. Center-piece subgraphs: problem definition and fast solutions. In *KDD*. 404–413.

[46] Ioanna Tsalouchidou, Francesco Bonchi, and Ricardo Baeza-Yates. 2020. Adaptive Community Search in Dynamic Networks. In *BigData*. 987–995.

[47] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *KDD*. 505–514.

[48] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In *SIGMOD*. 441–456.

[49] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. 2014. Path Problems in Temporal Graphs. *PVLDB* 7, 9 (2014), 721–732.

[50] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In *SIGMOD*. 1996–2008.

[51] Huanhuan Wu, Yunjian Zhao, James Cheng, and Da Yan. 2017. Efficient Processing of Growing Temporal Graphs. In *DASFAA*. 387–403.

[52] Yubao Wu, Ruoming Jin, Jing Li, and Xiang Zhang. 2015. Robust Local Community Detection: On Free Rider Effect and Its Elimination. *Proc. VLDB Endow.* 8, 7 (2015), 798–809.

[53] Yubao Wu, Ruoming Jin, and Xiang Zhang. 2016. Efficient and Exact Local Search for Random Walk Based Top-K Proximity Query in Large Graphs. *IEEE Trans. Knowl. Data Eng.* 28, 5 (2016), 1160–1174.

[54] Wenlei Xie, Yuanyuan Tian, Yannis Sismanis, Andrey Balmin, and Peter J. Haas. 2015. Dynamic interaction graphs with probabilistic edge decay. In *ICDE*. 1143–1154.

[55] Renchi Yang, Xiaokui Xiao, Zhewei Wei, Sourav S. Bhowmick, Jun Zhao, and Rong-Hua Li. 2019. Efficient Estimation of Heat Kernel PageRank for Local Clustering. In *SIGMOD*. 1339–1356.

[56] Yi Yang, Da Yan, Huanhuan Wu, James Cheng, Shuigeng Zhou, and John C. S. Lui. 2016. Diversified Temporal Subgraph Pattern Mining. In *KDD*. 1965–1974.

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

[58] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2018. Index-Based Densest Clique Percolation Community Search in Networks. *IEEE Trans. Knowl. Data Eng.* 30, 5 (2018), 922–935.

[59] Chun-Xue Zhu, Longlong Lin, Pingpeng Yuan, and Hai Jin. 2022. Discovering Cohesive Temporal Subgraphs with Temporal Density Aware Exploration. *J. Comput. Sci. Technol.* 37, 5 (2022), 1068–1085.