Towards Query-Centered Temporal Community Search via Time-Constrained Personalized PageRank

ABSTRACT

The existing studies on temporal community search, retrieving the community containing the user-specified query vertex from temporal networks, mainly focus on the structural and temporal cohesiveness. However, existing solutions suffer from two major defects: (i) they ignore the temporal proximity between the query vertex and other vertices but simply require the result to include the query vertex. Thus, they may find many temporal irrelevant vertices to the query vertex for satisfying their cohesiveness, resulting in the query vertex being marginalized. We refer to such temporal irrelevant vertices as *query-drifted vertices*; (ii) their methods are NP-hard, incurring prohibitively high costs for exact solutions or severely compromised results for approximate/heuristic algorithms. Inspired by these, we propose a novel problem named *query-centered* temporal community search to circumvent *query-drifted vertices*. Specifically, we first present a novel concept of Time-Constrained Personalized PageRank to characterize the temporal proximity between the query vertex and other vertices. Then, we introduce a new community search model called β-temporal proximity core, which can seamlessly combine the temporal proximity and structural cohesiveness. Subsequently, we formulate our problem as an optimization task that aims at finding a β-temporal proximity core with the largest β. To solve our problem, we first devise an exact and near-linear time greedy removing algorithm that iteratively removes unpromising vertices. To further 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.

ACM Reference Format:

. 2018. Towards Query-Centered Temporal Community Search via Time-Constrained Personalized PageRank. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Many 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, the research on community mining can be divided into community detection [4, 9, 32, 36] and community search [1, 5, 8, 42, 51, 54]. The former aims to find all communities by some predefined criteria (e.g., modularity [32]), 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-specified query vertex, which is more efficient and personalized. Additionally, it also witnesses a series of applications such as social recommendation [42], protein complexes identification [8] and impromptu activities organization [5].

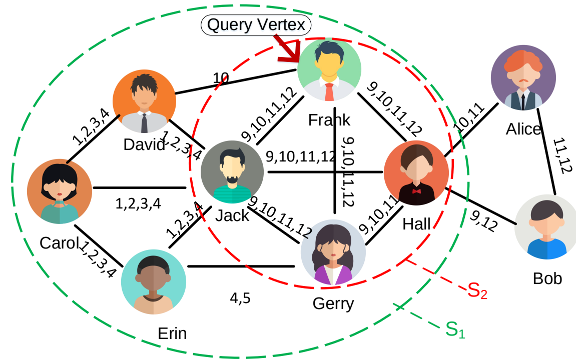


Figure 1: Motivation Example

Despite the significant success of community search, most existing approaches are tailored to static networks. However, many real networks often contain complex time interaction information among vertices, which are typically named temporal networks [15]. For example, in e-commerce or social media, the connection between two parties was made at a specific time. Thus, conventional static community search methods may find a sub-optimal result. For example, Figure 1 shows a sample temporal collaboration network, in which the timestamps of each edge indicate when the two authors co-authored papers. We assume *Frank* is the query vertex. By using 3-core as the community model (i.e., a 3-core is a community in which each vertex has at least 3 neighbors), the vertices S_1 within the green circle is the answer if the time information of edges is ignored [8, 42]. Although *David*, *Carol* and *Erin* meet the structural constraints in S_1 (i.e., each of them has at least 3 neighbors in S_1), they are far away from *Frank* from the perspective of temporal information. In other words, *David*, *Carol* and *Erin* have poor temporal proximity with respect to *Frank* (Section 2.2). Intuitively, the vertices S_2 included in the red circle may be the target community when considering the temporal properties of edges. This is because *Hall*, *Jack* and *Gerry* worked closely with *Frank* at time 9-12. One possible explanation is that *David*, *Carol* and *Erin* are collaborators of *Jack* when *Jack* was a Ph.D. student, while *Hall*, *Frank* and *Gerry* are students guided by *Jack* when *Jack* became a professor. Thus, S_2 is the research group of *Frank*. So, it is more interesting and meaningful to integrate structure and complex temporal information for temporal community search.

Surprisingly, only a few recent researches have been done on temporal community search [14, 44]. For example, Tsalouchidou et al. [44] identified the minimum temporal-inefficiency subgraph containing the given query vertex. On the other hand, there are some studies on temporal community detection that can also solve temporal community search with simple adjustments. For instance, they first find all possible communities by the predefined criteria [6, 24, 34], and then select the target community containing the query vertex from these communities. Unfortunately, these existing methods suffer from two major defects. First, the vertices in the target community should be closely related to the query vertex in community search problem [20, 48]. However, these existing methods do not consider the temporal proximity between the query

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

vertex and other vertices but simply require the result to include the query vertex. Thus, they may find many temporal irrelevant vertices to the query vertex for satisfying their objective functions (e.g., structural and temporal cohesiveness), resulting in the query vertex being marginalized (see case studies in Section 6 for details). We refer to such temporal irrelevant vertices as *query-drifted vertices* (Section 3.2). Second, these methods are NP-hard, incurring either prohibitively high costs for exact solutions or severely compromised results for approximate/heuristic algorithms. For example, [24, 44] cannot obtain the results within two days in our experiments, which is clearly impractical for online interactive graph explorations.

Solutions. To address the above defects, we formulate a novel problem and devise fast search algorithms to solve it. For the first defect, we extend the well-known proximity metric Personalized PageRank to Time-Constrained Personalized PageRank (*TPPR*) by 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. As a result, by maximizing the value of β of a β-temporal proximity core, we can ensure that these *query-drifted vertices* are removed and the query vertex is *centered* in the detected community (Section 3.2 and 6). Besides, β-temporal proximity core has only one parameter (i.e., the teleportation probability α in Section 2.3) like [14, 44], which is user-friendly. However, [6, 24, 34] have many parameters which are heavily dependent on datasets and are often hard-to-tune. For the second defect, 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 2.4). To compute *TPPR*, a straightforward solution is to apply the traditional power iteration method [33], but it requires prohibitively high time costs. Based on in-depth observations, we propose a non-trivial dynamic programming approach to compute *TPPR* for every vertex. To further boost efficiency, we then 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 listed as follows:

- Novel Model. We formulate the *query-centered* temporal community search problem in Section 2. To the best of our knowledge, the problem has never been studied in the literature.
- Theoretical Analysis. We introduce the concept of *query-drifted vertices* to analyze the limitations of the existing solutions in Section 3. In particular, we show that most existing methods contain many *query-drifted vertices*, resulting in the query vertex being marginalized. However, our proposed model can circumvent these *query-drifted vertices*, resulting in that the query vertex is *centered* in the target community.
- Efficient Algorithms. To solve our problem quickly, we propose two practical algorithms in Section 4 and 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. Experimental evaluations (Section 6) on eight datasets with different domains and sizes demonstrate our proposed solutions indeed are more efficient, scalable, and effective than the existing 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. Our model is much denser and more separable in terms of temporal feature than the competitors. Our model can find high-quality *query-centered* temporal communities by eliminating *query-drifted vertices* which the competitors cannot identify.

2 PROBLEM FORMULATION

In this section, we first give some important notations. Subsequently, we introduce a novel concept of Time-Constrained Personalized PageRank to capture the temporal proximity between the query vertex and other vertices. Finally, we state our problem.

2.1 Notations

We use $\mathcal{G}(V, \mathcal{E})$ to denote any 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 which 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 edge stream [15], 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 graph G. Let $G_S = (S, E_S)$ be the subgraph induced by S if $S \subseteq V$ and $E_S = \{(u, v) \in E | u, v \in S\}$. We use $N_S(v) = \{u \in S | (u, v) \in E\}$ to denote the neighbors of v in S.

2.2 Time-Constrained Personalized PageRank

Recall that Personalized PageRank (*PPR*) is a widely adopt proximity metric in network analysis, which can measure the structural proximity between two vertices [29, 33, 45]. Essentially, *PPR* models a random walk process that has a unique stationary distribution and generally solving the following equation[1]:

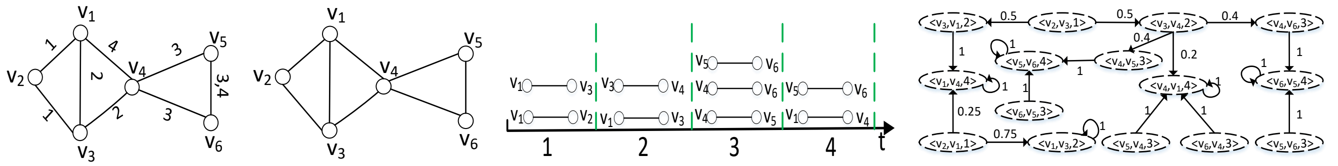
$$\mathbf{x} = \alpha \mathbf{s} + (1 - \alpha)\mathbf{x}\mathbf{W} \tag{1}$$

$\mathbf{x}$ is the stationary *PPR* distribution, α is the teleportation probability, and $\mathbf{s}$[2] 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, the research on how to design effective temporal proximity is not sufficient (Section 7). Thus, to preserve the rich temporal information in *PPR*, we face the following two challenges. First, how to design an effective walk in temporal networks. In real-world scenarios, the information transmission follows the time-order and asynchronous interaction behaviors. 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 with respect to (w.r.t.) time-order. Second, how to design an effective state transition matrix in temporal networks. Intuitively,

[1] We use lowercase letters to denote scalars (e.g., α), bold lowercase letters to denote row vectors (e.g., $\mathbf{s}$ or $\mathbf{x}$), bold capital letters to denote matrices (e.g., $\mathbf{W}$ or $\mathbf{P}$).

[2] $\mathbf{s}$ is a distribute in the original *PPR*. That is, multiple non-zero entries are allowed in $\mathbf{s}$. When $\mathbf{s}$ is a one-hot vector, *PPR* is also called random walk with restart [43].



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

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

the preference of an interaction decreases as time goes by [50] (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 $(v_1, v_4, 4)$. But the traditional state transition matrix $\mathbf{W}$ cannot distinguish such edge relationships. Besides, more than an 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 $< v, u, t >$[3]. 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 $\vec{e}$, $N^{>}(\vec{e}) = \{< u, v, t > | u = tail(\vec{e}), t > time(\vec{e})\}$, $\vec{e}_u^{out} = \{\vec{e} | head(\vec{e}) = u\}$, $\vec{e}_u^{in} = \{\vec{e} | tail(\vec{e}) = u\}$. Based on these symbols, we present the following definition to overcome the challenges discussed above.

Definition 2.1. **[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

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

$P(\vec{e}_i \rightarrow \vec{e}_j)$ 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 [22, 47]. 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 [29, 33, 45]. 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. Note that $\mathbf{P}$ is constructed only once for each dataset to support different queries. Figure 2(d) shows the state transition for Figure 2(a), in which we ignore the isolated *ordered* temporal edges.

Definition 2.2. **[Time-Constrained Personalized PageRank]** 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 w.r.t. q is denoted by $tppr_q(u) = \sum_{\vec{e} \in \vec{e}_u^{in}} \widetilde{ppr}(\alpha, \widetilde{\chi_q})(\vec{e})$.

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

$\widetilde{\chi_q} \in R^{1 \times m}$ 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 the Definition 2.2 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, $\widetilde{ppr}(\alpha, \widetilde{\chi_q})(\vec{e})$ reflects the temporal 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 [29, 45]. 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. So, it is reasonable to use $tppr_q(u)$ to describe 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 above, the Time-Constrained Personalized PageRank (*TPPR*) can be used to measure the temporal proximity between the query vertex and other vertices. Therefore, a naive way is to identify 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 [8, 42]. We call this model *QTCS_Baseline*, which serves as a baseline model for experimental comparison in Section 6.

Definition 2.3. **[*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; (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 (see Section 6 for details). 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 2.4. **[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 2.4, we know that the query-biased temporal degree measures the quality 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 that are closely related to the query vertex.

Definition 2.5. **[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 that are closely related to the query vertex. As a result, it ensures that the detected community is very related to

[3] () and <> represent the temporal edge and *ordered* temporal edge, respectively.

the query vertex, which is easier to interpret why the community is formed (see case studies of Section 6 for details).

Problem 1 (*QTCS*). Given a temporal graph $\mathcal{G}(V,\mathcal{E})$, a teleportation probability α and a query vertex q, ***query-centered* temporal community search** 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*) [8, 42] does have many similarities with our methods, but there are also pivotal differences. First, 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 [20, 48]. Unlike *CSM*, our solution is based on a new definition of query-biased temporal degree (Definition 2.4) which can capture the relative importance for temporal community search. Second, 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, needing 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., [46]), resulting in prohibitively computational costs. However, our technologies are directly oriented to temporal networks which are very efficient as shown in our experiments. Additionally, 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. 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 if and only if 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 [20, 48]. 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, our proposed QTCS does not suffer from the query drift issue.*

Proof. Let S^* be the solution for the *QTCS* problem, and thus $q \in S^*$. The Proposition can be proved by contradiction. Assume that there is a vertex set S such that $f(S \cup S^*) \geq f(S^*)$ and $\min\{\rho_{S \cup S^*}(u) | u \in S \cup S^*\} \leq \min\{\rho_{S^*}(u) | u \in S^*\}$. By Definition 2.5 and Problem 1, we have $f(C) = \min\{\rho_C(u) | u \in C\}$ for *QTCS*. Thus, $f(S \cup S^*) \geq f(S^*)$ is equivalent to $\min\{\rho_{S \cup S^*}(u) | u \in S \cup S^*\} \geq \min\{\rho_{S^*}(u) | u \in S^*\}$. So, $\min\{\rho_{S \cup S^*}(u) | u \in S \cup S^*\} = \min\{\rho_{S^*}(u) | u \in S^*\}$. As a result, (i) $q \in S \cup S^*$; (ii) $S \cup S^*$ is a β-temporal core with the largest β. This contradicts the maximality of S^* (i.e., condition (iii) of Problem 1). Thus, there does not exit *query-drifted vertices* S for *QTCS*. □

Proposition 3.3. *Given a temporal graph $\mathcal{G}$ and a query vertex q, the methods in [6, 14, 24, 34] suffer from the query drift issue.*

Proof. Let C_q be a vertex set that satisfies conditions (i) and (ii) of Problem 1. Thus, by Definition 2.5 and Problem 1, we know that condition (ii) of Definition 3.1 holds for C_q and any $C^*(f)$. Next, we prove that [6, 14, 24, 34] meet the condition (i) of Definition 3.1.

For [6]: The objective function $f(C) = \frac{m(\mathcal{G}_C)}{|C| \cdot |\mathcal{T}_C|}$, in which $m(\mathcal{G}_C)$ is the sum of edge weights within the temporal subgraph $\mathcal{G}_C$ and $\mathcal{T}_C$ is the time set of $\mathcal{G}_C$. For example, in Figure 2(a), we let $C = \{v_1, v_3, v_4, v_5, v_6\}$, thus $m(\mathcal{G}_C) = 7$ and $\mathcal{T}_C = \{2, 3, 4\}$. So, $C^*(f)$ is a vertex set with the largest f value. Since $m(\mathcal{G}_C)/\mathcal{T}_C$ is a monotonically increasing supermodular and $|C| > 0$ is a submodular, $f(C^*(f) \cup C_q) \geq f(C_q)$ according to [48]. Thus, [6] suffer from the *query drift* issue.

For [14]: Given a fixed interval I and a static "AND" graph $G_I(C) = \cap_{t \in I}\{(u,v) | u, v \in C, (u,v,t) \in \mathcal{G}\}$, the objection function $f(C) = \min_{u \in C} d_I(u, C)$, in which $d_I(u, C)$ is the degree of u in $G_I(C)$. So, $C^*(f)$ is a vertex set with the largest f value. Thus, $\min_{u \in C^*(f) \cup C_q} d_I(u, C^*(f) \cup C_q) \geq \min_{u \in C_q} d_I(u, C_q)$. That is, $f(C^*(f) \cup C_q) \geq f(C_q)$. Thus, [14] suffer from the *query drift* issue.

For [24]: If C is a (θ,τ)-persistent k-core, then $f(C) = |C|$, otherwise $f(C) = 0$. Thus, $C^*(f)$ is a (θ,τ)-persistent k-core with the largest f value. When C_q is a (θ,τ)-persistent k-core, then we have $C^*(f) \cup C_q$ is also a (θ,τ)-persistent k-core and $f(C^*(f) \cup C_q) = |C^*(f) \cup C_q| \geq f(C_q) = |C_q|$. When C_q is not a (θ,τ)-persistent k-core, we have $f(C_q) = 0$ and $f(C^*(f) \cup C_q) \geq f(C_q)$. So, [24] suffer from the *query drift* issue.

For [34]: If C is a periodic clique, then $f(C) = 1$, otherwise $f(C) = 0$. Thus $C^*(f)$ is any periodic clique. When C_q is a periodic clique, we let $C^*(f)$ contains C_q. Thus we have $C^*(f) \cup C_q$ is also a periodic clique and $f(C^*(f) \cup C_q) = 1 \geq f(C_q) = 1$. When C_q is not a periodic clique, we have $f(C_q) = 0$ and $f(C^*(f) \cup C_q) \geq f(C_q)$. So, [34] suffer from the *query drift* issue. □

Remark. The objection function $f(C) = \sum_{u,v \in C}(k(\frac{1}{dist_C(u,v)} + \frac{1}{dist_C(v,u)}) - 1)$ in [44], in which $k \in [0, 1/2]$ and $dist_C(.) \geq 1$ is an asymmetric distance function within $\mathcal{G}_C$ that linearly integrates the temporal and spatial dimensions. When $C_q \subseteq C^*(f)$, we have $f(C^*(f)) = f(C^*(f) \cup C_q) \geq f(C_q)$ because $C^*(f)$ is the vertex set with the largest f value. As a result, [44] has the *query drift* issue when $C_q \subseteq C^*(f)$. Unfortunately, the formal proof for $C_q \nsubseteq C^*(f)$ is quite difficult and we leave it as an open problem. In this regard, we note as follows. First, [44] involves complex distance calculations, so it has high time complexity and even it is NP-hard (more details in [44]). In particular, [44] cannot obtain the results within two days on some datasets (Exp-1 of Section 6). Second, [44] has poor community quality (Exp-6 of Section 6). This is because [44] only applied distance to measure the quality of the community, resulting in that it is a local measure and ignores the cohesiveness of the community.

3.3 Handle Multiple Query Vertices

In many applications, multiple query vertices may be initiated by users. We show that our proposed frameworks for single query vertex can be generalized to deal with multiple query vertices. Let S be the query vertex set, the *TPPR* of vertex u w.r.t. S is denoted by $tppr_S(u) = \sum_{q \in S} tppr_q(u)/|S|$. [4] By doing so, we propose a new definition and a new problem as follows.

Definition 3.4. Given a vertex set C and a query vertex set S, the query-biased temporal degree of vertex u w.r.t. C and S is defined as: $\rho_C^S(u) = \sum_{v \in N_C(u)} tppr_S(v)$.

Problem 2 (*QTCS* with multiple query vertices). Given a temporal graph $\mathcal{G}(V, \mathcal{E})$, a teleportation probability α and a query vertex set S, the problem is to identify a vertex set C, satisfying (i) $S \subseteq C$ and G_C is connected; (ii) $\min\{\rho_C^S(u) | u \in C\}$ is the maximum; (iii) there does not exist another community $C' \supseteq C$ meets the above conditions.

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

Here, we focus on calculating *TPPR* of every vertex. Straightforwardly, we can use the classic power iteration method [33] to solve the Equation 3 by utilizing the knowledge of linear algebra (i.e., matrix-vector product operations). However, such a method has a high time overhead when handling temporal networks. The reasons are as follows. The time complexity of the power iteration method is O(MN), 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). Thus, the time complexity of the power iteration method is O(m^2N). Motivated by this, we propose an efficient algorithm with near-linear time by simulating the process of the temporal walk and applying edge stream to reduce computational cost.

Definition 4.1. **[l-hop temporal walk]** A l-hop temporal walk from vertex i to vertex j is a sequence of 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 $tppr(u) = \sum_{i=0}^{\infty} \alpha(1-\alpha)^i \sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u}} P(tw_{i+1})$.*

Proof. First, the equation $\mathbf{x} = \alpha\mathbf{s} + (1-\alpha)\mathbf{xP}$ is equivalent to $\mathbf{x}(\mathbf{I} - (1-\alpha)\mathbf{P}) = \alpha\mathbf{s}$. Furthermore, the matrix $(\mathbf{I} - (1-\alpha)\mathbf{P})$ is nonsingular because it is strictly diagonally dominant, so this equation has a unique solution $\mathbf{x}$ according to Cramer's Rule.

Second, let $\mathbf{y} = \alpha \sum_{i=0}^{\infty}(1-\alpha)^i\mathbf{P}^i$, we have $\alpha\mathbf{s} + (1-\alpha)\mathbf{syP} = \alpha\mathbf{s} + (1-\alpha)\mathbf{s}\alpha\sum_{i=0}^{\infty}(1-\alpha)^i\mathbf{P}^i\mathbf{P} = \alpha\mathbf{s} + \mathbf{s}\alpha\sum_{i=1}^{\infty}(1-\alpha)^i\mathbf{P}^i = \mathbf{s}\alpha\sum_{i=0}^{\infty}(1-\alpha)^i\mathbf{P}^i = \mathbf{sy}$. That is $\alpha\mathbf{s} + (1-\alpha)\mathbf{syP} = \mathbf{sy}$. Since $\mathbf{x} = \alpha\mathbf{s} + (1-\alpha)\mathbf{xP}$ and $\mathbf{x}$ has a unique solution, $\mathbf{x} = \mathbf{sy} = \alpha\sum_{i=0}^{\infty}(1-\alpha)^i\mathbf{sP}^i$.

Third, for $\widetilde{ppr}(\alpha, \widetilde{\chi_q}) = \alpha\widetilde{\chi_q} + (1-\alpha)\widetilde{ppr}(\alpha, \widetilde{\chi_q})\mathbf{P}$, we have $\widetilde{ppr}(\alpha, \widetilde{\chi_q}) = \alpha\sum_{i=0}^{\infty}(1-\alpha)^i\widetilde{\chi_q}\mathbf{P}^i$ by the previous proof. Therefore, $\widetilde{ppr}(\alpha, \widetilde{\chi_q})(\vec{e}) = \alpha\sum_{i=0}^{\infty}(1-\alpha)^i P(q \overset{(i+1)hop}{\rightsquigarrow} \vec{e})$, in which $P(q \overset{(i+1)hop}{\rightsquigarrow} \vec{e})$ represents the probability that first from q to $head(\vec{e})$ by i-hop temporal walk and then walking to $tail(\vec{e})$. So, $tppr(u) = \sum_{\vec{e} \in \vec{e}_u^{\,in}} \widetilde{ppr}(\alpha, \widetilde{\chi_q})(\vec{e}) = \alpha\sum_{i=0}^{\infty}(1-\alpha)^i \sum_{\vec{e} \in \vec{e}_u^{\,in}} P(q \overset{(i+1)hop}{\rightsquigarrow} \vec{e}) = \alpha\sum_{i=0}^{\infty}(1-\alpha)^i \sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u}} P(tw_{i+1})$. □

A failed attempt. According to Lemma 4.3, a naive solution is first to enumerate all temporal walks from query vertex 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 present an important observation as follows.

An important observation. According to Definition 2.1 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.

Based on this observation, we further present an important lemma (Lemma 4.4). Before proceeding further, we denote a α-discount temporal walk as the following random walk process: (1) it starts from q; (2) at each step it stops in the current state with probability α, or it continues to walk according to Equation 2 with probability 1-α. Furthermore, we use u^t to denote 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.

Lemma 4.4. *Given a temporal graph $\mathcal{G}(V, \mathcal{E})$, a query vertex q, and a teleportation probability α, we have $tppr(u) = \sum_{t \in T_1} D[u][t] +$*

[4] Other alternatives are possible for defining $tppr_S(u)$. For example, $tppr_S(u) = \min\{tppr_q(u) | q \in S\}$ or $tppr_S(u) = \prod_{q \in S} tppr_q(u)$.

$\sum_{t \in T_2} D[u][t]/\alpha$, *in which* $T_1 = \{t | u^t$ *is not a dangling state*$\}$ *and* $T_2 = \{t | u^t$ *is a dangling state*$\}$.

Proof. Assume that there is a temporal walk $\{\vec{e}_1, \vec{e}_2, ...\vec{e}_l\}$ such that $head(\vec{e}_1) = q$ and $\vec{e}_l = u^t$.

Case 1: If $\vec{e}_i$ is not a dangling state for $1 \leq i \leq l$ and $P(tw_{i+1}) \neq 0$, we have $l \neq \infty$ by the previous observation. Let l_{max} be the maximum l that satisfies the above condition, we have $\sum_{i=0}^{l_{max}} \alpha(1-\alpha)^i \sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u^t}} P(tw_{i+1}) = D[u][t]$.

Case 2: If there are some dangling states, there must exist an integer k such that $\vec{e}_i$ is not a dangling state for $i < k$ and $\vec{e}_j$ is a dangling state for $k \leq j \leq l$. Let l_{max} be the maximum l that satisfies the above condition, note that l_{max} may be ∞. Thus, we have $\sum_{i=0}^{l_{max}} \alpha(1-\alpha)^i \sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u^t}} P(tw_{i+1}) = \sum_{i=0}^{l_{max}-k} (1-\alpha)^i D[u][t]$. So, $\sum_{i=0}^{\infty} \alpha(1-\alpha)^i \sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u^t}} P(tw_{i+1}) = \sum_{i=0}^{\infty} (1-\alpha)^i D[u][t] = D[u][t] * (1 + (1-\alpha) + (1-\alpha)^2 + ...(1-\alpha)^{\infty}) = D[u][t] * (1/(1-(1-\alpha))) = D[u][t]/\alpha$.

In short, if u^t is not a dangling state, $\sum_{i=0}^{\infty} \alpha(1-\alpha)^i \sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u^t}} P(tw_{i+1}) = D[u][t]$. If u^t is a dangling state, we have $\sum_{i=0}^{\infty} \alpha(1-\alpha)^i \sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u^t}} P(tw_{i+1}) = D[u][t]/\alpha$. Thus, we have $tppr(u) = \sum_{i=0}^{\infty} \alpha(1-\alpha)^i \sum_{tw_{i+1} \in TW_{i+1}^{q \rightsquigarrow u}} P(tw_{i+1}) = \sum_{t \in T_1} D[u][t] + \sum_{t \in T_2} D[u][t]/\alpha$ according to Lemma 4.3. □

Based on Lemma 4.4, we devise an efficient and non-trivial dynamic programming approach (Algorithm 1) to compute *TPPR* for every vertex 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 edge stream to ensure the time of temporal edges is non-decreasing, which can facilitate the $D[u][t]$ calculation (see the definition of $D[u][t]$ for details). Thus, for each temporal edge (u, v, t), we update the dictionary structures $D[u][t]$ and $D[v][t]$ accordingly (Lines 3-10). As a result, the *TPPR* of u is the sum of $D[u][t]$ for different t according to Lemma 4.4 (Lines 11-15).

Theorem 4.5. *Algorithm 1 can correctly compute TPPR for each vertex. The time complexity of Algorithm 1 is* $O(\mathcal{T}_{max} \cdot (m+n))$, *where* $\mathcal{T}_{max} = \max\{\mathcal{T}_u | u \in V\}$, $\mathcal{T}_u = |\{t | (u, v, t) \in \mathcal{E}\}|$.

Proof. The algorithm takes m rounds to update the dictionaries $D[u]$ (Line 2). In each round, it consumes $\mathcal{T}_{max}$ time to perform the update process. In Lines 11-15, it consumes $O(\mathcal{T}_{max} \cdot n)$ time to calculate *TPPR* of every vertex. Therefore, the time complexity of Algorithm 1 is $O(\mathcal{T}_{max} \cdot (m+n))$. □

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 4.6. ***[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.*

Proof. By Definition 2.4, we have $\rho_S(u) = \sum_{v \in N_S(u)} tppr(v)$ and $\rho_H(u) = \sum_{v \in N_H(u)} tppr(v)$. Since $S \subseteq H$, $N_S(u) \subseteq N_H(u)$, we have $\rho_S(u) \leq \rho_H(u)$. □

By Lemma 4.6, 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). In particular, Algorithm 2 first calls Algorithm 1 to calculate

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

Input: temporal graph $\mathcal{G}$; query vertex q; teleportation probability α
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:         $D[v][t] = D[v][t] + (1-\alpha) D[u][t_1] P(u^{t_1} \rightarrow <u, v, t>)$
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:         $D[u][t] = D[u][t] + (1-\alpha) D[v][t_2] P(v^{t_2} \rightarrow <v, u, t>)$
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*

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 2.4 (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 (Line 6-7). Finally, it returns $CC(R, q)$ as the exact *query-centered* temporal community (Line 13).

Theorem 4.7. *Algorithm 2 can identify the exact* query-centered *temporal community. 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.*

Proof. Let S be the exact *query-centered* temporal community. In Lines 4-12, Algorithm 2 executes the greedy removing process. That is, in each round, it greedily deletes the vertex with the minimum query-biased temporal degree. Consider the round t when the first vertex u of S is deleted. Let V_t be the vertex set from the beginning of round t. Clearly, S is the subset of V_t because u is the first deleted vertex of S. This implies that there must be a connected subgraph G_H of G_{V_t} such that $G_S \subseteq G_H$. Thus, $\rho_S(u) \leq \rho_H(u)$ according to Lemma 4.6. Moreover, $\rho_H(w) \geq \rho_H(u)$ for any $w \in H$ since u has the minimum query-biased temporal degree in V_t. Thus, $\rho_H(w) \geq \rho_H(u) \geq \rho_S(u)$, which implies that H has optimal minimum query-biased temporal degree. Since Algorithm 2 maintains the optimal solution during greedy removing process in Lines 8-9, H will be returned as the exact *query-centered* temporal community in Line 13.

Algorithm 2 first consumes $O(\mathcal{T}_{max} \cdot (m+n))$ time to calculate the *TPPR* for each vertex (Line 1). Subsequently, it consumes $O(n + \bar{m})$ time to initialize the query-biased temporal degree (Line 3). Finally, it consumes $O(n \log n + \bar{m})$ time to perform the greedy removing process (Lines 4-12). Thus, Algorithm 2 consumes a total

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

Input: temporal graph $\mathcal{G}$; query vertex q; teleportation probability α
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

of $O(\mathcal{T}_{max} \cdot (m + n) + n \log n + \bar{m})$. Algorithm 2 takes $O(\mathcal{T}_{max} \cdot n)$ extra space to maintain dictionaries of Algorithm 1 for computing *TPPR*. Additionally, we also take $O(m + n)$ space to maintain the entire temporal graph. Thus, the space complexity of Algorithm 2 is $O(\mathcal{T}_{max} \cdot n + m)$. □

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 be further reduced to $O(\mathcal{T}_{max} \cdot m)$. Moreover, Algorithm 2 is even 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 for calculating the exact *TPPR* of each vertex. Therefore, Algorithm 2 is nearly optimal.

Remark. We can simply adapt Algorithm 2 to solve Problem 2. Specifically, in Line 1, we can get $tppr_q$ by executing Compute_tppr $(\mathcal{G}, q, \alpha)$ for each $q \in S$, in which S is the query vertex set. Then, we modify Line 3 as $\rho(u) \leftarrow \sum_{v \in N_V(u)} \sum_{q \in S} tppr_q(v)/|S|$ and the iteration terminates (i.e., Lines 4-12) once the current search space is empty or any query vertex $q \in S$ is removed or there is no connected component containing S. Finally, we return the vertex set from the maximal connected component of G_R containing S.

Discussion for *EGR*. Although *EGR* has near-linear time complexity, it is still inefficient for handling huge temporal graphs, especially for processing online real-time queries. For example, on the DBLP dataset, *EGR* takes 47 seconds to process a query (see Section 6), which is disruptive to the online user experience. The reasons can be explained as follows: (1) It needs to compute the *TPPR* for all vertices in advance, which dominates the time of *EGR*. In particular, *EGR* takes 99% of the time to compute *TPPR* on most datasets. (2) Computing *TPPR* and the greedy removing process are isolated, which makes the search space of *EGR* relatively large. Fortunately, in many real-life scenarios, users may allow some inaccuracy for better response time in large networks. Thus, it is desirable to devise approximate solutions for queries. Inspired by this, we propose an approximate local search algorithm to tackle these issues.

5 APPROXIMATE TWO-STAGE 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]. Specifically, *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 instead of vertices. Note that we set $r(\vec{e}) \geq 1/m$ in Algorithm 3 to speed up the propagation and enhance the subsequent pruning technologies.

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

Proof. Let $nnz(\mathbf{s})$ and $\mathbf{e}_i(\mathbf{s})$ be the number of non-zero elements in $\mathbf{s}$ and the one-hot vector with only value-1 entry corresponding to the i-th non-zero element in $\mathbf{s}$, respectively. Thus, we can write $\mathbf{s} = \sum_{i=1}^{nnz(\mathbf{s})} s_i \mathbf{e}_i(\mathbf{s})$, where s_i is the i-th non-zeros element in $\mathbf{s}$. According to the linearity [1] and Equation 3, we have $\widetilde{ppr}(\alpha, \widetilde{\chi_q}) = \sum_{i=1}^{|\vec{e}_q^{out}|} (1/|\vec{e}_q^{out}|) \widetilde{ppr}(\alpha, \mathbf{e}_i(\widetilde{\chi_q}))$. Furthermore, according to Equation 3 and 4, we have $\widetilde{ppr}(\alpha, \mathbf{e}_i(\widetilde{\chi_q}))(\vec{e}) = \pi(\widetilde{\chi_q}^i, \vec{e}) + \sum_{\vec{e}_j} r(\widetilde{\chi_q}^i, \vec{e}_j)$ $PPR(\vec{e}_j, \vec{e})$, where $\widetilde{\chi_q}^i$ is the ordered temporal edge corresponding to the i-th non-zero element of $\widetilde{\chi_q}$. Thus, $\rho_H(u) = \sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}}$ $\widetilde{ppr}(\alpha, \widetilde{\chi_q})(\vec{e}) = \sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}} \sum_{i=1}^{|\vec{e}_q^{out}|} (1/|\vec{e}_q^{out}|) \sum_{\vec{e}_j} r(\widetilde{\chi_q}^i, \vec{e}_j)$ $PPR(\vec{e}_j, \vec{e}) + \sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}} \sum_{i=1}^{|\vec{e}_q^{out}|} (1/|\vec{e}_q^{out}|) \pi(\widetilde{\chi_q}^i, \vec{e})$. So, $\rho_H(u)$ $\geq \sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}} \sum_{i=1}^{|\vec{e}_q^{out}|} (1/|\vec{e}_q^{out}|) \pi(\widetilde{\chi_q}^i, \vec{e}) = \sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}}$ $\pi(\vec{e})$. $\sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}} \sum_{i=1}^{|\vec{e}_q^{out}|} (1/|\vec{e}_q^{out}|) \sum_{\vec{e}_j} PPR(\vec{e}_j, \vec{e}) r(\widetilde{\chi_q}^i, \vec{e}_j) =$ $\sum_{\vec{e}_j} \sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}} PPR(\vec{e}_j, \vec{e}) \sum_{i=1}^{|\vec{e}_q^{out}|} (1/|\vec{e}_q^{out}|) r(\widetilde{\chi_q}^i, \vec{e}_j) = \sum_{\vec{e}_j}$ $\sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}} PPR(\vec{e}_j, \vec{e}) r(\vec{e}_j) = \sum_{\vec{e}_j} r(\vec{e}_j) \sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}} PPR$ $(\vec{e}_j, \vec{e}) \leq \sum_{\vec{e}_j} r(\vec{e}_j)$. So, $\rho_H(u) \leq \sum_{v \in N_H(u)} \sum_{\vec{e} \in \vec{e}_v^{in}} \pi(\vec{e}) + \sum_{\vec{e}_j} r(\vec{e}_j)$. □

Based on Lemma 5.1, we present two powerful pruning techniques used in the expanding stage. These techniques can delete

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

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

some unqualified vertices or early terminate the expanding stage with theoretical guarantees. For simplicity, we denote C as the expanded vertex set for the following reducing stage, Q as the candidate vertices which are neighbors of C and not in C, $\widehat{\beta}$ as the best estimate of 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]*** *Given a vertex v, we can safely prune the vertex v if $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in N_V(v)} \widehat{tppr}(w) < \widehat{\beta}$.*

Proof. Assume that there is a *query-centered* temporal community S such that $v \in S$. Since the query-biased temporal degree is monotonically increasing by Lemma 4.6, $\rho_S(v) \leq \rho_V(v)$ for v holds due to $S \subseteq V$. According to Lemma 5.1, we have $\rho_S(v) \leq \rho_V(v) \leq \sum_{\vec{e}} r(\vec{e}) + \sum_{w \in N_V(v)} \widehat{tppr}(w)$. If $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in N_V(v)} \widehat{tppr}(w) < \widehat{\beta}$, we have that $\rho_S(v) < \widehat{\beta}$. Clearly, $\min\{\rho_S(u) | u \in S\} \leq \rho_S(v) < \widehat{\beta}$, which contradicts with S being a *query-centered* temporal community. So, we can safely remove v without loss of accuracy. □

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

Proof. Let $N_V(C) = \{u | N_V(u) \cap C \neq \emptyset\}$, we can clearly prune every vertex $u \in N_V(C)$ if $Q = \emptyset$. Assume that there is a query-centered temporal community S containing C, we have $N_V(v) \cap C = \emptyset$ for any $v \in S \setminus C$. Namely, G_S is a disconnected subgraph, which contradicts with G_S is connected by (i) of Definition 2.5. So, we can safely stop the expanding stage when $Q = \emptyset$. □

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 $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in Q} \widehat{tppr}(w) < \widehat{\beta}$.*

Proof. By Algorithm 3 and 4, we have $\widehat{tppr}(v) \neq 0$ for vertex $v \in D$. For any unvisited vertex $u \in V \setminus D$, we assume that there is a *query-centered* temporal community S such that $u \in S$. Thus, we have $\sum_{w \in N_S(u)} \widehat{tppr}(w) = \sum_{w \in N_S(u) \cap D} \widehat{tppr}(w) \leq \sum_{w \in Q} \widehat{tppr}(w) + \sum_{w \in N_S(u) \cap (D \setminus (C \cup Q))} \widehat{tppr}(w) = \sum_{w \in Q} \widehat{tppr}(w)$, because $D \setminus (C \cup Q)$ is the unqualified vertex set during the expanding stage and $S \cap (D \setminus (C \cup Q)) = \emptyset$. If $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in Q} \widehat{tppr}(w) < \widehat{\beta}$, we have $\sum_{\vec{e}} r(\vec{e}) + \sum_{w \in N_S(u)} \widehat{tppr}(w) < \widehat{\beta}$. Moreover, according to Lemma 5.1, we have that $\rho_S(u) < \widehat{\beta}$. Clearly, $\min\{\rho_S(v) | v \in S\} \leq \rho_S(u) < \widehat{\beta}$, which contradicts with S is a *query-centered* temporal community. So, we can safely remove u. That is, we can prune any vertex $u \in V \setminus D$ if $\sum_{\vec{e}_j} r(\vec{e}_j) + \sum_{w \in Q} \widehat{tppr}(w) < \widehat{\beta}$. There is no evidence to remove any vertex $u \in Q$, thus we directly set $C = C \cap Q$ for simplicity. □

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

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

Input: temporal graph $\mathcal{G}$; query vertex q; teleportation probability α
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 $\widehat{tppr}$

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 minimum 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.*

Proof. Algorithm 3 consumes $O(|N^{>}(\vec{e})|)$ time to execute the propagation process for each ordered temporal edge $\vec{e}$. Thus, in Lines 6-7 of Algorithm 4, it takes $O(\sum_{\vec{e} \in \vec{e}_u^{out}} |N^{>}(\vec{e})|)$ time for every vertex $u \in C$. So, Algorithm 4 consumes $O(\sum_{u \in C} \sum_{\vec{e} \in \vec{e}_u^{out}} |N^{>}(\vec{e})|)$ in total. Algorithm 4 uses $O(m)$ extra space to maintain the reserve r and residue π for estimating the query-biased temporal degree. Besides, we also need $O(m+n)$ space to maintain the whole temporal graph. So, the space complexity of Algorithm 4 is $O(n+m)$. □

Remark. By Theorem 5.5, the time complexity of Algorithm 4 depends on the vertex set C, while our experiments (Section 6) show C is typically very small due to the proposed powerful pruning techniques in Lemma 5.2, 5.3 and 5.4. Thus, the expanding stage can drastically delete many unqualified vertices, saving the time of the following reducing stage.

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 $\epsilon \cdot \sum_{v \in N_R(u)} \widehat{tppr}(v) < \max\{\sum_{w \in N_C(v)} \widehat{tppr}(w)|v \in C\} + \sum_{\vec{e}} r(\vec{e})$.*

Proof. Assume that there is an ϵ-approximate *QTCS* $H \subseteq R$ such that $u \in H$, we have $\epsilon \cdot \rho_H(u) \geq \beta^*$ due to Definition 5.6. Thus, if $\epsilon \cdot \rho_R(u) < \beta^*$, we can derive that there does not exist an ϵ-approximate *QTCS* $H \subseteq R$ such that $u \in H$. Moreover, $\rho_R(u) \geq \sum_{v \in N_R(u)} \widehat{tppr}(v)$ by Lemma 5.1. So, $\epsilon \cdot \sum_{v \in N_R(u)} \widehat{tppr}(v) < \beta^*$. On the one hand, since C covers all target community members (Algorithm 4), $\beta^* \leq \max\{\rho_C(v)|v \in C\}$ due to Definition 2.5 and Lemma 4.6. On the other hand, we have $\max\{\rho_C(v)|v \in C\} \leq \max\{\sum_{w \in N_C(v)} \widehat{tppr}(w)|v \in C\} + \sum_{\vec{e}} r(\vec{e})$ by Lemma 5.1. Therefore, $\epsilon \cdot \sum_{v \in N_R(u)} \widehat{tppr}(v) < \max\{\sum_{w \in N_C(v)} \widehat{tppr}(w)|v \in C\} + \sum_{\vec{e}} r(\vec{e})$. So, vertex u can be removed from R 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 uses $flag$ to indicate whether query vertex q is removed or not. If $flag$ 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). Clearly, 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 $G_C = \{(u, v) \in E|u, v \in C\}$.*

Proof. Algorithm 5 first takes $O(|G_C|)$ time to compute the estimated query-biased temporal degree (Lines 2-3). Then, in Lines 5-21, it executes the iterative update process. In each round, it takes $O(|G_C|)$ time to remove unpromising vertices and update the search space. Moreover, there are at most $\log_2(\frac{temp}{\min\{\widehat{\rho}(u)|u \in C\}})$ rounds due to the binary search. Since $temp \leq 1$ and $\min\{\widehat{\rho}(u)|u \in C\} \geq 1/m$ (by the previous expanding stage), $\log_2(\frac{temp}{\min\{\widehat{\rho}(u)|u \in C\}}) \leq \log m$. Putting these together, Algorithm 5 takes $O(\log m \cdot |G_C|)$ time in total. Algorithm 5 needs $O(|C|)$ space to store $\widehat{\rho}$ for the vertex set C. And we also require $O(|G_C|)$ space to store the subgraph graph G_C. So, the space complexity of Algorithm 5 is $O(|G_C|)$. □

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 α

Output: the ϵ-approximate *QTCS*

1: $R \leftarrow C$; $\widehat{\rho} \leftarrow \{\}$; $flag \leftarrow True$
2: **for** $u \in C$ **do**
3: $\quad \widehat{\rho}(u) \leftarrow \sum_{v \in N_C(u)} \widehat{tppr}(v)$
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: $\quad Q \leftarrow \emptyset$; $D \leftarrow \emptyset$
7: $\quad$ **for** $u \in R$ **do**
8: $\quad\quad$ **if** $\overline{\epsilon}\widehat{\rho}(u) \leq temp$ **then**
9: $\quad\quad\quad Q.push(u)$
10: $\quad\quad\quad$ **if** $u == q$ **then**
11: $\quad\quad\quad\quad flag \leftarrow False$; $Q \leftarrow \emptyset$
12: $\quad$ **while** $Q \neq \emptyset$ **do**
13: $\quad\quad u \leftarrow Q.pop$ and $D \leftarrow D \cup \{u\}$
14: $\quad\quad$ **for** $v \in N_R(u)$ and $v \notin D$ **do**
15: $\quad\quad\quad \widehat{\rho}(v) = \widehat{\rho}(v) - \widehat{tppr}(u)$
16: $\quad\quad\quad$ **if** $\overline{\epsilon}\widehat{\rho}(v) \leq temp$ **then**
17: $\quad\quad\quad\quad Q.push(v)$
18: $\quad\quad\quad\quad$ **if** $v == q$ **then**
19: $\quad\quad\quad\quad\quad flag \leftarrow False$; $Q \leftarrow \emptyset$
20: $\quad$ **if** $flag$ **then**
21: $\quad\quad \epsilon \leftarrow \overline{\epsilon}$; $R \leftarrow R \setminus D$; $\overline{\epsilon} \leftarrow \overline{\epsilon}/2$
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 ϵ is the corresponding approximation ratio

Remark. We can simply adapt Algorithm 4 and 5 to solve Problem 2. Let S is the query vertex set. For Algorithm 4, we set $r(\vec{e}) \leftarrow 1/|\vec{e}_S^{out}|$ for all $\vec{e} \in \vec{e}_S^{out}$ where $\vec{e}_S^{out} = \cup_{q \in S} \vec{e}_q^{out}$ (Line 2), $Q \leftarrow S$, $D \leftarrow S$ (Line 3). For Algorithm 5, the iteration terminates (i.e., Lines 5-21) once any query vertex $q \in S$ is removed or there is no connected component containing S. Finally, we return the vertex set from the maximal connected component of G_R containing S.

6 EXPERIMENTAL EVALUATION

In this section, we conduct comprehensive experiments to test the efficiency, effectiveness, and scalability of the proposed solutions. These experiments are executed on a server with an Intel Xeon 2.50GHZ CPU and 32GB memory running Ubuntu 18.04.

6.1 Experimental setup

Datasets. We evaluate our solutions on eight graphs[5] which are used in recent work [6, 24, 34] as benchmark datasets (Table 1). 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.

[5] http://snap.stanford.edu/, http://konect.cc/, http://www.sociopatterns.org/

Table 1: 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

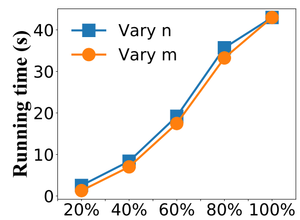
Table 2: State-of-the-art methods

Methods		Temporal	Remark
Community Detection	*MPC* [34]	✓	Clique-based
	PCore[24]	✓	k-Core-based
	DBS [6]	✓	Density-based
Community Search	*CSM* [8]	×	k-Core-based
	TCP [17]	×	k-Truss-based
	PPR_NIBBLE [1]	×	Conductance-based
	MTIS [44]	✓	Inefficiency-based
	MSCS [14]	✓	k-Core-based
	QTCS_Baseline	✓	*TPPR*-based
	EGR	✓	*TPPR*-based
	ALS	✓	*TPPR*-based

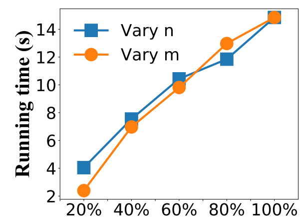
Algorithms. We implement several state-of-the-art methods for comparison (Table 2). Specifically, *CSM* [8] identifies the maximal k-core containing the query vertex with largest k. *TCP* [17] applies the triangle connectivity and k-truss to model the higher-order truss community. *PPR_NIBBLE* [1] is a local clustering method, which adopts the conductance as the criterion of a community. Note that *CSM*, *TCP*, and *PPR_NIBBLE* are static community search methods. *MPC* [34] extends the concept of clique to adapt the temporal setting. *PCore*[24] maintains persistently a k-core structure. *DBS* [6] uses the density and duration to model bursting communities. But *MPC*, *PCore*, *DBS* address the problem of temporal community detection. Thus, to fit our problem, we first find all possible communities by the predefined criteria[6, 24, 34], and then select the target community containing the query vertex from these communities. *MTIS* [44] and *MSCS* [14] are temporal community search methods. In particular, *MTIS* and *MSCS* model the temporal cohesiveness of the community by extending the network-inefficiency and k-core to temporal setting, respectively. *QTCS_Baseline* is an intuitive variant model (Definition 2.3). *EGR* and *LAS* are our proposed methods.

Effectiveness metrics. There are two effectiveness metrics: temporal density (*TD*) and temporal conductance (*TC*) [6, 41]. 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 [29, 45]. 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.



(a) *EGR*



(b) *ALS*

Figure 3: Scalability testing

Table 3: 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

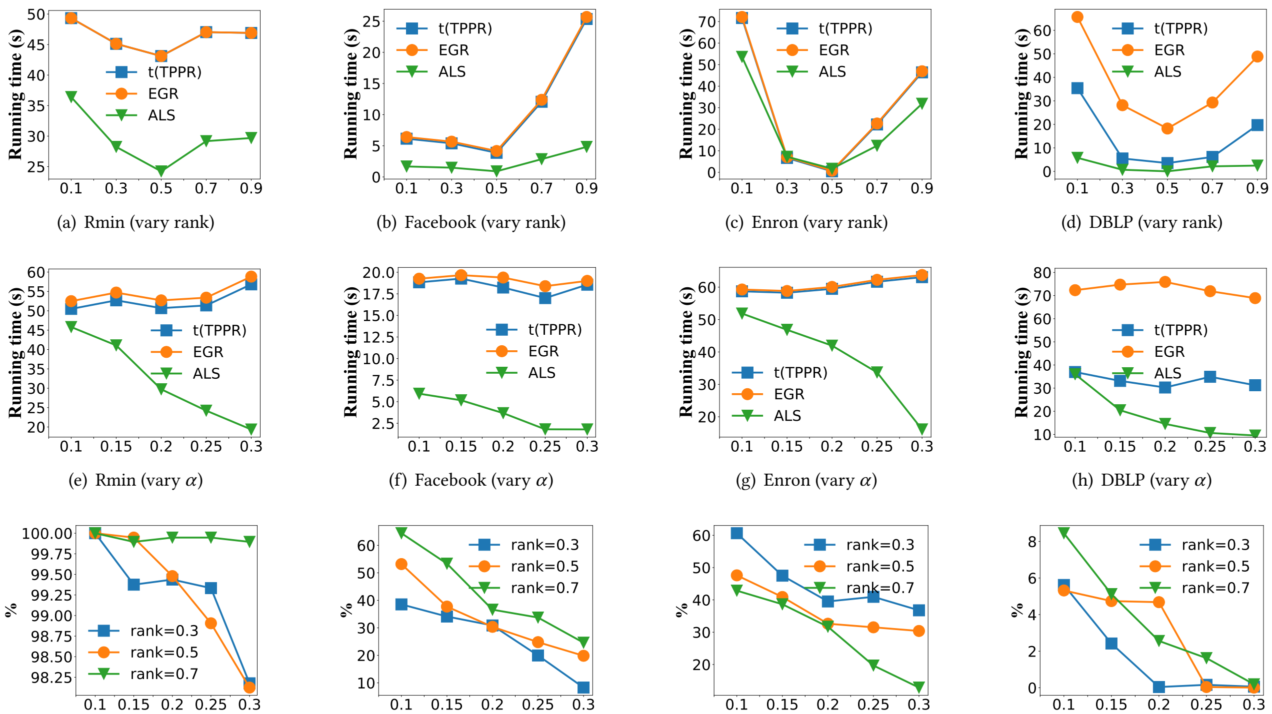
6.2 Efficiency testing

Exp-1: Running time of various temporal methods. From Table 4, 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 the 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) they are NP-hard in theory, thus they cannot be solved in polynomial time unless P=NP. 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 for 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),

Table 4: 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	4
PCore	35913.248	28561.989	>48h	>48h	148.447	>48h	21221.338	24.493	7
DBS	47.363	1722.200	2150.320	48.792	33179.300	**91.411**	614.998	2462.040	5
MTIS	>48h	42.339	154.161	>48h	152.064	>48h	>48h	78252.764	8
MSCS	241.613	25.204	28.786	753.186	42.699	859.255	1290.521	3083.327	6
QTCS_Baseline	47.283	1.879	6.703	16.107	1.800	226.457	82.66	45.391	<u>2</u>
EGR	47.293	1.881	6.711	16.067	2.604	224.592	83.168	47.259	<u>3</u>
ALS	**28.326**	**1.030**	**3.049**	**3.038**	**1.257**	191.889	**30.557**	**13.707**	**<u>1</u>**



(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

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 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* is 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. Fig 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 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 3 shows the results. As can be seen, *EGR* and *ALS* scales near-linear w.r.t. the size of the temporal subgraphs. These results indicate that our proposed algorithms can handle massive temporal networks.

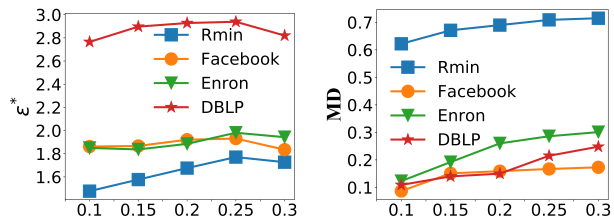
Exp-5: Memory overhead of *EGR* and *ALS*. From Table 3, we can see that the memory overhead of *EGR* and *ALS* is less than twice that of the original graph. Moreover, we can also see that the memory overhead of *ALS* is less than *EGR* in six of the eight datasets. This is because *ALS* is a local search algorithm, thus fewer vertices may be visited (Exp-3 also confirms this), which further results in less space used to store reserve and residue for estimating the *TPPR* values. But, *EGR* is a global algorithm, which needs to store $D[u]$ for computing the exact *TPPR* values. These results demonstrate that *EGR* and *ALS* can achieve near-linear space complexity, which is consistent with our theoretical analysis in Section 4 and 5.

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	4/9/3
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	5/8/4
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	2/10/9
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	9/**1**/8
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	7/4/5
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	8/2/7
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	10/7/10
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	6/6/6
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 5: The quality of *ALS* with various α.

6.3 Effectiveness testing

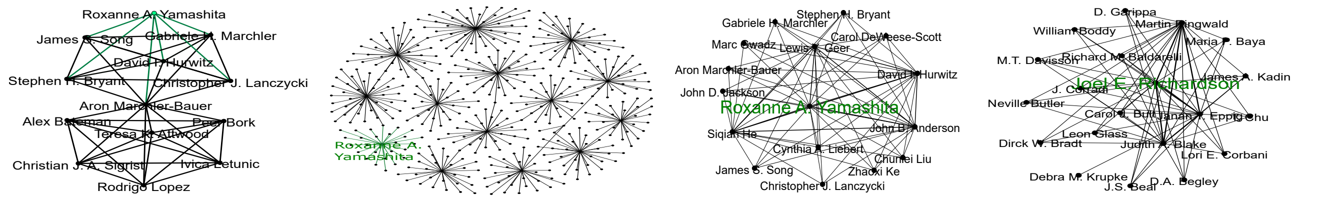
Exp-6: Effectiveness of different methods. Table 5 reports our results. For the *TC* metric, we have: (1) our model achieves the best scores on seven of the eight datasets. This is because our model can mitigate the *query drift* issue (Section 3.2), resulting in that it can keep good temporal separability by removing out 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) *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. In a nutshell, these results indicate that existing models cannot optimize our proposed objective function well, and our model is much denser and more separable in terms of temporal feature than existing models.

Remark. Optimizing *TD* and *TC* simultaneously is very challenging (or even impossible). So, our model is a trade-off between them. The reasons can be explained as follows. (1) Although the *TD* score of our model is slightly worse than the baselines (i.e., *MPC* and *DBS*), our algorithm is at least three orders of magnitude faster than the baselines. Thus, our solutions achieve better runtime by losing a small amount of quality, which is particularly important for processing massive datasets. (2) As we all know, a good community not only requires the vertices in the community to be internally cohesive (*TD*) but also separates from the remainder of the network (*TC*). In Table 5, we can see that *MPC* and *DBS* rank ninth and eighth in terms of *TC*, respectively, but our model is the best.

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 theoretically 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 5 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 5(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 5(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 2.4.

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. 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. Note that the community identified by *QTCS_Baseline* contains more than 1,000 authors (since it is too large to show in



(a) *PCore* (b) *MSCS* (c) *Our model* (d) *Our model*

Figure 6: Case studies on DBLP. (a-c) (resp. (d)) are the communities of Prof. Roxanne A. Yamashita (resp. Joel E. Richardson)

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. Thus, *QTCS_Baseline* suffers from the *query drift* issue. On the other hand, as shown in Figure 6 (c), the community obtained by our model is a meaningful *query-centered* temporal community and does not cause the *query drift* issue. This is because Roxanne A. Yamashita is *centered* in the detected community and worked closely and frequently with other researchers. Besides, these researchers mainly investigate conserved sequence, amino acid sequence, and proteins, which is consistent with Roxanne A. Yamashita. 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 6 (a), we can see that Roxanne A. Yamashita is marginalized, and the members on the upper and lower parts are connected by the hub vertex Aron Marchler-Bauer. 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 6 (c) do not appear in Figure 6 (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 6 (b), we can see that the community obtained by *MSCS* is a connected subgraph composed of multiple stars. Furthermore, Figure 6 (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 our model in Figure 6 (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, resulting in the query vertex being marginalized. Thus, *PCore* and *MSCS* also suffer from the *query drift* issue. In summary, these case studies further indicate that our model is indeed more effective than the other models to search *query-centered* temporal communities.

7 RELATED WORK

Community detection. Existing studies mainly rely on structure-based approach to identify all communities from graphs, including modularity optimization [32], spectral analysis [9], hierarchical clustering [36] and cohesive subgraph discovering [4]. However, all these methods do not consider the temporal dimension of networks. Until recently, a few researches have been done on community detection over temporal networks [6, 24–26, 31, 34, 37, 52]. For instance, Lin et al. [26] proposed the stable quasi-clique to capture the stability of cohesive subgraphs. Ma et al. [31] studied the heavy subgraphs for detecting traffic hotspots. However, all these researches are query-independent, which are often costly to mine all communities. Thus, they cannot be directly extended to perform online community search on temporal networks.

Community search. As a meaningful counterpart, community search has recently become a focal point of research in network analysis [12, 19]. 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 [2, 8, 42], k-truss [17, 27], clique [7, 53], density [48], connectivity [39, 40, 43] and conductance [1, 3, 51]. Recently, Wu et al. [48] observed the above approaches exist 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 (see Section 3.2 for details). Besides, graph diffusion-based local clustering methods have also been considered. For example, Tong et al. [43] 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 [11, 18, 28], location-based social networks [5, 10], multi-valued graphs [23] and heterogeneous information networks [13, 21]. However, they ignore the temporal properties of networks that frequently appear in applications. Recently, two studies are done on temporal community search [14, 44]. But, they suffer from several defects (Section 1, 3.2 and 6).

Temporal proximity. Node-to-node proximity is a fundamental concept in graph analysis, which captures the relevance between two nodes in a graph [49]. Perhaps, the most representative proximity model is the Personalized PageRank [29, 33, 45] due to its effectiveness and solid theoretical foundation. However, this model only considers graph structural information and ignores the temporal properties. Recently, several studies were done on temporal proximity. For example, [16, 35] first converted the temporal graph into a weighted graph and then applied the traditional method over the weighted graph to define the temporal PageRank. These methods, however, only consider the temporal information of two directly-connected vertices, missing higher-order temporal and structural information. [30] adopted the fourth-order tensor to represent the temporal network and calculated the eigenvector of the tensor to rank the vertices, which is inefficient for handling large graphs. The most related work to ours is [38]. However, [38] focuses on modeling the importance of vertices at a certain timestamp t. Thus, the method is to track the evolution of the importance of vertices. However, our *TPPR* models the importance of vertices on the entire graph by non-trivially considering all timestamps. Thus, our *TPPR* considers more structural and temporal information, which is more reasonable to capture temporal proximity.

8 CONCLUSION

In this work, we are the first to introduce and address the *query-centered* temporal community search problem. In particular, we first develop the Time-Constrained Personalized PageRank to capture the temporal proximity between 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.

REFERENCES

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

[2] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *Data Min. Knowl. Discov.* 29, 5 (2015), 1406–1433.

[3] Yuchen Bian, Yaowei Yan, Wei Cheng, Wei Wang, Dongsheng Luo, and Xiang Zhang. 2018. On Multi-query Local Community Detection. In *ICDM*. 9–18.

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

[5] 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.

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

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

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

[9] 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), P10012.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

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

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

[20] 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.

[21] 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.

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

[23] 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*.

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

[25] L. Lin, P. Yuan, R. Li, and H. Jin. 2021. Mining Diversified Top-r Lasting Cohesive Subgraphs on Temporal Networks. *IEEE Transactions on Big Data* (2021), 1–1. https://doi.org/10.1109/TBDATA.2021.3058294

[26] 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.

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

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

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

[30] 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.

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

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

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

[34] 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.

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

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

[37] 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.

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

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

[40] Natali Ruchansky, Francesco Bonchi, David García-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.

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

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

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

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

[45] 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.

[46] 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.

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

[48] 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.

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] Yifei Zhang, Longlong Lin, Pingpeng Yuan, and Hai Jin. 2022. Significant Engagement Community Search on Temporal Networks. In *DASFAA*. 250–258.