Stable Subgraph Isomorphism Search in Temporal Networks

Qi Zhang, Rong-Hua Li, Hongchao Qin, Guoren Wang, Zhiwei Zhang, and Ye Yuan

Abstract—In this paper, we study a new problem of seeking stable subgraph isomorphisms for a query graph in a temporal graph. To solve our problem, we first develop a pruning-based search algorithm using several new pruning tricks to prune the unpromising matching results during the search procedure. To further improve the efficiency, we propose a novel index structure called BCCIndex, based on an idea of bi-connected component decomposition of the query graph, which can efficiently support the stable subgraph isomorphism search. Equipped with the BCCIndex, we present an efficient query processing algorithm based on a carefully designed tree join technique. We conduct extensive experiments to evaluate our algorithms on four large real-life datasets, and the results demonstrate the efficiency and effectiveness of our algorithms.

Index Terms—Graph query, temporal graphs, subgraph isomorphism search

1 Introduction

Subgraph isomorphism (or subgraph matching) search is a fundamental problem in graph analysis. Given a data graph G and a query graph q, subgraph isomorphism search is a problem of finding all subgraphs in G that are isomorphic to q. Such a problem has been found in a wide range of applications in network analysis, including mining motif substructures in biological networks [1], analyzing the evolution of social networks [2], and finding syntheses of target structures in chemistry networks [3].

In applications such as analysis of collaboration networks, communication networks, financial transaction networks, and online social networks, edges in these networks are often associated with temporal information. For example, in a collaboration network of scientific papers, each co-authorship relation contains two authors and the time when they co-authored a paper. In an email communication network, each email consists of a sender, a receiver, as well as the time when the email was sent. In a financial transaction network, each transaction includes a sender and a receiver, as well as the time when the transaction was completed. In an online social network, each instant message may include two users and the time when the message was sent. Such networks are typically modeled as temporal graphs [4], [5]. In a temporal graph, each edge is represented as a triplet (u, v, t) where u, v are the end nodes of the edge and t denotes the interaction time between u and v. Consider a time sequence $\{t_0, ..., t_i, ..., t_T\}$. Suppose that $(t_i - t_{i-1})$ is a constant. Then, we refer to a graph as a snapshot if its temporal edges appear at the time interval $(t_{i-1}, t_i]$.

Although the subgraph isomorphism search techniques have been widely used in many graph analysis applications, most previous studies on subgraph isomorphism query are mainly tailored for traditional static and labeled graphs which ignore the temporal information, thus cannot be applied to analyze temporal graphs. In this paper, we focus on the keyword "temporal" and study a new problem of seeking stable subgraph isomorphisms in unlabeled temporal graphs. Our goal is to find all subgraph isomorphisms for a given query graph that are stable over time. More specifically, for a query graph q and a stability threshold θ, the stable subgraph isomorphism search problem is to identify all subgraphs that are isomorphic to q in no less than θ snapshots.

Such a stable subgraph isomorphism search problem can be used for many temporal graph analysis applications. For example, in a collaboration network, a stable k-clique subgraph represents that the k authors have co-authored many papers multiple times, indicating a long-term collaboration among them. A stable star-like structure may reveal a stable cooperative team in multi-discipline areas that have co-authored many papers over time. Finding these stable structures may be helpful for identifying the team of experts to conduct a particular research project. In an email communication network between staff in a company, a stable star structure may reveal the staff's implicit leadership, as a leader may often send tasks to the other staff and the staff may report their work to the leader frequently. In a financial transaction network, a stable small-circle structure may represent a kind of financial fraud behavior [6]. Finding such stable small-circle isomorphisms in a temporal financial transaction network can help detect financial fraud behaviors. In addition, searching stable subgraph isomorphisms in temporal graphs can find stable communities and reveal

• *The authors are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: {qizhangcs, yuan-ye}@bit.edu.cn, {lironghuabit, wanggrbit}@126.com, qhc.neu@gmail.com, cszwzhang@outlook.com.*

Manuscript received 27 Apr. 2021; revised 3 May 2022; accepted 12 May 2022. Date of publication 19 May 2022; date of current version 1 May 2023.
This work was supported in part by the National Key Research and Development Program of China under Grant 2020AAA0108503 and in part by NSFC under Grants 62072034 and U1809206.
(Corresponding author: Guoren Wang.)
Recommended for acceptance by T. Weninger.
Digital Object Identifier no. 10.1109/TKDE.2022.3175800

some implicit features of a network which may be useful for understanding the structure and function of a network.

The stable subgraph isomorphism search problem is NP-hard because when $\theta = 1$, it degenerates a problem of finding subgraph isomorphisms in every snapshot of $\mathcal{G}$, which is NP-hard [7], [8]. To solve the stable subgraph isomorphism search problem, a straightforward solution is to use a traditional subgraph isomorphism query algorithm, such as the Ullmann algorithm [9], to compute all subgraph isomorphisms for a query graph in each snapshot, and then pick the stable subgraph isomorphisms among all snapshots. Clearly, such a solution is very costly, because the subgraph isomorphism search is a NP-hard problem. To efficiently compute the stable subgraph isomorphisms in a temporal graph, we develop a new pruning-based search algorithm equipped with several pruning tricks based on the temporal stability constraint, which can significantly reduce the unpromising intermediate results during the search procedure. To further improve the efficiency, we propose a novel index structure, called BCCIndex, based on a bi-connected component (BCC) decomposition technique which can efficiently support stable subgraph isomorphism query. Armed with the BCCIndex and a carefully-designed tree join technique, we develop an efficient query processing algorithm to find stable subgraph isomorphisms. To the best of our knowledge, our work is the first to apply the BCC indexing technique to solve the stable subgraph isomorphism search problem in temporal networks. In summary, we make the following contributions.

An Pruning Based Algorithm. We propose an pruning-based stable subgraph isomorphism search algorithm PruneSearch integrated with several pruning techniques to avoid exploring the unpromising intermediate results during the search procedure. We also present a parallel version of PruneSearch to improve the scalability of the algorithm.

An Index-Based Algorithm. We devise an index structure, namely, BCCIndex, based on a BCC decomposition technique. Equipped with the BCCIndex, we propose an index-based solution, i.e., BCCIndexSearch, to efficiently find stable subgraph isomorphisms based on a newly-developed tree join technique. We also propose a parallel BCCIndex construction algorithm and a parallel BCCIndexSearch algorithm to further improve the scalability.

Extensive Experiments. We conduct comprehensive experiments to evaluate the efficiency of the proposed algorithms using four large real-world temporal graphs. The results show that 1) BCCIndexSearch is very efficient which is around 1-3 orders of magnitude faster than PruneSearch; 2) the BCCIndex can be constructed in a reasonable time for large temporal graphs and also the size of BCCIndex is often not very large; 3) both the parallel PruneSearch and parallel BCCIndexSearch can achieve very high speedup ratios. In addition, we also conduct a case study on a collaboration network DBLP. The results show that our solutions can indeed find meaningful and stable research teams in DBLP.

Remark. Note that given a graph $G = (V, E)$ and a query graph $Q = (V_q, E_q)$, there are two concepts of subgraph isomorphism in the studies of graph analysis. The first is defined as an injective function $M : V_q \rightarrow V$ such that $\forall(u_i, u_j) \in E_q, (M(u_i), M(u_j)) \in E$, which is often called subgraph monomorphism. The second is an injective function $M : V_q \rightarrow V$ satisfying that $\forall(u_i, u_j) \in E_q, (M(u_i), M(u_j)) \in E$ and $\forall(u_i, u_j) \notin E_q, (M(u_i), M(u_j)) \notin E$, which is also known as induced subgraph isomorphism. In this paper, we focus on the definition of the former, i.e., subgraph monomorphism, and also use "subgraph isomorphism" to represent "subgraph monomorphism" in the following.

Organization. We introduce some important notations and formulate our problem in Section 2. The pruning-based search framework is presented in Section 3. Section 4 presents the index structure and the index construction method. The index-based query processing algorithm is proposed in Section 5. Section 6 reports the experimental results. We survey the related work in Section 7 and conclude this work in Section 8.

2 Preliminaries

Given an undirected and unlabeled temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ vertices and $m = |\mathcal{E}|$ temporal edges. Each temporal edge $e \in \mathcal{E}$ is a triplet (u, v, t), where u, v are vertices in $\mathcal{V}$, and t is the interaction time between u and v. We assume that t is an integer, because the timestamp is an integer in practice. The *de-temporal* graph of $\mathcal{G}$ is defined as $G = (V, E)$ by discarding all timestamps on the temporal edges and condensing the multiple edges between any two vertices into a single edge. Clearly, we have $V = \mathcal{V}$ and $E = \{(u, v)|(u, v, t) \in \mathcal{E}\}$. We denote the neighbors of a vertex u by $N_u(G)$, i.e., $N_u(G) = \{v \in V|(u, v) \in E\}$, and the degree of u by $deg_G(u) = |N_u(G)|$. Given a subset $S \subseteq V$, the subgraph of G induced by S is defined as $G_S = (V_S, E_S)$ where $V_S = S$ and $E_S = \{(u, v)|u, v \in S, (u, v) \in E\}$ and we denote as $G_S \subseteq G$. We omit the symbol G in the above notations when the context is clear.

Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we can extract a series of *snapshots* based on the timestamps. Considering an arithmetic time sequence $\{t_0, t_1, t_2, ..., t_T\}$ satisfying that $(t_i - t_{i-1})$ is a constant for each integer $i > 0$, the i-th snapshot of $\mathcal{G}$ is a de-temporal graph $G_i = (V, E_i)$ where E_i is a set of edges that are extracted from $\mathcal{E}$ in the time interval $(t_{i-1}, t_i]$ and $\mathcal{V}$ remains the same in general. Let T be the number of snapshots of $\mathcal{G}$ and we have $T \leq m$. Denote by $\mathcal{G}_T$ the set of all snapshots of $\mathcal{G}$ based on the time interval. In the experiments, we set $(t_i - t_{i-1})$ to a default value of 1 month/year which means that every snapshot contains all the temporal edges in a 1-month/year length sliding window. Fig. 1 illustrates a temporal graph $\mathcal{G}$ with 77 temporal edges and $T = 6$. The de-temporal graph of $\mathcal{G}$ is shown in Figs. 1b. Figs. 1c, 1d, 1e, 1f, 1g, and 1h are all the six snapshots from G_1 to G_6 of $\mathcal{G}$, respectively.

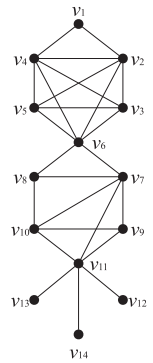
Before introducing the definition of stable subgraph embedding, we give the concepts of subgraph isomorphism and subgraph isomorphism embedding as follows.

Definition 1 (Subgraph isomorphism**).** *Given a query graph $q = (V_q, E_q)$, a data graph $g = (V_g, E_g)$, q is* subgraph isomorphic *to g if and only if there exists an injective function $M : V_q \rightarrow V_g$ such that $\forall(u_i, u_j) \in E_q, (M(u_i), M(u_j)) \in E_g$. We call g a subgraph isomorphism of q and denote by $q \simeq g$.*

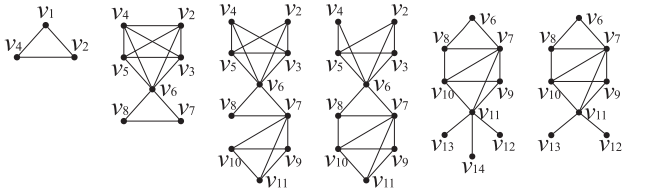
Definition 2 (Subgraph isomorphism embedding**).** *Given a query graph $q = (V_q, E_q)$ and its subgraph isomorphism graph $g = (V_g, E_g)$, a subgraph isomorphism embedding is an*

t	(u, v)
1	(v_1, v_2), (v_1, v_4), (v_2, v_4)
2	(v_2, v_3), (v_2, v_4), (v_2, v_5), (v_2, v_6), (v_3, v_4), (v_3, v_5), (v_3, v_6), (v_4, v_5), (v_4, v_6), (v_5, v_6), (v_6, v_7), (v_6, v_8), (v_7, v_8)
3	(v_2, v_3), (v_2, v_5), (v_2, v_6), (v_3, v_4), (v_3, v_5), (v_3, v_6), (v_4, v_5), (v_4, v_6), (v_5, v_6), (v_6, v_7), (v_6, v_8), (v_7, v_8), (v_7, v_9), (v_7, v_{10}), (v_7, v_{11}), (v_9, v_{10}), (v_9, v_{11}), (v_{10}, v_{11})
4	(v_2, v_3), (v_2, v_5), (v_2, v_6), (v_3, v_5), (v_3, v_6), (v_4, v_5), (v_4, v_6), (v_5, v_6), (v_6, v_7), (v_6, v_8), (v_7, v_8), (v_7, v_9), (v_7, v_{10}), (v_7, v_{11}), (v_8, v_{10}), (v_9, v_{10}), (v_9, v_{11}), (v_{10}, v_{11})
5	(v_6, v_7), (v_6, v_8), (v_7, v_8), (v_7, v_9), (v_7, v_{10}), (v_7, v_{11}), (v_8, v_{10}), (v_9, v_{10}), (v_9, v_{11}), (v_{10}, v_{11}), (v_{11}, v_{12}), (v_{11}, v_{13}), (v_{11}, v_{14})
6	(v_6, v_7), (v_6, v_8), (v_7, v_8), (v_7, v_9), (v_7, v_{10}), (v_7, v_{11}), (v_8, v_{10}), (v_9, v_{10}), (v_9, v_{11}), (v_{10}, v_{11}), (v_{11}, v_{12}), (v_{11}, v_{13})

(a) Temporal edges in $\mathcal{G}$



(b) De-temporal graph G



(c) G_1 (d) G_2 (e) G_3 (f) G_4 (g) G_5 (h) G_6

Fig. 1. Basic concepts of a temporal graph $\mathcal{G}$.

injective mapping $M : V_q \to V_g$. We use g_M to represent the subgraph isomorphism graph specified by the mapping M.

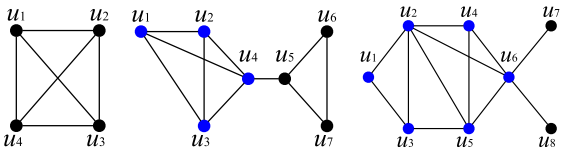
Example 1. Consider a graph G in Fig. 1b and a query graph q_1 in Fig. 2a. In G, the 4-clique C induced by the vertex set $V_g = \{v_2, v_3, v_4, v_5\}$ is a subgraph isomorphism of q_1. The injective mapping $M(u_1 \to v_2, u_2 \to v_3, u_3 \to v_4, u_4 \to v_5)$ is a subgraph isomorphism embedding. And the mapping $M'(u_1 \to v_5, u_2 \to v_4, u_3 \to v_3, u_4 \to v_2)$ is also a subgraph isomorphism embedding. Clearly, there are 24 subgraph isomorphism embeddings in the 4-clique C.

Below, we introduce the concepts of *temporal subgraph embedding* and *stable value*, which are essential to define a stable subgraph embedding.

Definition 3. (Temporal subgraph embedding) *Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a query graph $q = (V_q, E_q)$, for any snapshot $G_i = (V_i, E_i) \in \mathcal{G}$, if an injective function $M : V_q \to V_i$ satisfies $\forall (u_i, u_j) \in E_q, (M(u_i), M(u_j)) \in E_i$, we say that M is a* temporal subgraph embedding *of q. We use $G_i(q)$ to represent the collection of subgraph isomorphism graphs based on all temporal subgraph embeddings in a snapshot G_i, i.e., $G_i(q) = \{g_M | q \simeq g_M \subseteq G_i\}$.*

Example 2. Consider a temporal graph $\mathcal{G}$ in Fig. 1a and a query graph q_1 in Fig. 2a. In the snapshot G_2 of $\mathcal{G}$, the injective mapping $M(u_1 \to v_2, u_2 \to v_3, u_3 \to v_4, u_4 \to v_5)$ is a temporal subgraph embedding. Moreover, we can see that in the snapshot G_3, the mappings: $M_1(u_1 \to v_2, u_2 \to v_3, u_3 \to v_5, u_4 \to v_6)$, $M_2(u_1 \to v_3, u_2 \to v_4, u_3 \to v_5, u_4 \to v_6)$, $M_3(u_1 \to v_7, u_2 \to v_9, u_3 \to v_{10}, u_4 \to v_{11})$ are also temporal subgraph embeddings.

Definition 4. (Stable value) *Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a query graph $q = (V_q, E_q)$ and a temporal subgraph embedding M of q, the* stable value *of M is defined as the number of snapshots that M appears in, i.e., $sv(M) = |\{G_t | q \simeq g_M \subseteq G_t, 1 \le t \le T\}|$.*



(a) q_1 (b) q_2 (c) q_3

Fig. 2. The examples of query graphs.

Stable value can measure the degree of stability of a temporal subgraph embedding. A larger stable value sv represents that the vertices in g_M maintain the connections over sv times, which indicates a long-term stable structure. With the stable value, a stable subgraph embedding is defined as follows.

Definition 5. (Stable subgraph embedding) *Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a query graph $q = (V_q, E_q)$ and an integer θ as stability threshold, a mapping M is a θ-*stable subgraph embedding *when it is a temporal subgraph embedding of q in at least θ snapshots, i.e., $sv(M) \ge \theta$.*

Example 3. Consider a temporal graph $\mathcal{G}$ in Fig. 1 and a query graph q_1 in Fig. 2a. Suppose that the stability threshold θ equals 3. The temporal subgraph embedding $M_1(u_1 \to v_2, u_2 \to v_3, u_3 \to v_4, u_4 \to v_5)$ is only contained in G_2 among all six snapshots of $\mathcal{G}$. Thus we have $sv(M_1) = 1$. While the mapping $M_2(u_1 \to v_7, u_2 \to v_9, u_3 \to v_{10}, u_4 \to v_{11})$ appears in four snapshots, namely, G_3, G_4, G_5 and G_6, thus $sv(M_2)$ equals 4. By $\theta = 3$, we can clearly see that M_2 is a 3-stable subgraph embedding of q but M_1 is not due to $sv(M_1) = 1 < 3$.

Based on the above definitions, we formulate the problem of seeking stable subgraph embeddings in temporal networks as follows.

Problem Formulation. Given a temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a query graph $q = (V_q, E_q)$ and an integer θ, our goal is to find all stable subgraph embeddings of q in $\mathcal{G}$ on the basis of the stability threshold θ.

The following example illustrates the definition of our problem.

Example 4. Reconsider the temporal graph $\mathcal{G}$ in Fig. 1 and a query graph q_1 in Fig. 2a. There are six subgraph isomorphisms of q_1 in all snapshots of $\mathcal{G}$ which are induced by $V_1 = \{v_2, v_3, v_4, v_5\}$, $V_2 = \{v_2, v_3, v_4, v_6\}$, $V_3 = \{v_2, v_3, v_5, v_6\}$, $V_4 = \{v_2, v_4, v_5, v_6\}$, $V_5 = \{v_3, v_4, v_5, v_6\}$ and $V_6 = \{v_7, v_9, v_{10}, v_{11}\}$, respectively. Each subgraph isomorphism can generate 24 temporal subgraph embeddings with the same stable values. The stable values of temporal subgraph embeddings corresponding to the six subgraph isomorphisms are 1, 1, 3, 1, 2 and 4, respectively. Suppose that the stability threshold $\theta = 3$, the answers of stable subgraph embedding search problem are the mappings generated by the subgraphs induced by V_3 and V_6. When θ equals 5, there is no 5-stable subgraph embedding as none of the $6 * 24$ temporal subgraph embeddings satisfying $sv(*) \ge 5$. Consider q_2 in Fig. 2b as a query graph.

Then, there are $7 * 6$ 2-stable subgraph embeddings of q_2 in $\mathcal{G}$ whose stable values are 2. Further, when $\theta > 2$, the result of stable subgraph embedding search is empty.

Clearly, for a query graph q, all subgraph isomorphism embeddings can be easily revealed by a subgraph isomorphism g (i.e., $q \simeq g$). Thus, in the remaining of this paper, we use the term "isomorphism" to refer to "embedding" for simplicity when there is no ambiguity, and we may use embedding, match, and mapping interchangeably.

Remark. Note that our stable subgraph isomorphism search problem is different from the classic frequent subgraph mining problem. A frequent subgraph is a pattern that appears multiple times on a large graph [10], [11], [12], [13], [14] or in a set of graphs [15], [16], [17], [18]. In particular, a temporal motif is a frequent subgraph with the temporal information of edges. A stable subgraph isomorphism is an embedding of a given query graph that appears in multiple snapshots over time. The key difference between the frequent subgraph mining problem and our stable subgraph search problem is that the former does not require a query graph as an input, while the latter is based on a query graph. Due to this key difference, existing solutions for the problem of frequent subgraph mining cannot be directly applied to solve our problem.

Challenges. We first discuss the hardness of the stable subgraph isomorphism search problem. Consider a special case: $\theta = 1$. Clearly, the problem is equivalent to finding subgraph isomorphism in each snapshot of $\mathcal{G}$ which is NP-hard. Thus, finding all isomorphisms of a query graph in at least θ snapshots is also NP-hard.

To solve the stable subgraph isomorphism search problem in temporal graphs, a straightforward solution is to compute the stable value for each temporal subgraph isomorphism in all snapshots and then pick the subgraphs whose stable values are no less than the stability threshold θ as the answers. Such an approach, however, is very costly for large temporal graphs. This is because the solution needs to explore all subgraph isomorphisms in all snapshots of $\mathcal{G}$, which is often intractable for large temporal graphs due to its NP-hard. To improve the efficiency, a potential solution is to apply temporal information of the edges to prune the unpromising intermediate results that definitely cannot obtain a stable subgraph isomorphism. The challenge of the problem is how can we apply the temporal information to speed up the stable subgraph isomorphism search procedure.

In addition, searching subgraph isomorphisms is often not very efficient for large graphs, because it typically needs to perform a backtracking search procedure on the large data graph to identify all subgraph isomorphisms. A natural question is that can we design an index-based solution to efficiently support the stable subgraph isomorphism query? Clearly, we cannot pre-compute all the stable subgraph isomorphisms for all possible small-sized subgraph queries (in practice, the size of the query subgraph is often smaller than 10). Thus, the challenge to answer this question is how can we maintain some stable subgraph isomorphism results to efficiently support all possible subgraph queries.

To tackle the above challenges, we propose a pruning-based search algorithm which can efficiently prune the unpromising intermediate results using the temporal information, as well as an index-based algorithm with a bi-connected component decomposition technique which can efficiently support stable subgraph isomorphism search.

3 A Pruning Search Algorithm

This section proposes a pruning-based stable subgraph isomorphism search algorithm, called PruneSearch, to solve our problem. The PruneSearch extends the classic Ullmann algorithm to handle temporal graphs which also integrates several pruning techniques to prune unpromising candidate matches. Below, we first introduce the pruning rules, followed by the PruneSearch algorithm.

3.1 The Pruning Rules

Let $u_i \in V_q$ be a query vertex and $v_i \in \mathcal{V}$ be a data vertex. $C(u_i)$ denotes the candidate set of u_i which includes the data vertices that may form a mapping $u_i \rightarrow v_i$ in a stable subgraph isomorphism. Let $T(v_i, v_j)$, also known as active time, be the collection of snapshots derived by the temporal edges in $\mathcal{G}$ whose end-vertices are v_i and v_j, i.e., $T(v_i, v_j) = \{t|(v_i, v_j) \in E_t, 1 \leq t \leq T\}$. Denote by $g = (V_g, E_g)$ an arbitrary stable subgraph isomorphism of q in $\mathcal{G}$. Below, we give four observations based on which the search algorithm can prune the intermediate results that definitely cannot form a stable subgraph isomorphism.

Observation 1. *For an edge $(v_i, v_j) \in \mathcal{G}$, if $|\{G_t|(v_i, v_j) \in E_t, 1 \leq t \leq T\}| < \theta$ holds, then (v_i, v_j) is not included in g, i.e., $(v_i, v_j) \notin E_g$.*

Observation 2. Degree reduction*: for a data vertex v_i in $\mathcal{G}$, if $|\{G_t|deg_{G_t}(v_i) \geq deg_q(u_i), 1 \leq t \leq T\}| < \theta$ holds, then $v_i \notin C(u_i)$.*

Observation 3. Failed neighbors reduction*: for a data vertex v_i in $\mathcal{G}$, let $N_{v_i} = N_{v_i}(G)$. We iterative update the sets: $T_{u_i}(v_i) \leftarrow \{t||N_{v_i}(G_t) \cap N_{v_i}| \geq deg_q(u_i), t \in T\}$ and $N_{v_i} \leftarrow \{v_j||\{t|v_j \in N_{v_i}(G_t), t \in T\} \cap T_{u_i}(v_i)| \geq \theta\}$ until N_{v_i} does not changes. If $T_{u_i}(v_i) \geq \theta$ holds, we say that v_i is a candidate of u_i and $T_{u_i}(v_i)$ is the active time of v_i.*

Observation 4. *For a data vertex $v_i \in C(u_i)$, v_i should be removed from $C(u_i)$ if one of the following conditions is satisfied:*

1) Neighbor restriction*: $\exists u_j \in N_{u_i}(q), N_{v_i}(\mathcal{G}) \cap C(u_j) = \emptyset$.*
2) Time restriction*: $\forall v_j \in N_{v_i}(\mathcal{G}) \cap C(u_j), |T_{u_i}(v_i) \cap T_{u_j}(v_j) \cap T(v_i, v_j)| < \theta$.*

3.2 The Proposed Algorithm

Equipped with the above pruning rules, we propose PruneSearch to solve the stable subgraph isomorphism search problem. The main idea of our PruneSearch is to find the results by expanding partial solutions or abandoning them when they definitely cannot form full answers. The algorithm can reduce the computing of unpromising intermediate matches during the backtracking procedure, thus improving the efficiency significantly.

Algorithm 1. PruneSearch

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a query $q = (V_q, E_q)$, an integer θ.
Output: The stable subgraph isomorphism set S.
1 $S \leftarrow \emptyset$; $\hat{G} = (\hat{V}, \hat{E}) \leftarrow \mathcal{G}$;
2 Construct the de-temporal graph $G = (V, E)$ of $\mathcal{G}$;
3 **for** $(v_i, v_j) \in E$ **do**
4 $T(v_i, v_j) = \{t | (v_i, v_j) \in E_t, 1 \le t \le T\}$;
5 $w_{(v_i,v_j)} = |T(v_i, v_j)|$;
6 **if** $w_{(v_i,v_j)} < \theta$ **then** Delete all edges (v_i, v_j, t) from $\hat{G}$;
7 **if** $deg_{\hat{G}}(v_i) = 0$ **then** Delete v_i from $\hat{G}$;
8 **if** $deg_{\hat{G}}(v_j) = 0$ **then** Delete v_j from $\hat{G}$;
9 **for** $u_i \in V_q$ **do**
10 $C(u_i) \leftarrow \emptyset$;
11 **for** $v_i \in \hat{V}$ **do**
12 $T_{u_i}(v_i) \leftarrow \emptyset$; $N_{v_i} \leftarrow N_{v_i}(G)$; $\hat{N}_{v_i} \leftarrow \emptyset$;
13 $flag(v_i) \leftarrow true$;
14 **while** $N_{v_i} \neq \hat{N}_{v_i}$ **do**
15 $\hat{N}_{v_i} \leftarrow N_{v_i}$;
16 $T_{u_i}(v_i) \leftarrow \{t | |N_{v_i}(G_t) \cap N_{v_i}| \ge deg_q(u_i), t \in T\}$;
17 $N_{v_i} \leftarrow \{v_j | |\{t | v_j \in N_{v_i}(G_t), t \in T\} \cap T_{u_i}(v_i)| \ge \theta\}$;
18 **if** $|T_{u_i}(v_i)| \ge \theta$ **then**
19 Insert $(v_i, T_{u_i}(v_i), flag(v_i))$ into $C(u_i)$;
20 **break**;
21 **if** $|C(u_i)| = 0$ **then return** S;
22 $M \leftarrow \emptyset$; $T_c \leftarrow \{t | 1 \le t \le T\}$;
23 SubGraphSearch$(q, \hat{G}, M, T_c)$;
24 **return** S;

The pseudo-code of PruneSearch is shown in Algorithm 1. It first constructs the de-temporal graph G and processes $\mathcal{G}$ by removing the edges that appear in less than θ snapshots as well as isolated vertices (lines 2-8). Such edges and vertices are not contained in a stable subgraph isomorphism with threshold θ according to Observation 1. The algorithm then calculates candidate sets for query vertices in q by applying degree reduction (Observation 2) and failed neighbors reduction (Observation 3); we call this process FirstFilter (lines 9-21). In FirstFilter, PruneSearch determines whether v_i is a candidate of u_i by iteratively updating the sets: $T_{u_i}(v_i)$ and N_{v_i} (lines 14-17). Here a set $\hat{N}_{v_i}$ is used to check whether N_{v_i} was no longer changed. When the loop ends, if $|T_{u_i}(v_i)| \ge \theta$ holds, that means v_i is a candidate of u_i. PruneSearch adds v_i into $C(u_i)$ with a variable $flag(v_i) = true$ and active time $T_{u_i}(v_i)$ (lines 18-20). The variable $flag(v_i)$ is used to indicate whether the candidate v_i is valid. If v_i can be mapped to a query vertex u_i, we set $flag(v_i)$ to true in $C(u_i)$, otherwise it is false. After calculating the initial $C(u_i)$, if it is empty, that means the query vertex u_i cannot be mapped to any data vertex in $\mathcal{G}$, thus the algorithm terminates (line 21). On the other hand, PruneSearch invokes the SubGraphSearch procedure to iteratively map vertices one by one from q to $\hat{G}$ with the candidate sets for seeking the stable subgraph isomorphisms on the basis of θ (line 23). Note that during the backtracking procedure, the search is based on partial matches, thus some candidates in $C(u_i)$ will fail and SubGraphSearch will update the status of *flag* for them (line 18, lines 25-28, line 30 in Algorithm 2). Finally, it returns S as the answers.

The workflow of SubGraphSearch is outlined in Algorithm 2. M is employed to maintain the mapping information $u_i \rightarrow v_i$. The procedure prunes candidates for query vertices by identifying whether they satisfy both neighbor restriction and time restriction (Observation 4); we call this process SecondFilter (lines 3-21). Specifically, for each $u_j \in N_{u_i}(q)$, a variable $disjoint$, initialized to true, is used to indicate that no $v_j \in N_{v_i}(\mathcal{G})$ can be a candidate of u_j. The procedure calculates $temp_t$ by considering both the time T obtained in the last iteration and the active times of v_i, v_j, and (v_i, v_j) (lines 10-11). If $temp_t \ge \theta$ holds, u_j can be mapped to v_j under $u_i \rightarrow v_i$ and M; thus, the SubGraphSearch sets $disjoint$ to false and checks the next neighbor of u_i (line 13). On the other hand, no data vertex is both a neighbor of v_i and a candidate of u_j; thus, v_i is not a candidate of u_i and the procedure updates $C(u_i)$ based on the round of iteration (lines 14-18). During the SecondFilter, once any candidate set $C(u_i)$ is empty, the procedure terminates (line 21). After pruning candidates, the SubGraphSearch picks an unmapped vertex u_s with the smallest size of candidate-set as the selected vertex, and then performs the next iteration for each candidate of u_s (lines 23-30). Before executing the iteration, SubGraphSearch re-computes the active time T_c and updates candidate sets by marking the indicator $flag$ based on the new mapping of u_s (lines 24-28). When the inner SubGraphSearch completes, the procedure needs to recover the candidates' status (line 30). Since SubGraphSearch iteratively maps vertices one by one from q to $\hat{G}$, it adds M into the result set S when $|M| = |V_q|$ holds, thus a θ-stable subgraph isomorphism of q in $\mathcal{G}$ is discovered (line 1).

Algorithm 2. SubGraphSearch $(q, \hat{G}, M, T)$

1 **if** $|M| = |Vq|$ **then** $S \leftarrow S \cup M$;
2 **else**
3 **for**$u_i \in V_q$ **do**
4 $cnt_{u_i} \leftarrow 0$;
5 **for** $(v_i, T_{u_i}(v_i), flag(v_i)) \in C(u_i)$ *and* $flag(v_i) = true$ **do**
6 **for** $u_j \in N_{u_i}(q)$ **do**
7 $disjoint \leftarrow true$;
8 **for** $v_j \in N_{v_i}(\hat{G})$ **do**
9 **if** $(v_j, T_{u_j}(v_j), flag(v_j)) \in C(u_j)$ *and* $flag(v_j) = true$ **then**
10 $temp_t \leftarrow T_{u_i}(v_i) \cap T_{u_j}(v_j) \cap T(v_i, v_j)$;
11 $temp_t \leftarrow temp_t \cap T$;
12 **if** $|temp_t| \ge \theta$ **then**
13 $disjoint \leftarrow false$; **break**;
14 **if** $disjoint = true$ **then**
15 **if** $M = \emptyset$ **then**
16 Delete $(v_i, T_{u_i}(v_i), flag(v_i))$ from $C(u_i)$;
17 **else**
18 $(v_i, T_{u_i}(v_i), flag(v_i)) \leftarrow (v_i, T_{u_i}(v_i), false)$;
19 **for** $(v_i, T_{u_i}(v_i), flag(v_i)) \in C(u_i)$ **do**
20 **if** $flag(v_i) = true$ **then** $cnt_{u_i} \leftarrow cnt_{u_i} + 1$;
21 **if** $cnt_{u_i} = 0$ **then return**;
22 $U = \{u_i | u_i \in M\}$; $u_s = \text{argmin}_{u_i \in (V_q \setminus U)} cnt_{u_i}$;
23 **for** $(v_i, T_{u_i}(v_i), flag(v_i)) \in C(u_s)$ *and* $flag(v_i) = true$ **do**
24 $M.insert(u_s, v_i)$; $T_c = T \cap T_{u_i}(v_i)$;
25 **for** $u_i \in V_q \setminus u_s$ **do**
26 $(v_i, T_{u_i}(v_i), flag(v_i) \leftarrow (v_i, T_{u_i}(v_i), false)$;
27 **for** $v_g \in V \setminus v_i$ **do**
28 $(v_g, T_{u_s}(v_g), flag(v_g) \leftarrow (v_g, T_{u_s}(v_g), false)$;
29 SubGraphSearch$(q, \hat{G}, M, T_c)$;
30 Perform the inverse operation of lines 25-28 for all $C(u_i)$s;

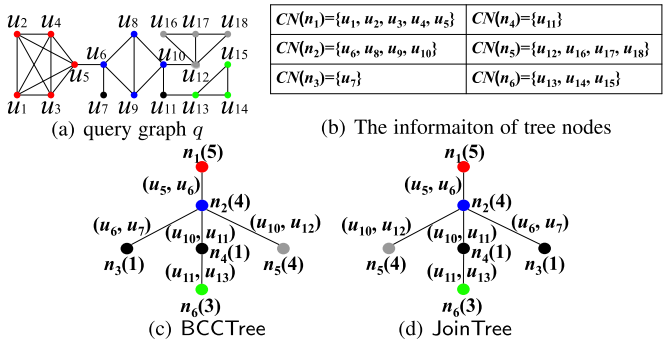


Fig. 3. Illustration of BCCDecompose.

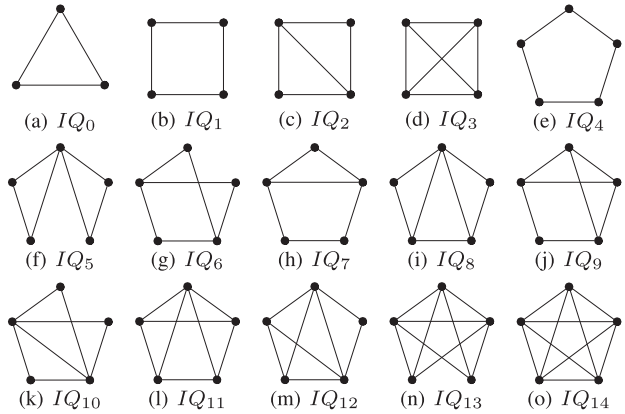


Fig. 4. The index-query graph set G_{IQ}.

Remark. In PruneSearch, we can apply the symmetry-breaking trick to handle the query graphs with automorphism mapping as described in [19], to make a stable subgraph isomorphism search only once. When finding an answer, the other matches with different mapping relationships can be easily revealed by the automorphisms of q.

3.3 The Parallel PruneSearch Algorithm

To further improve the scalability, we develop a parallel version of the pruning-based search algorithm, called PPruneSearch. Specifically, in lines 9-22 of Algorithm 1, calculating the candidates in FirstFilter can be performed independently, thus we can process the query vertices in parallel in this procedure. In addition, in lines 23-30 of Algorithm 2, when SubGraphSearch is first called, i.e., $M = \emptyset$, it picks the first query vertex u_f and generates new mappings with candidates to perform the deeper iterations. For each candidate $v_i \in C(u_f)$, SubGraphSearch finds θ-stable subgraph isomorphisms based on the mapping $u_f \rightarrow v_i$, thus we can process all v_i's in parallel as all of them are independent. We will show that the parallel algorithm PPruneSearch can achieve a very good speedup ratio on real-life graphs in the experiments.

4 The BCCIndex Structure

In this section, we propose an index structure, called BCCIndex, to efficiently support the stable subgraph isomorphism query. Below, we first introduce the BCCIndex structure, followed by the index construction algorithms.

4.1 The Proposed BCCIndex

Before introducing the index structure, we give the definition of *Bi-Connected Component* (BCC) of a graph [20], [21], [22]. Consider a graph G and a subgraph g, we say that g is a BCC if 1) the remaining graph is still connected after removing any 1 edge from g and 2) any super-graph in G of g cannot satisfy 1). Any graph can be decomposed into several BCCs and isolated vertices [20]. We refer to such a decomposition as BCCDecompose. For instance, by performing BCCDecompose on the graph in Fig. 3a, we can obtain four BCCs induced by the vertices colored red, blue, green, and gray, respectively; the vertices u_7 and u_{11} colored black are isolated.

Based on the BCCDecompose, we develop an index structure, called BCCIndex, which maintains all the stable subgraph embeddings for small-size BCCs. In particular, BCCIndex maintains a sorted list of the stable subgraph isomorphisms for all index-query graphs. Here, the index-query graphs are all BCC-graphs with size no larger than 5 as illustrated in Fig. 4. Clearly, there are 15 index-query graphs in total and we denote the collection of them as G_{IQ}. In many practical applications, the size of the query graph is often no larger than 10. By BCCDecompose, the query graph can be decomposed into very small subgraphs. Thus, it is sufficient to store all the isomorphism results of such small subgraphs in G_{IQ} as an index and design an efficient index-based query algorithm to handle different query graphs. In addition, we focus mainly on the stable subgraph isomorphism search problem in temporal graphs. An isomorphism is not considered as a stable isomorphism if it appears in only one snapshot. If we want to find an isomorphism that appears in at least one snapshot, the time constraint will fail; and this problem degenerates to a problem of finding all subgraph isomorphisms in each snapshot. Therefore, we maintain the stable subgraph matches with stable values no less than 2 in our BCCIndex structure.

More specifically, the BCCIndex structure, denoted by EI, contains 15 sorted lists corresponding to the BCC-graphs in Fig. 4. For each graph $IQ_i \in G_{IQ}$, we search stable subgraph isomorphisms based on the stability threshold $\theta = 2$. For a temporal isomorphism m_{IQ_i}, we maintain the snapshots $T(m_{IQ_i})$ that it appears in and calculate $sv(m_{IQ_i})$. Then, a sorted list $EI(IQ_i)$ can be obtained by sorting all temporal isomorphisms in a non-increasing order of their stable values. The goal that we maintain the snapshots is to extend the partial solutions easily in our query processing algorithm to find the complete stable subgraph isomorphisms. Note that if the stable value $sv(m_{IQ_i})$ equals 1, then m_{IQ_i} is not added into $EI(IQ_i)$. The following example illustrates the BCCIndex structure.

Example 5. Consider a temporal graph $\mathcal{G}$ in Fig. 1. The index structure of $\mathcal{G}$ for IQ_0, IQ_3 and IQ_{12} is shown in Fig. 5. Due to space limitations, we only illustrate one instance of the automorphisms for query graphs. Clearly, the BCCIndex structure $EI(IQ_i)$ maintains the stable subgraph isomorphisms whose stable values are no less than 2 and sorts them in a non-increasing order based on the stable values. For instance, the triangle induced by (v_1, v_2, v_4) only appears in G_1, so it is not included in $EI(IQ_0)$. The match containing v_6, v_7, v_8 exists in

IQ_0	snapshots	sv
(v_6, v_7, v_8)	$(G_2, G_3, G_4, G_5, G_6)$	5
(v_7, v_9, v_{10})	(G_3, G_4, G_5, G_6)	4
(v_7, v_9, v_{11})	(G_3, G_4, G_5, G_6)	4
(v_7, v_{10}, v_{11})	(G_3, G_4, G_5, G_6)	4
(v_9, v_{10}, v_{11})	(G_3, G_4, G_5, G_6)	4
(v_2, v_3, v_5)	(G_2, G_3, G_4)	3
(v_2, v_3, v_6)	(G_2, G_3, G_4)	3
(v_2, v_5, v_6)	(G_2, G_3, G_4)	3
(v_3, v_5, v_6)	(G_2, G_3, G_4)	3
(v_4, v_5, v_6)	(G_2, G_3, G_4)	3
(v_7, v_8, v_{10})	(G_4, G_5, G_6)	3
(v_3, v_4, v_5)	(G_2, G_3)	2
(v_4, v_5, v_6)	(G_2, G_3)	2

(a) $EI(IQ_0)$

IQ_3	snapshots	sv
$(v_7, v_9, v_{10}, v_{11})$	(G_3, G_4, G_5, G_6)	4
(v_2, v_3, v_5, v_6)	(G_2, G_3, G_4)	3
(v_4, v_3, v_5, v_6)	(G_2, G_3)	2

(b) $EI(IQ_3)$

IQ_{12}	snapshots	sv
$(v_7, v_{10}, v_8, v_9, v_{11})$	(G_4, G_5, G_6)	3
$(v_5, v_6, v_4, v_2, v_3)$	(G_2, G_3, G_4)	3
$(v_3, v_5, v_4, v_2, v_6)$	(G_2, G_3)	2
$(v_3, v_5, v_2, v_4, v_6)$	(G_2, G_3)	2
$(v_3, v_6, v_4, v_2, v_5)$	(G_2, G_3)	2
$(v_3, v_6, v_2, v_4, v_5)$	(G_2, G_3)	2
$(v_5, v_6, v_2, v_4, v_3)$	(G_2, G_3)	2

(c) $EI(IQ_{12})$

Fig. 5. The BCCIndex structure of $\mathcal{G}$ in Fig. 1.

G_2, G_3, G_4, G_5 and G_6, and its stable value equals 5 which is the largest among all isomorphisms, thus it ranks first in $EI(IQ_0)$. Similarly, we can see that $EI(IQ_3)$ contains three matches induced by $\{v_7, v_9, v_{10}, v_{11}\}$, $\{v_2, v_3, v_5, v_6\}$, and $\{v_4, v_3, v_5, v_6\}$, respectively. It is easy to verify that their stable values are equal to 4, 3, and 2, as illustrated in Fig. 5b. The index $EI(IQ_{12})$ is shown in Fig. 5c which contains seven stable subgraph isomorphisms of IQ_{12} in $\mathcal{G}$.

4.2 The BCCIndex Construction

A Sequential Implementation. We present the BCCIndexBuild algorithm to construct the BCCIndex structure EI. The pseudo-code of BCCIndexBuild is shown in Algorithm 3. For each index-query graph IQ_i in G_{IQ}, the BCCIndexBuild performs PruneSearch to search stable subgraph isomorphisms on the basis of the stability threshold $\theta = 2$. When PruneSearch finds a stable subgraph isomorphism M, it pushes M with the snapshots, that it appears in, into the sorted list $EI(IQ_i)$ and then sorts these matches in $EI(IQ_i)$ in a non-increasing order of their stable values.

A Parallel Implementation. To improve the scalability, we discuss the parallel methods for index construction. Specifically, in lines 2-6 of Algorithm 3, seeking all stable subgraph isomorphisms for each index-graph IQ_i can be performed independently by applying PruneSearch, thus we can process these index-query graphs in parallel. On the other hand, in line 4 of Algorithm 3, for each $IQ_i \in G_{IQ}$, we can perform PPruneSearch (instead of PruneSearch) to calculate stable subgraph isomorphisms for the first query vertex's candidates in parallel.

4.3 Discussions

To the best of our knowledge, the proposed BCCIndex is a novel technique to solve the stable subgraph isomorphism search problem in temporal graphs. Furthermore, we are the first to apply the technique of bi-connected component decomposition, i.e., BCCDecompose, to solve the subgraph isomorphism search problem. We do not use BCCIndex for traditional static subgraph isomorphism search based on the following reasons. First, for the static labeled graphs, the types of labeled bi-connected components grow exponentially with the number of labels, because each vertex can be associated with different labels. It is intractable to compute and store all the subgraph isomorphism results for all labeled bi-connected components. Second, for the unlabeled static graphs, although the number of bi-connected components may not be very large, the corresponding subgraph isomorphism results can be exponentially large which typically cannot be stored as an off-line index. However, for temporal graphs, when adding the temporal constraint, the results of temporal subgraph isomorphism are often not very large which generally can be stored in modern computers as an index (as confirmed in our experiments). Therefore, we can apply the BCCIndex technique to solve the stable subgraph isomorphism search problem on temporal graphs.

Algorithm 3. BCCIndexBuild

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an index-query set G_{IQ}.
Output The BCCIndex EI.
1 $EI \leftarrow \emptyset$;
2 **for** $IQ_i \in G_{IQ}$ **do**
3 $EI(IQ_i) \leftarrow \emptyset$;
4 $EI(IQ_i) \leftarrow$ PruneSearch$(\mathcal{G}, IQ_i, 2)$;
5 Sort all matches in $EI(IQ_i)$ in a non-increasing order based the stable values;
6 $EI \leftarrow EI \cup EI(IQ_i)$;
7 **return** EI;

5 The Index-Based Search Algorithm

We propose an index-based query processing algorithm, called BCCIndexSearch, to search stable subgraph isomorphisms for a query graph q. The main idea is to decompose q into BCCs and isolated vertices, and then join the partial solutions to obtain the results. Below, we first introduce an algorithm, namely BCCMatch, to find the stable subgraph isomorphisms for a BCC based on the BCCIndex, followed by a heuristic join order and the BCCIndexSearch algorithm to solve our problem.

5.1 The BCCMatch Algorithm

After decomposing the query graph q, there may exist some BCCs with sizes larger than 5 which are not contained in our BCCIndex. Hence, we propose an algorithm, called BCCMatch, to handle this case. Specifically, in BCCMatch, all BCCs are categorized into three types as follows. 1) IndexIsoBCC: a BCC that is an isomorphism of any index-query graph in G_{IQ}; 2) StarIsoBCC: a BCC that can be decomposed into an IndexIsoBCC and a star; 3) GeneralBCC: the remaining BCCs that do not satisfy 1) and 2). For instance, the BCC illustrated in Fig. 2a is an isomorphism of IQ_3, thus it is an IndexIsoBCC. In Fig. 2c, the BCC colored blue is not isomorphic to any index-query graph, but we can decompose it into an IndexIsoBCC induced by $\{u_2, u_3, u_4, u_5, u_6\}$ and a star pivoted at u_1, thus it is a StarIsoBCC.

We propose an efficient algorithm to identify whether a BCC is a StarIsoBCC as follows. Given a BCC graph $G_e = (V_e, E_e)$, we sort the vertices in V_e in a non-decreasing order of the degree and then remove vertices following this ordering to decompose G_e. After removing the vertex u and the edges ending with u, we check whether the remaining graph is an IndexIsoBCC. If so, a decomposition strategy is found, and thus G_e is a StarIsoBCC. Otherwise, we continue to remove the next vertex and perform the above procedure to determine a decomposition strategy. When all vertices in

G_e are removed and no strategy is obtained, G_e is recognized as a GeneralBCC.

Algorithm 4. BCCMatch

Input : $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a query $q = (V_q, E_q)$, a BCC $G_e = (V_e, E_e)$, the BCCIndex EI, an integer θ.
Output: The stable subgraph isomorphism set $M(G_e)$ of G_e.

1 $M(G_e) \leftarrow \emptyset$;
2 **if** $\forall IQ_q \in G_{IQ}, \nexists IQ_q \simeq G_e$ **then**
3 Let $\mathcal{Q}$ be a priority queue;
4 $\mathcal{Q} \leftarrow \emptyset$; $isdecom \leftarrow false$;
5 **for** $u_i \in V_e$ **do** $\mathcal{Q}.push(u_i, deg(u_i))$;
6 **while** $\mathcal{Q} \neq \emptyset$ **do**
7 $(u_r, d_{min}) \leftarrow \mathcal{Q}.pop()$; Let $\hat{G} = (\hat{V}, \hat{E})$ be a graph;
8 $\hat{V} \leftarrow V_e \backslash \{u_r\}$; $\hat{E} \leftarrow E_e \backslash \{(u_i, u_j) | u_i = u_r \text{ or } u_j = u_r\}$;
9 **if** $\nexists IQ_{\hat{q}} \in G_{IQ}, IQ_{\hat{q}} \simeq \hat{G}$ **then continue**;
10 **else**
11 $isdecom \leftarrow true$; $C(u_r) \leftarrow \mathcal{V}$;
12 **for** $\hat{m} \in EI(IQ_{\hat{q}})$ **do**
13 **if** $sv(\hat{m}) \geq \theta$ **then**
14 **for** $(u_r, u_i) \in E_e$ **do**
15 $v_i \leftarrow \hat{m}.find(u_i)$;
16 $C(u_r) \leftarrow C(u_r) \cap N_{v_i}(\mathcal{G})$;
17 **for** $v_r \in C(u_r)$ **do**
18 $\hat{T} \leftarrow T$;
19 **for** $(u_r, u_i) \in E_e$ **do**
20 $v_i \leftarrow \hat{m}.find(u_i)$;
21 $\hat{T} \leftarrow T_{u_r}(v_r) \cap T(v_i, v_r) \cap T(\hat{m}_{IQ_{\hat{q}}})$;
22 **if** $\hat{T} \geq \theta$ **then**
23 $m \leftarrow \hat{m}$; $m.insert(u_r, v_r)$;
24 $M(G_e) \leftarrow M(G_e) \cup m$;
25 **else break**;
26 **break**;
27 **if** $isdecom = false$ **then** $M(G_e) \leftarrow$ PruneSearch ($\mathcal{G}$, G_e, θ);
28 **else**
29 **for** $m \in EI(IQ_q)$ **do**
30 **if** $sv(m) \geq \theta$ **then** $M(G_e) \leftarrow M(G_e) \cup m$;
31 **else break**;
32 **return** $M(G_e)$;

The BCCMatch algorithm is depicted in Algorithm 4. Specifically, it first identifies whether G_e is an IndexIsoBCC. If there is an index-query graph IQ_q satisfying $IQ_q \simeq G_e$, BCCMatch outputs the solutions with stable values no less than θ in $EI(IQ_q)$ as the results (lines 29-31). Otherwise, the BCCMatch checks whether G_e is a StarIsoBCC. It pushes the vertices in V_e into a priority queue $\mathcal{Q}$ following a non-decreasing order based on their degrees, and then pops the first element in $\mathcal{Q}$ at each loop to find a decomposition strategy (lines 3-27). A variable *isdecom*, initialized as false, is used to indicate whether G_e is a StarIsoBCC (line 4). When a decomposition strategy is found, i.e., G_e can be decomposed into an IndexIsoBCC $\hat{G}$ and a star pivoted on the removed vertex u_r, BCCMatch sets $isdecom$ to true (line 11). We denote the index-query graph as $IQ_{\hat{q}}$ which is isomorphic to $\hat{G}$. Then, for each match with stable value no less than θ in $EI(IQ_{\hat{q}})$, BCCMatch extends it to obtain the complete solutions of G_e (lines 12-25). If $\mathcal{Q}$ is empty and $isdecom$ still equals false, that means removing any vertex in G_e cannot derive a decomposition, thus we recognize G_e as a GeneralBCC. In this case, BCCMatch performs PruneSearch to search stable isomorphisms for G_e on the basis of the stability threshold θ (line 27). Note that $EI(IQ_q)$ and $EI(IQ_{\hat{q}})$ are sorted lists, BCCMatch terminates when the stable value of a solution is less than θ for handling both IndexIsoBCC (line 25) and StarIsoBCC (line 31). Finally, the set $M(G_e)$ stores the stable subgraph isomorphisms of G_e.

Example 6. Consider a temporal graph $\mathcal{G}$ in Fig. 1 and a query graph q_3 in Fig. 2c. Suppose that $\theta = 2$. We denote the BCC induced by the vertices colored blue as G_e. Obviously, G_e is a StarIsoBCC because after removing u_1, the remaining graph is isomorphic to IQ_{12}. For each solution in $EI(IQ_{12})$ shown in Fig. 5c, we extend it by considering the star induced by the edges $(u_1, u_2), (u_1, u_3)$ to obtain the matches of G_e. We can easily check that only the match induced by $\{v_7, v_{10}, v_8, v_9, v_{11}\}$ can be extended by adding the mapping $u_1 \rightarrow v_6$. With the automorphisms of G_e, the results of G_e are $(v_6, v_7, v_8, v_9, v_{10}, v_{11})$ and $(v_6, v_7, v_8, v_{11}, v_{10}, v_9)$.

Remark. For a query graph q, the BCCMatch algorithm processes its decomposed-BCCs according to their types. When the BCC is a IndexIsoBCC or StarIsoBCC, BCCMatch finds the stable matches with the BCCIndex which maintains stable results for index-query graphs with size no larger than 5. While for the GeneralBCC, the BCCMatch algorithm needs to perform PruneSearch to search stable isomorphisms on the basis of the stability threshold θ.

5.2 A Heuristic Join Order

As aforementioned, a graph G can be decomposed into several BCCs and isolated vertices [20]. There is only one edge to link one BCC/isolated vertex to another BCC/isolated vertex in G. Thus, we can convert G into a tree, called BCCTree, by treating each BCC or isolated vertex as a tree node and adding the edges between them. A tree node n is associated with a set, denoted as $CN(n)$, which represents the corresponding vertices in G. If n is an isolated vertex, we refer to $n = CN(n)$. For brevity, $G_n = (V_n, E_n)$ is used to represent the subgraph induced by $CN(n)$. We connect the tree nodes n_i and n_j if there is an edge between $u_i \in CN(n_i)$ and $u_j \in CN(n_j)$ in G, and the tree edge we label as (u_i, u_j). In this way, the BCCTree of G is created.

Example 7. Consider a graph q in Fig. 3a. Clearly, there are four BCCs and two isolated vertices in q. Fig. 3b illustrates the information of tree nodes. The BCCTree of q is shown in Fig. 3c in which the tree node's color is consistent with the vertices' color in $CN(n)$ in Fig. 3a. We connect n_1 and n_2 with the edge (u_5, u_6) in BCCTree because vertex $u_5 \in CN(n_1)$ and vertex $u_6 \in CN(n_2)$ are linked in q.

Clearly, a join order can be derived by traversing BCCTree from any tree node for our BCCIndexSearch algorithm to merge the partial stable subgraph isomorphisms. However, the pruning performance of BCCIndexSearch with various join orders can be significantly different. Here we design a heuristic join order for BCCIndexSearch by constructing a JoinTree. Let c_i be a child of a tree node n and $dep(c_i)$ be the depth of descendants of c_i in BCCTree. For two children c_1, c_2 of n, we define $c_1 \succ c_2$ if: 1) $|CN(c_1)| > |CN(c_2)|$; 2) $|CN(c_1)| = |CN(c_2)|$ and $dep(c_1) > dep(c_2)$. Following this order, the tree node with a larger size and

deeper descendant has a higher priority to join which is intuitively reasonable. Therefore, we create a root node n_r of the JoinTree with the highest rank based on the order. For each tree node in JoinTree, we then iteratively add its children into JoinTree according to the above order too. Finally, the breadth-first traversal sequence of JoinTree is our join order for merging the partial solutions in BCCIndexSearch. Following such a heuristic join order can form a larger subgraph of q, thus avoiding the invalid merging operation from small substructures.

Example 8. Consider a query graph q in Fig. 3a and its BCCTree in Fig. 3c. Fig. 3d illustrates the JoinTree of q obtained by our method. By performing breadth-first traversal on the JoinTree, we can obtain a join order: $n_1 \rightarrow n_2 \rightarrow n_5 \rightarrow n_4 \rightarrow n_3 \rightarrow n_6$. Following this order, when processing the children of node n_2, that is, when deciding on the next substructure to extend, the BCCIndexSearch first merges n_5 as it has the largest size among all children. Clearly, such a join operation can derive a relatively large partial solution.

5.3 The Query Processing Algorithm

Here we present the query processing algorithm, namely, BCCIndexSearch. The main idea of BCCIndexSearch is to find partial solutions for BCCs of q based on the BCCIndex and then join them to derive the results. The pseudo-code of BCCIndexSearch is outlined in Algorithm 5.

Like PruneSearch, the map structure M in BCCIndexSearch maintains the mapping relationships for each stable subgraph isomorphism. Algorithm 5 works as follows. First, it performs BCCDecompose to calculate BCCs of q and adds BCCs into $\mathcal{V}_B$ and isolated vertices into V_I (lines 1-2). Based on $\mathcal{V}_B$ and V_I, BCCIndexSearch constructs the BCCTree BT and the JoinTree JT to obtain our heuristic join order $\mathcal{Q}$ by breadth-first traversal of JT (lines 3-4). Then, The algorithm pops the head element n_r in $\mathcal{Q}$ (the root of JT) as the initial substructure. If n_r is isolated, it means that the query graph q is a tree, and we search θ-stable subgraph isomorphisms by PruneSearch (line 6). On the other hand, the BCCIndexSearch algorithm performs BCCMatch to find θ-stable isomorphisms for BCCs based on our BCCIndex (lines 8-9). Since n_r is the initial substructure (n_r ranks first), BCCIndexSearch pops the head element n_t in $\mathcal{Q}$ as the next selected substructure and performs the TreeJoin procedure to extend each θ-stable isomorphism of n_r (lines 10-12). Finally, the result set S containing the θ-stable subgraph isomorphism of q is returned.

TreeJoin finds the θ-stable subgraph isomorphisms of q by joining tree nodes based on the order in $\mathcal{Q}$. It extends the current match M by adding the current tree node n_c's solutions that satisfy both neighbor and time restrictions (Observation 4). For a match m_c of tree node n_c, we recognize m_c is not an active candidate of n_c if: 1) multiple query vertices are mapped to the same data vertex; 2) the number of snapshots that contain both m_c and M is less than θ. For each m_c, TreeJoin identifies whether it is active. If no, the procedure terminates. Otherwise, it performs extension depend on the types of n_c: isolated vertex extension (lines 18-24) and BCC extension (lines 25-30). The difference between these two extensions is the selection of candidate match m_c. The former chooses the neighbors of the matched vertex in M, and the latter selects candidates based on the solutions obtained by BCCMatch. When M contains all mappings of query vertices in q, a θ-stable subgraph isomorphism of q is found, thus Algorithm 5 adds it into the result set S (line 15).

Algorithm 5. BCCIndexSearch

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a query $q = (V_q, E_q)$, an integer θ, the BCCIndex EI.
Output: The stable subgraph isomorphism set S.

```
1  S ← ∅; V_B ← ∅; V_I ← ∅;
2  (V_B, V_I) ← BCCDecompose(q);
3  Construct BCCTree BT and JoinTree JT;
4  Q ← the join order by traversing JT with BFS;
5  n_r ← Q.pop(); G_r = (V_r, E_r) ← G_{n_r};
6  if E_r = then S ← PruneSearch (G, q, θ);
7    else
8    for G_ei ∈ V_B do
9      M(G_ei) ← BCCMatch(G, q, G_ei, EI, θ);
10   for m_r ∈ M(G_r) do
11     T ← T(m_r); M ← m_r;
12     n_t ← Q.pop(); TreeJoin(q, G, M, T, n_t);
13 return S;
14 Procedure TreeJoin(q, G, M, T, n_c)
15 if |M| = |Vq| then S ← S ∪ M;
16   else
17     G_c = (V_c, E_c) ← G_{n_c}; V(M) ← {v_i|(u_i, v_i) ∈ M};
18     if V_c = n_c, E_c = ∅ do
19       u_i ← u_i, (u_i, n_c) ∈ E_q;
20       v_i ← M.find(u_i);
21       for v_j ∈ N_{v_i}(G) do
22         if {v_j} ∩ V(M) = ∅ and T ∩ T(v_i, v_j) ≥ θ then
23           T ← T ∩ T(v_i, v_j); M.insert(u_j, v_j);
24           n_t ← Q.pop(); TreeJoin(q, G, M, T, n_t);
25   else
26     for m_c ∈ M(G_c) do
27       V(m_c) ← {v_i|(u_i, v_i) ∈ m_c};
28       if V(m_c) ∩ V(M) = ∅ and T ∩ T(m_c) ≥ θ then
29         T ← T ∩ T(m_c); M ← m_c;
30         n_t ← Q.pop(); TreeJoin(q, G, M, T, n_t);
31 end procedure
```

Remark. In the BCCIndexSearch algorithm, we also use the symmetry-breaking trick to handle the query graphs with automorphism mapping as described in [19], to make a stable subgraph isomorphism search only once.

5.4 The Parallel Query Algorithm

To improve the scalability, we introduce a parallel version of the index-based search algorithm, called PBCCIndexSearch. Specifically, in line 8 of Algorithm 5, the calculation of θ-stable subgraph isomorphisms for BCCs is independent, thus we can process BCCs in parallel. In addition, in lines 10-12 of Algorithm 5, the BCCIndexSearch chooses the n_r with the highest rank as the initial substructure and performs TreeJoin for each θ-stable isomorphism of n_r. Similar to the PPruneSearch, this procedure can also be processed in parallel. Our experiments show that such a simple parallel implementation can achieve a very good speedup ratio compared to the sequential algorithm.

TABLE 1
Datasets

Dataset	n	$\|E\|$	$\|\mathcal{E}\|$	$d_{\max}$	$\|T\|$	Time scale
Chess	7,301	55,899	63,689	263	100	month
Lkml	26,885	159,996	327,077	14,117	97	month
Enron	86,978	297,456	501,116	4,229	60	month
DBLP	1,729,816	8,546,306	12,007,380	5,980	78	year

6 Experiments

In this section, we conduct extensive experiments to evaluate the efficiency and effectiveness of the proposed algorithms. For comparison, we implement an algorithm, called BaselineSearch, as a baseline. BaselineSearch uses a parallel Ullmann algorithm [9] to compute subgraph isomorphisms for a query graph in the de-temporal graph, and then picks the stable isomorphisms among them. We also implement another baseline algorithm, called BaseSnapSearch, which uses the VF2 algorithm [23] to compute subgraph isomorphisms for each snapshot and then selects the stable subgraph embeddings as the results. The BaseSnapSearch also applies the pruning technique derived from Observation 1 before searching subgraph isomorphisms in each snapshot. For our pruning-based stable subgraph isomorphism search algorithm, we implement PruneSearch (Algorithm 1). We implement the BCCIndexBuild algorithm (Algorithm 3) to construct the BCCIndex, as well as the index-based stable subgraph isomorphism search algorithm BCCIndexSearch (Algorithm 5). All algorithms are implemented in C++. In addition, we also implement the parallel versions of PruneSearch, BCCIndexBuild and BCCIndexSearch using OpenMP, namely, PPruneSearch, PBCCIndexBuild and PBCCIndexSearch. All experiments are conducted on a PC with 2.10GHz Intel(R) Xeon(R) Sliver 4110 (8-core) CPU and 256GB memory running Red Hat 4.8.5-16. In all experiments, both the temporal graph and the BCCIndex are stored in the main memory. We set the time limit to 7 days.

Datasets. We use four different types of real-world temporal networks in the experiments. The detailed statistics of the datasets are summarized in Table 1. Chess is a temporal network in which each temporal edge represents two chess players playing a game at time t. Lkml is a temporal communication network of the Linux kernel mailing list. Enron is an email communication network between employees of Enron. DBLP is a temporal collaboration network of authors in dblp from 1940 to 2018. In Table 1, $d_{\max}$ and $|T|$ denote the maximum number of temporal edges associated with a vertex and the number of snapshots respectively. All these datasets are downloaded from http://konect.uni-koblenz.de/.

Parameters. There are two input parameters in our algorithms: the query graph q and the stability threshold θ. The input query graphs used in our experiments are summarized in Fig. 6 which includes eight different types of subgraphs with 5-8 vertices. Since BCCIndex is constructed based on the indexed-query graphs, the BCCIndexSearch algorithm can calculate stable subgraph isomorphisms in linear time if the query graph is one of the indexed-query graphs. Therefore, we only use q_1 as an example where the query graph is an indexed-query graph; for other indexed-query graphs, the results are consistent. For the hard cases, in which the query graph is not an indexed-query graph, we pick seven query graphs (namely, $q_2, q_3, q_4, q_5, q_6, q_7, q_8$ in Fig. 6) to evaluate the proposed algorithms. q_2 is used to evaluate the BCCIndexSearch algorithm for handling StarIsoBCC, and q_7 and q_8 are designed for processing GeneralBCC. For the subgraph isomorphism problem, a slight difference between two query graphs may result in significant query performance changes. Thus, we select q_4, q_5 as the query graphs to exhibit an "edge growing" pattern. Since different datasets have various time scales, the stability threshold θ is set within different time intervals. Specifically, for Chess, θ is selected from the interval $[2, 7]$ with a default value 4. For Lkml and Enron, θ is chosen from the interval $[5, 15]$ with a default value 9. For DBLP, θ is selected from the interval $[5, 10]$ with a default value 7.

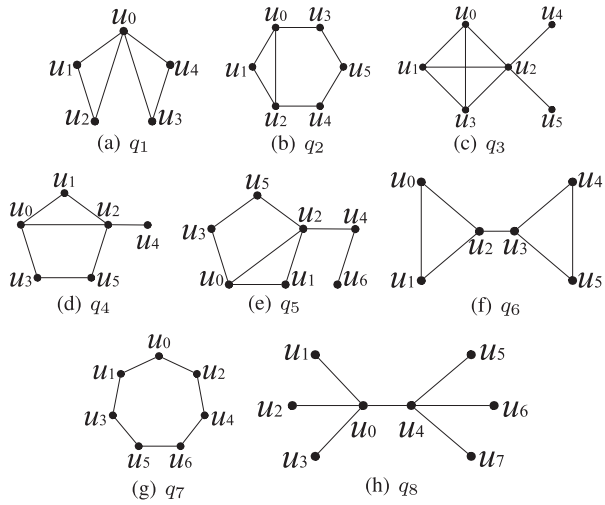


Fig. 6. The query graphs in the experiments.

6.1 Performance Studies

Comparison Among BaselineSearch, BaseSnapSearch *and* PruneSearch. We evaluate BaselineSearch, BaseSnapSearch and PruneSearch with varying parameters on different datasets. Fig. 8 depicts the runtime of BaselineSearch, BaseSnapSearch and PruneSearch with q_1 and q_3 on Chess. The results for other query graphs on Chess are consistent. From Fig. 8, we can see that the runtime of BaselineSearch increases very smoothly with increasing k, while the runtime of BaseSnapSearch and PruneSearch decreases as θ increases for each query graph. This is because BaselineSearch needs to compute all subgraph isomorphisms to select the stable results, while BaseSnapSearch and PruneSearch equipped with pruning techniques can significantly reduce the search space of stable subgraph isomorphisms. Moreover, the running time of PruneSearch is at least 3 orders of magnitude lower than that of BaselineSearch within all parameter settings as expected. Compared with BaseSnapSearch, the PruneSearch is faster for smaller θ (i.e., $\theta = 2, 3$), and for relatively larger θ, the runtime of two algorithms is close. This is because the pruning effect of Observation 1 can significantly reduce the scale of the graph on the basis of a larger θ. For example, for query graph q_3 on Chess, when θ equals 2, PruneSearch takes 0.562 seconds to output all stable subgraph isomorphisms, while BaselineSearch and BaseSnapSearch consume 103,953 seconds and 760 seconds. The runtime of PruneSearch is up to 5 orders of

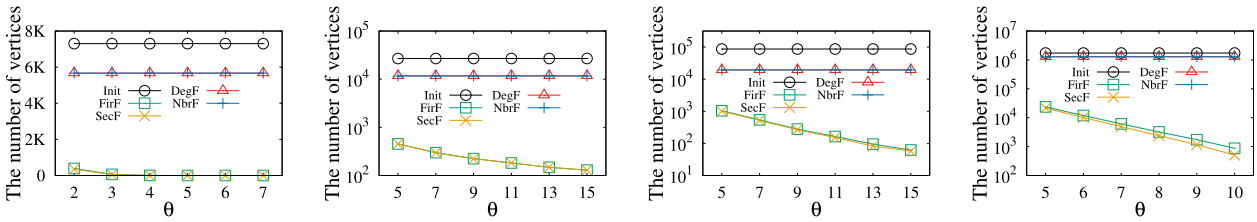


Fig. 7. The pruning effect of FirstFilter and SecondFilter on different datasets.

magnitude and up to 2 orders of magnitude faster than that of BaselineSearch and BaseSnapSearch, respectively. These results confirm that the pruning-based algorithm PruneSearch is substantially faster than the BaselineSearch and BaseSnapSearch algorithms in real-life temporal graphs, which is consistent with our analysis in Section 2.

The Pruning Effect of FirstFilter *and* SecondFilter. In this experiment, we evaluate the effect of FirstFilter and SecondFilter in PruneSearch, and we refer to FirF and SecF for brevity. We also employ two pruning tricks, i.e., degree reduction and neighbor reduction (also known as DegF and NbrF), in BaselineSearch for comparison. We evaluate these pruning techniques before performing the recursive search procedures. For each query vertex, the initial size of candidates is the number of vertices and we denote as Init. Fig. 7 shows the size of candidates for vertex u_2 in q_6 with different pruning techniques on all the datasets. The results with varying θ for other query vertices are consistent using different query graphs. As can be seen from Fig. 7, both FirF and SecF can significantly prune the vertices that are definitely not included in a stable subgraph isomorphism compared with DegF and NbrF. For example, in the case of $\theta = 10$ on DBLP, the number of candidates of u_2 in q_6 after performing FirF is 874 vertices while the size of initial candidates is 1,729,816. SecF can further reduce the size of $C(u_2)$ to 508. For DegF and NbrF, they only reduce the size of candidates to 1,287,416 and 1,285,661 respectively. In addition, the pruning effect of SecF (NbrF) is not obviously superior to that of FirF (DegF). This is because SecF and NbrF work better in the case of extending partial matches, i.e., the recursion procedure of PruneSearch and BaselineSearch. These results suggest that our pruning techniques can significantly prune those vertices that are not included in a stable subgraph isomorphism. Again, these results confirm that PruneSearch is significantly better than BaselineSearch, which are consistent with our previous experiments.

Comparison Among BaseSnapSearch, PruneSearch *and* BCCIndexSearch. Fig. 9 shows the running time of the three algorithms with varying θ for $q_1 \sim q_6$ on different datasets.

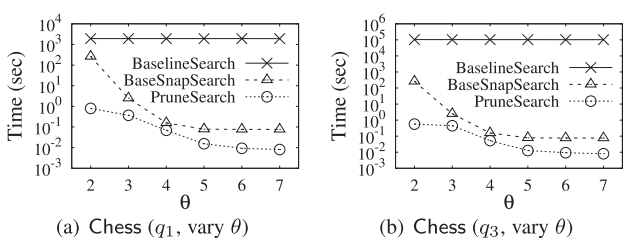


Fig. 8. Running time of BaselineSearch, BaseSnapSearch and PruneSearch on Chess.

As expected, the runtime of BaseSnapSearch, PruneSearch and BCCIndexSearch decreases as θ increases for each query graph. In general, the three algorithms achieve the maximum runtime at the smallest θ. This is because for a smaller stability threshold θ, there are large numbers of temporal subgraph isomorphisms with stable values no less than θ in the graph, thus increasing computational costs. Moreover, we can also see that the proposed PruneSearch and BCCIndexSearch algorithms work well while the BaseSnapSearch algorithm exceeds the time limit within most parameter settings. The running time of PruneSearch is significantly lower than that of BaseSnapSearch for a small θ over all datasets. For a relatively large θ, PruneSearch is also faster than BaseSnapSearch expect for some subgraphs on DBLP (i.e., $\theta = 9$ or 10). This is because the pruning technique derived from Observation 1 can significantly reduce the scale of DBLP for a large θ, which is indeed the case for temporal collaboration networks. The runtime of BCCIndexSearch is at least one order of magnitude and two orders of magnitude lower than that of PruneSearch and BaseSnapSearch within almost all parameter settings, respectively. For example, for query graph q_2 on DBLP, when $\theta = 5$, BCCIndexSearch takes 88 seconds to output all the stable subgraph isomorphisms, while PruneSearch consumes 159,964 seconds and BaseSnapSearch cannot calculate the results within the limited time. The runtime of BCCIndexSearch is at least three orders of magnitude faster than that of PruneSearch and BaseSnapSearch. We also evaluate PruneSearch and BCCIndexSearch with q_7 and q_8. The results on DBLP are depicted in Fig. 10 and similar results can also be found for other datasets. As expected, the runtime of PruneSearch and BCCIndexSearch is relatively close. This is because the q_7 and q_8 are GeneralBCC graphs, and BCCIndexSearch needs to perform PruneSearch to search all stable subgraph isomorphisms. These results demonstrate that the index-based solution BCCIndexSearch is substantially faster than the PruneSearch and BaseSnapSearch algorithm in real-life temporal graphs for query graphs with small-size BCCs.

Evaluation of Parallel Query Processing Algorithms. In this experiment, we evaluate the running time of PPruneSearch and PBCCIndexSearch with varying the number of threads $t \in \{1, 2, 4, 8, 12, 16\}$. Fig. 11 illustrates the results of two query graphs q_3, q_4 on Lkml and Enron. Similar results can also be observed on the other datasets and using the other query graphs. As can be seen from Fig. 11, the runtime of PBC *CIndexSearch* is significantly lower than that of PPruneSearch. For example, for query graph q_4 on Enron, when $t = 16$, PBCCIndexSearch takes 5.9 seconds to output all stable subgraph isomorphisms, while PPruneSearch consumes 3,254.596 seconds. The running time of BCCIndexSearch is at least two orders of magnitude faster than that of PPruneSearch.

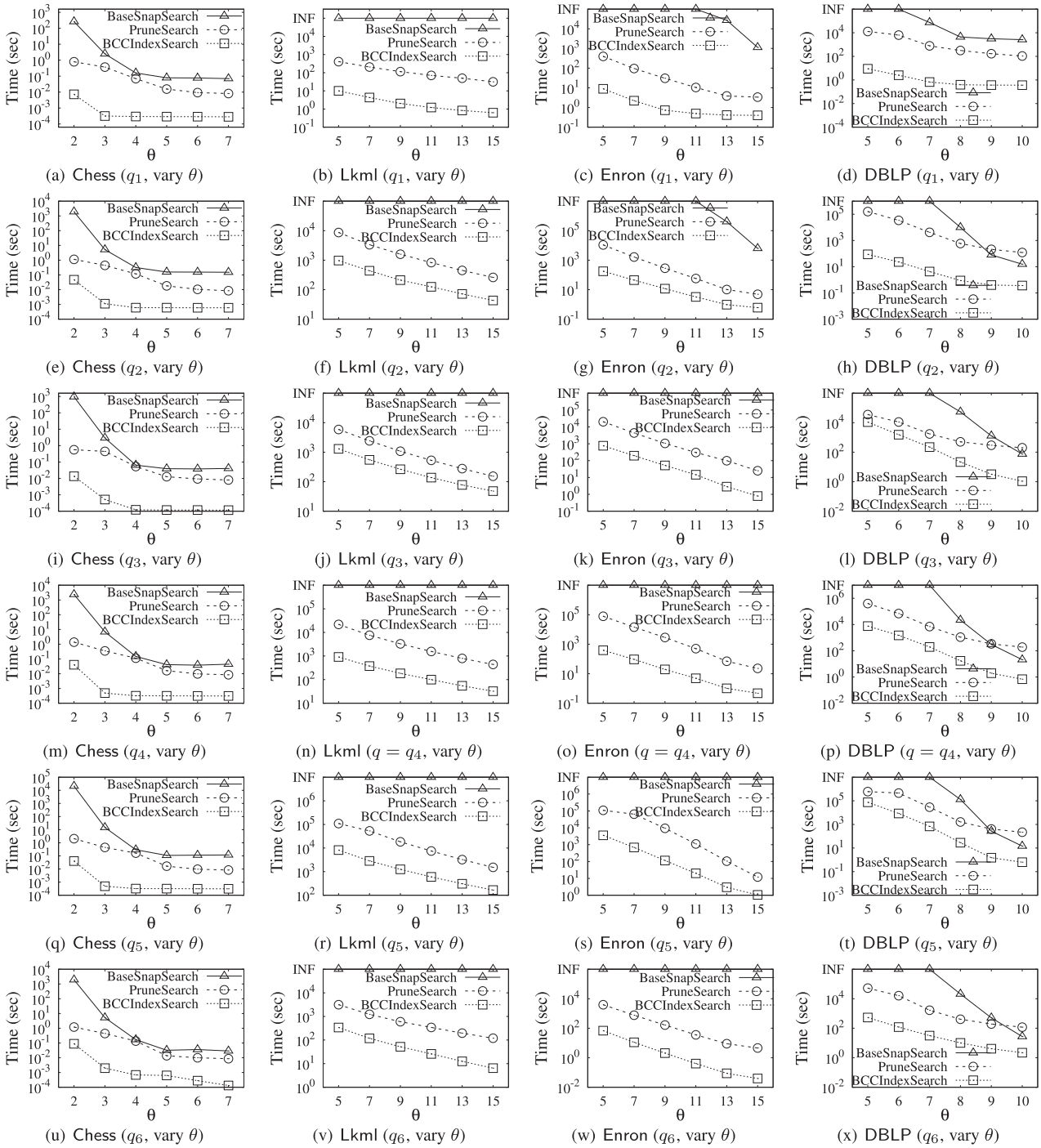
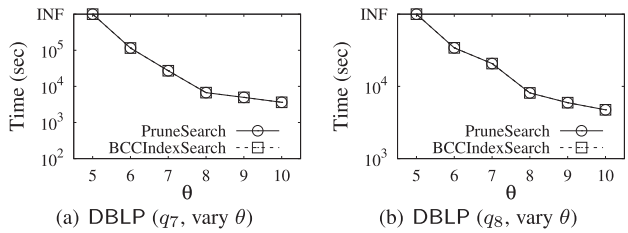


Fig. 9. Running time of PruneSearch and BCCIndexSearch on different datasets.

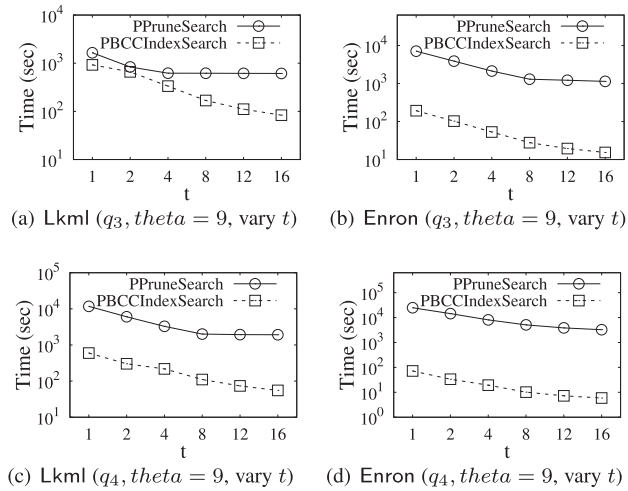
Moreover, we can see that both PPruneSearch and PBCC *IndexSearch* can achieve near-linear speedup ratios over these two datasets. For example, in Fig. 11d, when $t = 16$ and $q = q_4$, the speedup ratios of PPruneSearch and PBCCIndex *Search* on Enron are roughly equal to 7.6 and 12, respectively. These results indicate that our parallel stable subgraph isomorphism search algorithms can achieve very high speedup ratios on real-life graphs.

Evaluation of the BCCIndex. In this experiment, we evaluate the performance of our index construction algorithm. Fig. 12a shows the size of BCCIndex and the graph size. As can be seen, the BCCIndex sizes on all datasets are less than 2.5GB which can be easily stored in the main memory of a modern computer. These results imply that the BCCIndex size is not very large on real-life temporal graphs. In addition, Fig. 12b reports the BCCIndex construction time using the sequential algorithm BCCIndexBuild. In Chess, Lkml, and Enron, our index construction algorithm BCCIndexBuild is very efficient which takes less than 3 hours to construct BCCIndex. In DBLP, BCCIndexBuild is a little bit time-consuming, but it is still able to construct the BCCIndex within 7 days (less than 600,000 seconds). Once the BCCIndex is



(a) DBLP (q_7, vary θ) (b) DBLP (q_8, vary θ)

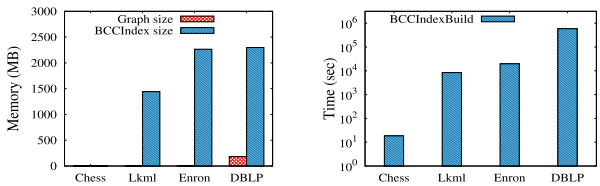
Fig. 10. Running time of PruneSearch and BCCIndexSearch on DBLP.



(a) Lkml ($q_3, theta = 9$, vary t) (b) Enron ($q_3, theta = 9$, vary t)

(c) Lkml ($q_4, theta = 9$, vary t) (d) Enron ($q_4, theta = 9$, vary t)

Fig. 11. Running time of PPruneSearch and PBCCIndexSearch on different datasets.



(a) Index size (b) Construction time

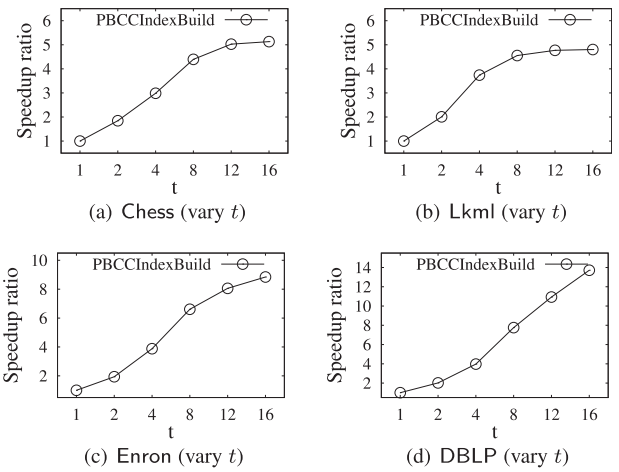
Fig. 12. Evaluation of the BCCIndex.

established, it can be used to handle different query graphs in practical applications. These results suggest that the proposed index-based solution can work on large real-life temporal graphs.

Evaluation of Parallel Index Construction. Here we evaluate the speedup ratio of our parallel index construction algorithm PBCCIndexBuild. To this end, we vary the number of threads t from 1 to 16, and record the runtime of PBCCIndexBuild to compute the speedup ratio for each dataset. Fig. 13 reports the results on all datasets. As expected, PBCCIndexBuild achieves near-linear speedup ratios over all datasets. Moreover, we can observe that in the largest dataset DBLP, the runtime of PBCCIndexBuild is around 14 times lower than the sequential algorithm BCCIndexBuild. These results indicate that our parallel index construction algorithm is very efficient on real-life temporal graphs.

6.2 Case Study

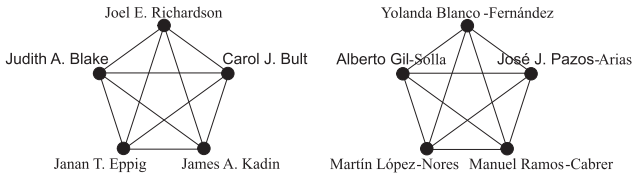
In this experiment, we conduct case studies on DBLP and HCWs, to evaluate the effectiveness of the proposed algorithms. Aforementioned, DBLP is a temporal collaboration network of authors in dblp from 1940 to 2018. HCWs is a



(a) Chess (vary t) (b) Lkml (vary t)

(c) Enron (vary t) (d) DBLP (vary t)

Fig. 13. Speedup ratio of PBCCIndexBuild on various datasets.



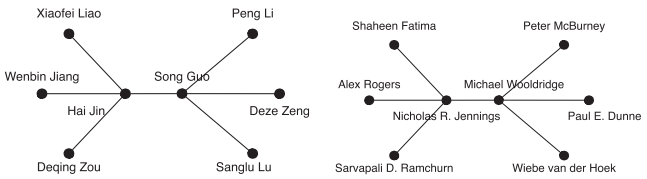
(a) Top-1 ($sv = 18$) (b) Top-2 ($sv = 11$)

Fig. 14. Case study on DBLP with 5-clique query graph.

temporal network of contacts between patients and healthcare workers in a hospital which is downloaded from http://www.sociopatterns.org/datasets.

Case Study on DBLP. In DBLP, we search the stable subgraph isomorphisms of a 5-clique to identify stable teams with the highest stable scores. To this end, we set the stability threshold θ to 2 to find all teams in which the authors have co-authored many papers for more than 2 years, and then sort them in a non-increasing order based on their stable values. The runtime of PruneSearch and BCCIndexSearch is 111,298 seconds and 274 seconds respectively, which is consistent with previous experiments. Fig. 14 shows the top-2 teams with the highest stable values. As shown in Fig. 14a, all the five authors are interested in bioinformatics and work on the Jackson Laboratory. This team contains the long-term collaborators who have co-authored papers from 2000 to 2017, indicating that our algorithm can find stable relationships in real-world applications. In Fig. 14b, we can also see that the authors are the professors of the AtlantTIC research center in Universidade de Vigo. Furthermore, they are the key members of GSSI (Grupo de Servicios Para la Sociedad de la Información) where the coordinator is José J. Pazos-Arias. These authors are interested in semantic web, recommender systems, crowdsourcing and crowd computing; and they have co-authored many papers from 2004 to 2006, 2008 to 2011 and 2013 to 2016. Thus, this team contains long-term collaborators which indicates that our algorithm can indeed find stable communities in real-world applications.

We also search the stable subgraph isomorphisms in DBLP using a "double star" structure as a query graph to reveal stable multi-discipline cooperations. Fig. 15 shows two results as examples. As shown Fig. 15a, Jin and Guo play the roles as "bridge". From their homepages, Jin is mainly interested in parallel and distributed architecture, data query processing,

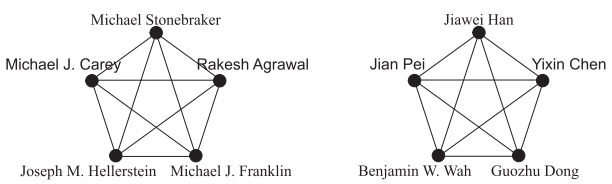


(a) DBLP, $sv = 8$ (b) DBLP, $sv = 8$

Fig. 15. Case study on DBLP with double-star query graph.

computational complexity. Guo's research interests include big data, edge AI, mobile computing, and distributed systems, and others in Fig. 15a are interested in multi-discipline areas as expected. For instance, Liao focuses on memory computing, runtime system, graph computing. Zou's research lies in data security, software vulnerability detection and network attack and defense. Lu and Li mainly focus on wireless networking, cloud computing, pervasive computing, and big data. Zeng is interested in network function visualization, software-defined networking, and edge computing. The members maintain stable multi-discipline cooperative relationships who have co-authored papers from 2009 to 2015, and 2017. Analogously, the authors in Fig. 15b have co-authored papers from 2004 to 2011 whose research covers multiple areas. For example, Jennings and his neighbors are mainly interested in machine learning, autonomous agents and multi-agent systems. Wooldridge's research lies in the intersection of logic, computational complexity and game theory. McBurney focuses on AI and computational finance, including distributed ledgers, blockchain, smart contracts. Van der Hoek is interested in logics for agent systems, data mining and the semantic Web. Dunne's research areas are the complexity of dialogue and argumentation, coalitional games, contract and resource allocation mechanisms. Therefore, this team contains long-term and multiple-areas collaborators. The results indicate that our algorithm can indeed find stable communities with multi-discipline cooperations in real-world applications.

In addition, we extract two temporal subgraphs, namely, DB and DM, from DBLP which contain the authors in DBLP who have published at least one paper in the area of database and data mining, respectively. In both DB and DM, We search the stable subgraph isomorphisms of a 5-clique. Figs. 16a and 16b show the top-1 teams with the highest stable values in DB and DM, respectively. From Fig. 16a, we can observe that the top-1 team is a stable group formed by five famous database researchers. Similarly, as shown in Fig. 16b, the other authors in the top-1 group identified in DM are stable collaborators of Professor Jiawei Han who have co-authored many papers in the past few years. These results further confirm that our algorithm can find stable communities in real-world applications.



(a) Top-1 (DB, $sv = 4$) (b) Top-1 (DM, $sv = 4$)

Fig. 16. Case study on DBLP with 5-clique query graph.

TABLE 2
The Number of Different Types With Varying θ on HCWs

θ	Total	MED	NUR	PAT	ADM
5	520	76	279	47	18
10	162	33	105	16	8
15	84	14	57	8	5
20	54	6	40	4	4
25	45	0	38	3	4
30	33	0	28	2	3

Case Study on HCWs. In HCWs, there are four identity types, i.e., nurse (Nur), medic (Med), administrator (Adm) and patient (Pat), and each individual is associated with an identity label. We search the stable isomorphisms of a triangle with varying stability threshold θ and observe the individual type of these results. Table 2 and Fig. 17 illustrate the number of different types of vertices and the proportion of different types of vertices in all stable triangles, respectively. As can be seen, the number of patients involved in the stable triangles decreases with an increasing θ, while the nurses are the majorities of the stable communities. Moreover, for a large θ, the proportion of medics in stable communities decreases. This is because medics often do not need to maintain such a stable relationship with other people. In addition, the administrator engagement in stable communities is less affected by θ which is consistent with our intuition. These results further confirm that our stable subgraph isomorphism technique can find stable communities and reveal some implicit features of a network.

7 Related Work

Subgraph Isomorphism. Our work is closely related to the subgraph isomorphism on static graphs, where the goal is to find all embeddings of a query graph q in the data graph G. Such a subgraph isomorphism is a classic NP-hard problem [7], [8]. To solve this problem, Ullmann [9] proposed a backtracking algorithm that iteratively maps vertices from q to G by following a fixed order of query vertices. There are many algorithms proposed to improve the efficiency of the classic Ullmann algorithm, including VF2 [23], QuickSI [24], GraphQL [25], SPath [26], TurboISO [27], CFL-Match [28], and DAF [29]. Specifically, VF2 [23] generated the matching order by selecting a vertex connected to one of the already selected vertices rather than a randomly selected vertex. QuickSI [24] created a matching order based on an infrequent-labels first strategy. GraphQL [25] and SPath [26] focused on reducing the candidates of query vertices by exploiting infrequent paths. TurboISO [27] further reduced

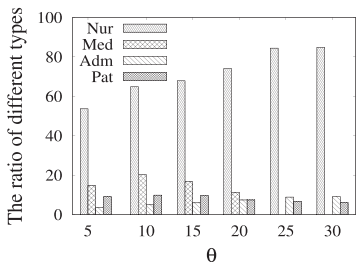


Fig. 17. The ratio of different type with varying θ on HCWs.

the unnecessary cartesian products by employing neighborhood equivalence class to merge similar vertices in a query graph q. Ren *et al.* [30] improved the efficiency of TurboISO based on a technique of compressing the data graph G. CFL-Match [28] made use of the spanning tree instead of the original query graph to postpone cartesian products. Han *et al.* proposed DAF [29] which employs the knowledge learned from past computations to reduce redundant computations. The performance of these subgraph matching algorithms was compared in several previous studies [31], [32]. Most of those improved algorithms mentioned above are tailored for static and labeled graphs. In this work, we investigate a new subgraph isomorphism problem in unlabeled temporal graphs and the algorithms mentioned above cannot be directly used for efficiently solving our problem.

Temporal Graph Analysis. Our work is also related to temporal graph analysis which has attracted much attention in recent years [17], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43]. For example, Bansal *et al.* [33] studied the problem of identifying keyword clusters in large collections of blog posts for specific temporal intervals. Li *et al.* [35] introduced a persistent community model and developed algorithms to efficiently solve this problem. Gurukar *et al.* [17] proposed an algorithm to identify the recurring subgraphs in a temporal graph. Recently, the subgraph isomorphism problem in temporal graphs has been studied. Redmond and Cunningham [39] introduced a time-respecting subgraph isomorphism problem which requires the edges of the query graph following a temporal order. Semertzidis *et al.* [40] studied the problem of mining durable subgraph patterns in a temporal graph. Franzke *et al.* [41] investigated the problem of pattern search in temporal graphs, which focuses on whether the subgraphs exist after Δt time satisfying that the temporal order of edges is consistent with the temporal pattern. Thus, their model only searches isomorphisms within a fixed time interval, which cannot be used to measure the stability. Wang *et al.* [44] introduced a temporal stable community model based on the temporal similarity of edges and developed algorithms by extending the Louvain method to detect stable communities. Since the definition of our problem is different from those of the above problems, all the existing techniques cannot be directly applied for solving our problem. To the best of our knowledge, our work is the first to apply the BCC indexing technique to solve the stable subgraph isomorphism search problem in temporal networks.

8 Conclusion

In this paper, we study the problem of finding stable subgraph isomorphisms in temporal graphs. To solve the problem, we first develop a pruning-based search algorithm based on several non-trivial pruning techniques which can significantly reduce unpromising intermediate results during the search procedure. To further improve the efficiency, we propose a novel index structure, called BCCIndex, to support the stable subgraph isomorphism search in temporal graphs efficiently. We also develop an efficient query processing algorithm based on the BCCIndex and an efficient tree join technique. Finally, we conduct extensive experiments using four real-life datasets to evaluate the efficiency of the proposed algorithms. The results show that the index-based solution BCCIndexSearch is around 1-3 orders of magnitude faster than the PruneSearch algorithm. The results also show that our parallel implementations for both pruning-based search and index-based search algorithms can achieve very high speedup ratios. Finally, we conduct a case study in DBLP and the results demonstrate that our solution for stable subgraph isomorphism search can be useful to identify stable research groups in DBLP.

References

[1] N. Pržulj, D. G. Corneil, and I. Jurisica, "Efficient estimation of graphlet frequency distributions in protein–protein interaction networks," *Bioinformatics*, vol. 22, no. 8, pp. 974–980, 2006.

[2] T. A. Snijders, P. E. Pattison, G. L. Robins, and M. S. Handcock, "New specifications for exponential random graph models," *Sociol. Methodol.*, vol. 36, no. 1, pp. 99–153, 2006.

[3] X. Yan, P. S. Yu, and J. Han, "Graph indexing: A frequent structure-based approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 335–346.

[4] A.-L. Barabasi, "The origin of bursts and heavy tails in human dynamics," *Nature*, vol. 435, no. 7039, pp. 207–211, 2005.

[5] P. Holme and J. Saramäki, "Temporal networks," *Phys. Rep.*, vol. 519, no. 3, pp. 97–125, 2012.

[6] X. Qiu *et al.*, "Real-time constrained cycle detection in large dynamic graphs," *VLDB*, vol. 11, no. 12, pp. 1876–1888, 2018.

[7] J. Hartmanis, "Computers and intractability: A guide to the theory of np-completeness (Michael R. Garey and David S. Johnson)," *Siam Rev.*, vol. 24, no. 1, p. 90, 1982.

[8] H.-N. Tran, J.-J. Kim, and B. He, "Fast subgraph matching on large graphs using graphics processors," in *Proc. Int. Conf. Database Syst. Adv. Appli.*, 2015, pp. 299–315.

[9] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, 1976.

[10] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[11] R. Wan and H. Mamitsuka, "Discovering network motifs in protein interaction networks," in *Biological Data Mining in Protein Interaction Networks*, Hershey, PA, USA: IGI Global, 2009, pp. 117–143.

[12] M. Koyutürk, A. Grama, and W. Szpankowski, "An efficient algorithm for detecting frequent subgraphs in biological networks," *Bioinformatics*, vol. 20, no. suppl_1, pp. i200–i207, 2004.

[13] C. Jiang, F. Coenen, and M. Zito, "A survey of frequent subgraph mining algorithms," *Knowl. Eng. Rev.*, vol. 28, no. 1, pp. 75–105, 2013.

[14] P. Zhao and J. X. Yu, "Fast frequent free tree mining in graph databases," *World Wide Web*, vol. 11, no. 1, pp. 71–92, 2008.

[15] R. Jin, S. McCallen, and E. Almaas, "Trend motif: A graph mining approach for analysis of dynamic complex networks," in *Proc. 17th IEEE Int. Conf. Data Mining*, 2007, pp. 541–546.

[16] P. Gupta *et al.*, "Real-time twitter recommendation: Online motif detection in large dynamic graphs," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1379–1380, 2014.

[17] S. Gurukar, S. Ranu, and B. Ravindran, "COMMIT: A scalable approach to mining communication motifs from dynamic networks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 475–489.

[18] R. Ahmed and G. Karypis, "Algorithms for mining the coevolving relational motifs in dynamic networks," *ACM Trans. Knowl. Discov. Data*, vol. 10, no. 1, pp. 4:1–4:31, 2015.

[19] J. A. Grochow and M. Kellis, "Network motif discovery using subgraph enumeration and symmetry-breaking," in *Proc. Annu. Int. Conf. Res. Comput. Mol. Biol.*, 2007, pp. 92–106.

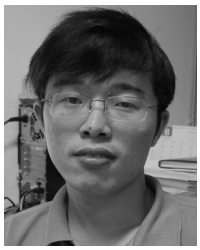
[20] A. Gibbons, *Algorithmic Graph Theory*, Cambridge, U.K.: Cambridge Univ. Press, 1985.

[21] T. Akiba, Y. Iwata, and Y. Yoshida, "Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction," in *Proc. 22nd ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 909–918.

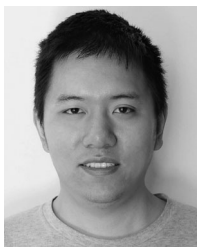
[22] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 205–216.

[23] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, Sep. 2004.
[24] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: An efficient algorithm for testing subgraph isomorphism," *PVLDB*, vol. 1, no. 1, pp. 364–375, 2008.
[25] H. He and A. K. Singh, "Graphs-at-a-time: Query language and access methods for graph databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 405–418.
[26] P. Zhao and J. Han, "On graph query optimization in large networks," *PVLDB*, vol. 3, no. 1/2, pp. 340–351, 2010.
[27] W.-S. Han, J. Lee, and J.-H. Lee, "Turboiso: Towards ultrafast and robust subgraph isomorphism search in large graph databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 337–348.
[28] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing cartesian products," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 1199–1214.
[29] M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han, "Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2019, pp. 1429–1446.
[30] X. Ren and J. Wang, "Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs," *PVLDB*, vol. 8, no. 5, pp. 617–628, 2015.
[31] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee, "An in-depth comparison of subgraph isomorphism algorithms in graph databases," *VLDB*, vol. 6, no. 2, pp. 133–144, 2012.
[32] S. Sun and Q. Luo, "In-memory subgraph matching: An in-depth study," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1083–1098.
[33] N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa, "Seeking stable clusters in the blogosphere," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 806–817.
[34] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *PVLDB*, vol. 7, no. 9, pp. 721–732, 2014.
[35] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *Proc. IEEE 34th Int. Conf. Data Eng.*, 2018, pp. 797–808.
[36] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. Lui, "Diversified temporal subgraph pattern mining," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1965–1974.
[37] Z. Yang, A. W.-C. Fu, and R. Liu, "Diversified top-k subgraph querying in a large graph," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 1167–1182.
[38] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai, "Fast computation of dense temporal subgraphs," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 361–372.
[39] U. Redmond and P. Cunningham, "Subgraph isomorphism in temporal networks," 2016, *arXiv:1605.02174*.
[40] K. Semertzidis and E. Pitoura, "Durable graph pattern queries on historical graphs," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2016, pp. 541–552.
[41] M. Franzke, T. Emrich, A. Züfle, and M. Renz, "Pattern search in temporal social networks," in *Proc. 21st Int. Conf. Extending Database Technol.*, 2018, pp. 289–300.
[42] H. Qin, R.-H. Li, G. Wang, L. Qin, Y. Yuan, and Z. Zhang, "Mining bursting communities in temporal graphs," 2019, *arXiv: 1911.02780*.
[43] H. Qin, R. Li, G. Wang, L. Qin, Y. Cheng, and Y. Yuan, "Mining periodic cliques in temporal networks," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2019, pp. 1130–1141.
[44] W. Wang and X. Li, "Temporal stable community in time-varying networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 3, pp. 1508–1520, Mar. 2019.

Qi Zhang is currently working toward the PhD degree with the Beijing Institute of Technology, China. Her current research interests include social network analysis and data-driven graph mining.



Rong-Hua Li received the PhD degree from the Chinese University of Hong Kong, in 2013. He is currently a professor with the Beijing Institute of Technology, Beijing, China. His research interests include graph data management and mining, social network analysis, graph computation systems, and graph-based machine learning.



Hongchao Qin received the BS degree in mathematics and the ME and PhD degrees in computer science from Northeastern University, China, in 2013, 2015 and 2020, respectively. He is currently a postdoc with the Beijing Institute of Technology, China. His current research interests include social network analysis and data-driven graph mining.

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

Zhiwei Zhang received the BS degree from the Renmin University of China, in 2010, and the PhD degree from the Chinese University of Hong Kong, in 2014. He is currently a professor with the Beijing Institute of Technology (BIT), Beijing, China. His research interests include federal learning, data pricing and transaction, distributed system, blockchain, and algorithm analysis.

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