Finding Theme Communities from Database Networks: from Mining to Indexing and Query Answering

Lingyang Chu  
Simon Fraser University  
Vancouver, Canada  
lca117@sfu.ca

Zhefeng Wang  
Univ. of Science and Tech. of China  
Hefei, China  
zhefwang@mail.ustc.edu.cn

Jian Pei  
Simon Fraser University  
Vancouver, Canada  
jpei@cs.sfu.ca

Yanyan Zhang  
Simon Fraser University  
Vancouver, Canada  
yanyanz@sfu.ca

Yu Yang  
Simon Fraser University  
Vancouver, Canada  
yya119@sfu.ca

Enhong Chen  
Univ. of Science and Tech. of China  
Hefei, China  
cheneh@ustc.edu.cn

ABSTRACT

Given a database network where each vertex is associated with a transaction database, we are interested in finding theme communities. Here, a theme community is a cohesive subgraph such that a common pattern is frequent in all transaction databases associated with the vertices in the subgraph. Finding all theme communities from a database network enjoys many novel applications. However, it is challenging since even counting the number of all theme communities in a database network is #P-hard. Inspired by the observation that a theme community shrinks when the length of the pattern increases, we investigate several properties of theme communities and develop TC FI, a scalable algorithm that uses these properties to effectively prune the patterns that cannot form any theme community. We also design TC-Tree, a scalable algorithm that decomposes and indexes theme communities efficiently. Retrieving 1 million theme communities from a TC-Tree takes only 1 second. Extensive experiments and a case study demonstrate the effectiveness and scalability of TCFI and TC-Tree in discovering and querying meaningful theme communities from large database networks.

ACM Reference format:
Lingyang Chu, Zhefeng Wang, Jian Pei, Yanyan Zhang, Yu Yang, and Enhong Chen. 2017. Finding Theme Communities from Database Networks: from Mining to Indexing and Query Answering. In Proceedings of ACM, Canada, 2017 (arXiv version), 14 pages.
https://doi.org/10.1145/nnnnnn.nnnnnn

1 INTRODUCTION

Recently, Facebook developed the “shop section” that encourages users to buy and sell products on their home pages. Amazon launched the “social media influencer program” that allows influential users to promote products to their followers. These and many other incoming innovative business initiatives alike integrate social networks and e-commerce, and introduce the new social e-commerce networks, which contain rich information of the social interactions and e-commercial transactions and activities of users. Essentially, in such a network, every vertex is associated with a transaction database, which records the user’s activities. In this paper, we call such networks database networks.

Can we find interesting patterns in social e-commerce networks and gain valuable business insights? There are two critical and complementary aspects of information: the network structures and the transactions associated with vertices. Consequently, it is interesting to find overlapping social groups (network structures) such that people in the same group share the same dominant e-commercial activity patterns. For example, we may find a group of people who frequently buy diapers with beer together. The finding of social groups of people associated with the same dominant e-commercial activity patterns provides valuable knowledge about the strong buying habits of social communities, and is very useful in personalized advertising and business marketing. We call this data mining task finding theme communities from database networks.

Finding theme communities is also meaningful in other database networks beyond social e-commerce networks. For example, as reported in our empirical study, we can model location-based social networks and the check-in information associated as database networks, where each user is a vertex and the set of locations that the user checks in within a period (e.g., a day) as a transaction. A theme community in this database network represent a group of friends who frequently visit the same set of locations. Moreover, in a co-author network, authors are vertices and two authors are linked if they co-authored before. The network can be enhanced by associating each author with a transaction database where each transaction is the set of keywords in an article published by the author. Then, a theme community in this database network is a group of collaborating authors who share the same research interest.

There are established community detection methods for vertex attributed networks and simple networks without vertex attributes, as to be reviewed in Section 2. Can we adapt existing methods to tackle the problem of finding theme communities? Unfortunately, the answer is no due to the following challenges.

First, the vertex databases in database network create a huge challenge for the existing methods that work well in vertex attributed networks. To the best of our knowledge, existing methods for vertex attributed networks only consider the case where each
vertex is associated with a single set of items. Those methods cannot distinguish the different frequencies of patterns in millions of vertex databases. Besides, we cannot simply transform each vertex database into a single set of items by taking the union of all transactions in the vertex database. Because this wastes the valuable information of item co-occurrence and pattern frequency.

Second, every vertex database in a database network contains an exponential number of patterns. Since the community detection methods that work in simple networks only detect the theme communities of one pattern at a time, we have to perform community detection for each of the exponential number of patterns, which is computationally intractable.

Last but not least, a large database network usually contains a huge number of arbitrarily overlapping theme communities. Efficiently detecting overlapping theme communities and indexing them for fast query-answering are both challenging problems.

In this paper, we tackle the novel problem of finding theme communities from database networks and make the following contributions.

First, we introduce database network as a network of vertices associated with transaction databases. A database network contains rich information about item co-occurrence, pattern frequency and graph structure. This presents novel opportunities to find theme communities with meaningful themes that consist of frequently co-occurring items.

Second, we motivate the novel problem of finding theme communities from database network and prove that even counting the number of theme communities in a database network is #P-hard.

Third, we design a greedy algorithm to find theme communities by detecting maximal pattern trusses for every pattern in all vertex databases. To improve the efficiency in practice, we first investigate several useful properties of maximal pattern trusses, then apply these properties to design two effective pruning methods that reduce the time cost by more than two orders of magnitudes in our experiments without losing the community detection accuracy.

Fourth, we advocate the construction of a data warehouse of maximal pattern truss. To facilitate indexing and query answering of the data warehouse, we show that a maximal pattern truss can be efficiently decomposed and stored in a linked list. Using such decomposition, we design an efficient theme community indexing tree. We also develop an efficient query answering method that takes only 1 second to retrieve 1 million theme communities from the indexing tree.

Last, we report extensive experiments on both real and synthetic datasets and demonstrate the efficiency of the proposed theme community finding method and indexing method. The case study in a large database network shows that finding theme communities discovers meaningful groups of collaborating scholars who share the same research interest.

The rest of the paper is organized as follows. We review related work in Section 2 and formulate the theme community finding problem in Section 3. We present a baseline method and a maximal pattern truss detection method in Section 4. We develop our major theme community finding algorithms in Section 5, and the indexing querying answering algorithms in Section 5. We report a systematic empirical study in Section 7 and conclude the paper in Section 8.

2 RELATED WORKS

To the best of our knowledge, finding theme communities from database networks is a new problem that has not been formulated or tackled in literature. Broadly it is related to truss detection and vertex attributed network clustering.

2.1 Truss Detection

Truss detection aims to detect $k$-truss from unweighted networks. Cohen [8] defined $k$-truss as a subgraph $S$ where each edge in $S$ is contained by at least $k - 2$ triangles in $S$. He also proposed a polynomial time algorithm for efficient $k$-truss detection [8]. As demonstrated by many studies [7–9], $k$-truss naturally models cohesive communities in social networks and are elegantly related to some other graph structures, such as $k$-core [24] and $k$-clique [19].

The elegance of $k$-truss attracts much attention in research. Huang et al. [16] designed an online community search method to find $k$-truss communities that contain a query vertex. They also proposed a memory efficient index structure to support fast $k$-truss search and online index update. Wang et al. [26] focused on solving the truss decomposition problem, which is to find all non-empty $k$-trusses for all possible values of $k$ in large unweighted networks. They first improved the in-memory algorithm proposed by Cohen [8], then proposed two efficient methods to deal with large networks that cannot be held in memory: Huang et al. [28] proposed a new structure named $(k, \gamma)$-truss that further extends the concept of $k$-truss from deterministic networks to probabilistic networks. They also proposed several algorithmic tools for detection, decomposition and approximation of $(k, \gamma)$-trusses.

All the methods mentioned above perform well in detecting communities from networks without vertex databases. However, since the database network contains an exponential number of patterns and we don’t know which pattern forms a theme community, enumerating all theme communities in the database network requires to perform community detection for each of the exponential number of patterns, which is computationally intractable.

2.2 Vertex Attributed Network Clustering

A vertex attributed network is a network where each vertex is associated with a set of items. Vertex attributed network clustering aims to find cohesive communities such that all vertices within the same community contain the same set of items.

Various methods were proposed to solve this problem. Among them, the frequent pattern mining based methods have been proved to be effective. Berlingerio et al. [5] proposed ABACUS to find multi-dimensional communities by mining frequent closed itemsets from multi-dimensional community memberships. Moser et al. [21] devised CoPaM to efficiently find all maximal cohesive communities by exploiting various pruning strategies. Prado et al. [22] designed several interestingness measures and the corresponding mining algorithms for cohesive communities. Moosavi et al. [20] applied frequent pattern mining to find cohesive groups of users who share similar features. There are also effective methods that are based on graph weighting [11, 25], structural embedding [10, 12], random walk [30], statistical inference [4, 29], matrix compression [2], subspace clustering [13, 27], and $(k, d)$-truss detection [17].
3 PROBLEM DEFINITION

In this section, we first introduce the notions of database network, theme network and theme community, and then formalize the theme community finding problem.

3.1 Database Network and Theme Network

Let \( S = \{s_1, \ldots, s_m\} \) be a set of items. An itemset \( x \) is a subset of \( S \), that is, \( x \subseteq S \). A transaction \( t \) is an itemset. Transaction \( t \) is said to contain itemset \( x \) if \( x \subseteq t \). The length of transaction \( t \), denoted by \( |t| \), is the number of items in \( t \). A transaction database \( d = \{t_1, \ldots, t_h\} \) (\( h \geq 1 \)) is a multi-set of transactions, that is, an itemset may appear multiple times as transactions in a transaction database.

A database network is an undirected graph \( G = (V, E, D, S) \), where each vertex is associated with a transaction database, that is,

- \( V = \{v_1, \ldots, v_n\} \) is a set of vertices;
- \( E = \{(v_i, v_j) \mid v_i, v_j \in V\} \) is a set of edges;
- \( D = \{d_1, \ldots, d_k\} \) is a set of transaction databases, where \( d_i \) is the transaction database associated with vertex \( v_i \); and
- \( S = \{s_1, \ldots, s_m\} \) is the set of items that constitute all transaction databases in \( D \). That is, \( \cup_{i \in D} v_i \cup_{j \in D} t = S \).

Figure 1(a) gives a toy database network of 9 vertices, where each vertex is associated with a transaction database, whose details are omitted due to the limit of space.

A theme is an itemset \( p \subseteq S \), which is also known as a pattern in the field of frequent pattern mining [1, 15]. The length of \( p \), denoted by \( |p| \), is the number of items in \( p \). The frequency of \( p \) in transaction database \( d_i \), denoted by \( f_i(p) \), is the proportion of transactions in \( d_i \) that contain \( p \) [1, 15]. \( f_i(p) \) is also called the frequency of \( p \) on vertex \( v_i \). In the rest of this paper, we use the terms theme and pattern interchangeably.

Given a pattern \( p \), the theme network \( G_p \) is a subgraph induced from \( G \) by the set of vertices satisfying \( f_i(p) > 0 \). We denote it by \( G_p = (V_p, E_p) \), where \( V_p = \{v_i \in V \mid f_i(p) > 0\} \) is the set of vertices and \( E_p = \{(v_i, v_j) \mid v_i, v_j \in V_p \} \) is the set of edges.

Figures 1(b) and 1(c) present two theme networks induced by two different patterns \( p \) and \( q \), respectively. The edges and vertices marked in dashed lines are not included in the theme networks.

All these methods mentioned above cannot exploit the rich and useful information of item co-occurrences and pattern frequencies in a database network.

We can induce a theme network by each pattern \( p \subseteq S \), a database network \( G \) can induce at most \( 2^{|S|} \) theme networks, where \( G \) can be regarded as the theme network of \( p = \emptyset \).

3.2 Theme Community

A theme community is a subgraph of a theme network so that the vertices form a cohesively connected subgraph. Intuitively, in a good theme community with theme \( p \), every vertex is expected to satisfy at least one of the following criteria:

(1) It has a high frequency of \( p \) in its vertex database.
(2) It is connected to a large proportion of vertices in the theme community.

The rationale is that, a vertex with a high frequency of \( p \) is a theme leader that strengthens the theme coherence of the theme community. A vertex connecting to many vertices in the community is a social leader that strengthens the edge connection in the theme community. Both theme leaders and social leaders are important members of the theme community. Take Figure 1(b) as an example, \( v_1, v_2, v_3, v_4, v_5 \) form a theme community with theme \( p \), where \( v_1 \) and \( v_2 \) are theme leaders, \( v_3 \) and \( v_4 \) are social leaders, and \( v_5 \) is both a theme leader and a social leader.

The above criteria inspire us to measure the cohesion of edge \( e_{ij} = (v_i, v_j) \) by considering the number of common neighbour vertices between \( v_i \) and \( v_j \), as well as the frequencies of \( p \) on \( v_i, v_j \) and their common neighbour vertices. The rationale is that, in a good theme community with theme \( p \), two connected vertices \( v_i \) and \( v_j \) should have a large edge cohesion, if they each has a high frequency of \( p \), or have many common neighbours in the theme community.

Let \( v_k \) be a common neighbor of two connected vertices \( v_i \) and \( v_j \). Then, \( v_i, v_j \) and \( v_k \) form a triangle, denoted by \( \triangle_{ijk} = \{v_i, v_j, v_k\} \). Since every common neighbor of \( v_i \) and \( v_j \) corresponds to a unique triangle that contains edge \( e_{ij} = (v_i, v_j) \), the number of common neighbors of \( v_i \) and \( v_j \) is exactly the number of triangles that contain \( e_{ij} = (v_i, v_j) \).

Now we define edge cohesion, which measures the cohesion between two connected vertices \( v_i, v_j \) by considering both the number of triangles containing \( e_{ij} = (v_i, v_j) \) and the frequencies of \( p \) on the vertices of those triangles.

**Definition 3.1 (Edge Cohesion).** Consider a theme network \( G_p \) and a subgraph \( G_p \subseteq G_p \), for an edge \( e_{ij} = (v_i, v_j) \) in \( G_p \), the edge
Table 1: Frequently used notations.

| Notation | Description |
|----------|-------------|
| $G$      | The database network. |
| $G_p$    | The theme network induced by pattern $p$. |
| $S$      | The complete set of items in $G$. |
| $|\cdot|$  | The set volume operator. |
| $C_p^\alpha(\cdot)$ | The maximal pattern truss in theme network $G_p$ with respect to threshold $\alpha$. |
| $L_p$    | The linked list storing the decomposed results of maximal pattern truss $C_p^\alpha(0)$. |
| $f_i(p)$ | The frequency of pattern $p$ on vertex $v_i$. |
| $e$      | The threshold of pattern frequency for TCS. |

cohesion of $e_{ij}$ in $G_p$ is

$$eco_{ij}(G_p) = \sum_{\Delta_{ij,k} \subseteq C_p} \min(f_1(p), f_2(p), f_3(p))$$

Example 3.2. In Figure 1(b), for subgraph $C_p$ induced by the set of vertices $\{v_1, v_2, v_3, v_4, v_5\}$, edge $e_{12}$ is contained by $\Delta_{123}$ and $\Delta_{125}$, thus the edge cohesion of $e_{12}$ is $eco_{12}(C_p) = \min(f_1(p), f_2(p)) = 0.2$.

In a subgraph $C_p$, if for every vertex $v_i$ in the subgraph, $f_i(p) = 1$, then the edge cohesion $eco_{ij}(C_p)$ equals the number of triangles containing edge $e_{ij}$. In this case, $eco_{ij}(C_p)$ is exactly the edge cohesion used by Cohen [8] to define $k$-truss.

Now, we propose pattern truss, a subgraph such that the cohesion of every edge in the subgraph is larger than a threshold.

Definition 3.3 (Pattern Truss). Given a minimum cohesion threshold $\alpha \geq 0$, a pattern truss $C_p^\alpha = (V_p(\alpha), E_p(\alpha))$ is an edge-induced subgraph of $G_p$ on the set of edges $E_p(\alpha) = \{e_{ij} \mid eco_{ij}(C_p) > \alpha\}$.

If $\alpha = k - 3$ and $\forall v_i \in V_p(\alpha), f_i(p) = 1$, a pattern truss $C_p^\alpha$ becomes a $k$-truss [8], which is a subgraph where every edge in the subgraph is contained by at least $(k - 2)$ triangles. Furthermore, if $C_p(\alpha)$ is also a maximal connected subgraph in $G_p$, it will also be a $(k - 1)$-core [24].

Similar to $k$-truss, a pattern truss is not necessarily a connected subgraph. For example, in Figure 1(b), when $\alpha \in [0, 0.2)$, the subgraph marked in bold lines is a pattern truss, but is not connected.

It is easy to see that, for a given $\alpha$, the union of multiple pattern trusses is still a pattern truss.

Definition 3.4 (Maximal Pattern Truss). A maximal pattern truss in $G_p$ with respect to a minimum cohesion threshold $\alpha$ is a pattern truss that any proper superset is not a pattern truss with respect to $\alpha$ in $G_p$.

Apparentlly, a maximal pattern truss in $G_p$ with respect to $\alpha$ is the union of all pattern trusses in $G_p$ with respect to the same threshold $\alpha$. Moreover, a maximal pattern truss is not necessarily a connected subgraph. We denote by $C_p^\alpha(\cdot) = (V_p^\alpha(\cdot), E_p^\alpha(\cdot))$ the maximal pattern truss in $G_p$.

Now we are ready to define theme community.

Definition 3.5 (Theme Community). Given a minimum cohesion threshold $\alpha$, a theme community is a maximal connected subgraph in the maximal pattern truss with respect to $\alpha$ in a theme network.

Example 3.6. In Figure 1(b), when $\alpha \in [0, 0.2)$, $\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_7, v_8, v_9\}$ are two theme communities in $G_p$. In Figure 1(c), when $\alpha \in [0.2, 0.4)$, $\{v_2, v_3, v_5, v_6, v_7, v_9\}$ is a theme community in $G_q$, and partially overlaps with the two theme communities in $G_p$.

There are several important benefits from modeling theme communities using maximal pattern trusses. First, there exists polynomial time algorithms to find maximal pattern trusses. Second, maximal pattern trusses of different theme networks may overlap with each other, which reflects the application scenarios where a vertex may participate in communities of different themes. Last, as to be proved in Sections 5.1 and 6.1, maximal pattern trusses have many good properties that enable us to design efficient mining and indexing algorithms for theme community finding.

3.3 Problem Definition and Complexity

Definition 3.7 (Theme Community Finding). Given a database network $G$ and a minimum cohesion threshold $\alpha$, the problem of theme community finding is to compute all theme communities in $G$.

Since extracting maximal connected subgraphs from a maximal pattern truss is straightforward, the core of the theme community finding problem is to identify the maximal pattern trusses of all theme networks. This is a challenging problem, since a database network can induce up to $2^{\cotruss}$ theme networks and each theme network may contain a maximal pattern truss. As a result, finding theme communities for all themes is computationally intractable.

Theorem 3.8. Given a database network $G$ and a minimum cohesion threshold $\alpha$, the problem of counting the number of theme communities in $G$ is #P-hard.

The proof of Theorem 3.8 is given in Appendix A.1.

In the rest of the paper, we develop an exact algorithm for theme community finding and investigate various techniques to speed up the search.

4 A BASELINE AND MAXIMAL PATTERN TRUSS DETECTION

In this section, we present a baseline for theme community finding. Before that, we introduce Maximal Pattern Truss Detector (MPTD) that detects the maximal pattern truss of a given theme network $G_p$ with respect to a threshold $\alpha$.

4.1 Maximal Pattern Truss Detection

Given $G_p$ and $\alpha$, an edge in $G_p$ is referred to as an unqualified edge if the edge cohesion is not larger than $\alpha$. The key idea of MPTD is to remove all unqualified edges so that the remaining edges and connected vertices constitute the maximal pattern truss.

As shown in Algorithm 1, MPTD consists of two phases. Phase 1 (Lines 1-8) computes the initial cohesion of each edge and pushes unqualified edges into queue $Q$. Phase 2 (Lines 9-18) removes the unqualified edges in $Q$ from $E_p$. Since removing $e_{ij}$ also breaks...
we introduce a baseline method, called (TCS). The key idea is to detect maximal pattern truss on each

The worst case happens when all edges are removed. Therefore, the

Algorithm 1: Maximal Pattern Truss Detector (MPTD)

| Input: A theme network $G_p$ and a user input $\alpha$. |
| Output: The maximal pattern truss $C_p^*(\alpha)$ in $G_p$. |
| 1: Initialize: $Q \leftarrow \emptyset$. |
| 2: for each $eij \in E_p$ do |
| 3: \hspace{1em} $ecoij(G_p) \leftarrow 0$. |
| 4: \hspace{1em} for each $vk \in \triangle_{ijk}$ do |
| 5: \hspace{2em} $ecoij(G_p) \leftarrow ecoij(G_p) + \min(f_i(p), f_j(p), f_k(p))$. |
| 6: end for |
| 7: \hspace{1em} if $ecoij(G_p) \leq \alpha$ then $Q.push(eij)$. |
| 8: end for |
| 9: while $Q \neq \emptyset$ do |
| 10: \hspace{1em} $Q.pop(eij)$. |
| 11: \hspace{1em} for each $vk \in \triangle_{ijk}$ do |
| 12: \hspace{2em} $ecojk(G_p) \leftarrow ecojk(G_p) - \min(f_i(p), f_j(p), f_k(p))$. |
| 13: \hspace{1em} $ecojk(G_p) \leftarrow ecojk(G_p) - \min(f_i(p), f_j(p), f_k(p))$. |
| 14: \hspace{1em} if $ecojk(G_p) \leq \alpha$ then $Q.push(ejk)$. |
| 15: \hspace{1em} if $ecojk(G_p) \leq \alpha$ then $Q.push(ejk)$. |
| 16: end for |
| 17: Remove $eij$ from $G_p$. |
| 18: end while |
| 19: $C_p^*(\alpha) \leftarrow G_p$. |
| 20: return $C_p^*(\alpha)$. |

$\triangle_{ijk}$, we update $ecoijk(G_p)$ and $ecoijk(G_p)$ in Lines 12-13. After the update, if $eik$ or $ejk$ becomes unqualified, they are pushed into $Q$ (Lines 14-15). Last, the remaining edges and connected vertices are returned as the maximal pattern truss.

We show the correctness of MPTD as follows. If $C_p^*(\alpha) = \emptyset$, then all edges in $E_p$ are removed as unqualified edges and MPTD returns $\emptyset$. If $C_p^*(\alpha) \neq \emptyset$, then all edges in $E_p \setminus E_p^*(\alpha)$ are removed as unqualified edges and MPTD returns exactly $C_p^*(\alpha)$.

The time complexity of Algorithm 1 is dominated by the complexity of triangle enumeration for each edge $eij$ in $E_p$. This requires checking all neighbouring vertices of $v_i$ and $v_j$, which costs $O(d(v_i) + d(v_j))$ time, where $d(v_i)$ and $d(v_j)$ are the degrees of $v_i$ and $v_j$, respectively. Since all edges in $E_p$ are checked, the cost for Lines 1-8 in Algorithm 1 is $O(|\sum_{eij \in E_p} (d(v_i) + d(v_j))) = O(|\sum_{v_i \in V} d^2(v_i))$. The cost of Lines 9-18 is also $O(|\sum_{v_i \in V} d^2(v_i)$). The worst case happens when all edges are removed. Therefore, the time complexity of MPTD is $O(|\sum_{v_i \in V} d^2(v_i))$. As a result, MPTD can efficiently find the maximal pattern truss of a sparse theme network.

4.2 Theme Community Scanner: A Baseline

Since a database network $G$ may induce up to $2^{\left|V\right|}$ theme networks, running MPTD on all theme networks is impractical. In this section, we introduce a baseline method, called Theme Community Scanner (TCS). The key idea is to detect maximal pattern truss on each theme network using MPTD, and improve the detection efficiency by pre-filtering out the patterns whose maximum frequencies in all vertex databases cannot reach a minimum frequency threshold $\epsilon$. The intuition is that patterns with low frequencies are not likely to be the theme of a theme community.

Algorithm 2: Generate Apriori Candidate Patterns

| Input: The set of Length-$(k-1)$ qualified patterns $p^{k-1}$. |
| Output: The set of Length-$k$ candidate patterns $M^k$. |
| 1: Initialize: $M_k \leftarrow \emptyset$. |
| 2: for $(p^{k-1}, q^{k-1}) \subseteq p^{k-1} \land |p^{k-1} \cup q^{k-1}| = k$ do |
| 3: $p^k \leftarrow p^{k-1} \cup q^{k-1}$. |
| 4: if all length-$(k-1)$ sub-patterns of $p^k$ are qualified then |
| 5: \hspace{1em} $M^k \leftarrow M^k \cup p^k$.
| 6: return $M^k$. |

Given a frequency threshold $\epsilon$, TCS first obtains the set of candidate patterns $P = \{p \mid \exists v_i \in V, f_i(p) > \epsilon\}$ by enumerating all patterns in each vertex database. Then, for each candidate pattern $p \in P$, we induce theme network $G_p$ and find the maximal pattern truss by MPTD. The final result is a set of maximal pattern trusses, denoted by $C(\alpha) = \{C_p^*(\alpha) | C_p^*(\alpha) \neq \emptyset, p \in P\}$.

The pre-filtering method of TCS improves the detection efficiency of theme communities, however, it may miss some theme communities, since a pattern $p$ with relatively small frequencies on all vertex databases can still form a good theme community, if a large number of vertices containing $p$ form a densely connected subgraph. As a result, TCS performs a trade-off between efficiency and accuracy. Detailed discussion on the effect of $\epsilon$ will be conducted in Section 7.1.

5 THEME COMMUNITY FINDING

In this section, we first explore several fundamental properties of maximal pattern truss that enable fast theme community finding. Then, we introduce two efficient and exact theme community finding methods.

5.1 Properties of Maximal Pattern Truss

Theorem 5.1 (Graph Anti-monotonicity). If $p_1 \subseteq p_2$, then $C_{p_1}^*(\alpha) \subseteq C_{p_2}^*(\alpha)$.

The proof of Theorem 5.1 is given in Appendix A.2.

Proposition 5.2 (Pattern Anti-monotonicity). For $p_1 \subseteq p_2$ and a cohesion threshold $\alpha$, the following two properties hold.

1. If $C_{p_1}^*(\alpha) \neq \emptyset$, then $C_{p_2}^*(\alpha) \neq \emptyset$.
2. If $C_{p_1}^*(\alpha) = \emptyset$, then $C_{p_2}^*(\alpha) = \emptyset$.

Proof. According to Theorem 5.1, since $p_1 \subseteq p_2$, $C_{p_2}^*(\alpha) \subseteq C_{p_1}^*(\alpha)$.

Proposition 5.3 (Graph Intersection Property). If $p_1 \subseteq p_3$ and $p_2 \subseteq p_3$, then $C_{p_1}^*(\alpha) \subseteq C_{p_3}^*(\alpha) \cap C_{p_2}^*(\alpha)$.

Proof. Since $p_1 \subseteq p_3$, according to Proposition 5.1, $C_{p_1}^*(\alpha) \subseteq C_{p_3}^*(\alpha)$. Similarly, $C_{p_2}^*(\alpha) \subseteq C_{p_3}^*(\alpha)$. The proposition follows.
Algorithm 3: Theme Community Finder Apriori (TCFA)

**Input:** A database network \( G \) and a user input \( \alpha \).

**Output:** The set of maximal pattern trusses \( \mathcal{C}(\alpha) \) in \( G \).

1. Initialize: \( \mathcal{P}^{1}, \mathcal{C}^{1}(\alpha), \mathcal{C}(\alpha) \leftarrow \mathcal{C}^{1}(\alpha), k \leftarrow 2. \)
2. while \( \mathcal{P}^{k-1} \neq \emptyset \) do
3.    Call Algorithm 2: \( \mathcal{M}^{k} \leftarrow \mathcal{P}^{k-1} \).
4.    \( \mathcal{P}^{k} \leftarrow \emptyset, \mathcal{C}^{k}(\alpha) \leftarrow \emptyset. \)
5.    for each length-\( k \) pattern \( p^{k} \in \mathcal{M}^{k} \) do
6.        Induce \( \mathcal{G}_{p^{k}} \) from \( G \).
7.        Compute \( \mathcal{C}_{p^{k}}^{\star}(\alpha) \) using \( \mathcal{G}_{p^{k}} \) by Algorithm 1.
8.        if \( \mathcal{C}_{p^{k}}^{\star}(\alpha) \neq \emptyset \) then \( \mathcal{C}(\alpha) \leftarrow \mathcal{C}(\alpha) \cup \mathcal{C}_{p^{k}}^{\star}(\alpha), \) and \( \mathcal{P}^{k} \leftarrow \mathcal{P}^{k} \cup \mathcal{C}_{p^{k}}^{\star}(\alpha). \)
9.    end for
10. \( \mathcal{C}(\alpha) \leftarrow \mathcal{C}(\alpha) \cup \mathcal{k}(\alpha). \)
11. \( k \leftarrow k + 1. \)
12. end while
13. return \( \mathcal{C}(\alpha) \).

### 5.2 Theme Community Finder Apriori

In this subsection, we introduce Theme Community Finder Apriori (TCFA) to solve the theme community finding problem. The key idea of TCFA is to improve theme community finding efficiency by early pruning unqualified patterns in an Apriori-like manner [1].

A pattern \( p \) is said to be unqualified if \( \mathcal{C}_{p}^{\star}(\alpha) = \emptyset \), and to be qualified if \( \mathcal{C}_{p}^{\star}(\alpha) \neq \emptyset \). For two patterns \( p_{1} \) and \( p_{2} \), if \( p_{1} \subseteq p_{2} \), \( p_{1} \) is called a sub-pattern of \( p_{2} \).

According to the second property of Proposition 5.2, for two patterns \( p_{1} \) and \( p_{2} \), if \( p_{1} \subseteq p_{2} \) and \( p_{1} \) is unqualified, then \( p_{2} \) is unqualified, thus \( p_{2} \) can be immediately pruned without running MPTD (Algorithm 1) on \( \mathcal{G}_{p_{1}} \). Therefore, we can prune a length-\( k \) pattern if any of its length-(\( k - 1 \)) sub-patterns is unqualified.

Algorithm 2 shows how we generate the set of length-\( k \) candidate patterns by retaining only the length-\( k \) patterns whose length-(\( k - 1 \)) sub-patterns are all qualified. This efficiently prunes a large number of unqualified patterns without running MPTD.

Algorithm 3 introduces the details of TCFA. Line 1 calculates the set of length-1 qualified patterns \( \mathcal{P}^{1} = \{ p \in S \mid \mathcal{C}_{p}^{1}(\alpha) \neq \emptyset, |p| = 1 \} \) and the corresponding set of maximal pattern trusses \( \mathcal{C}^{1}(\alpha) = \{ \mathcal{C}_{p}^{1}(\alpha) \mid p \in \mathcal{P}^{1} \} \). This requires to run MPTD (Algorithm 1) on each theme network induced by a single item in \( S \). Line 3 calls Algorithm 2 to generate the set of length-\( k \) candidate patterns \( \mathcal{M}^{k} \). Lines 5-9 remove the unqualified candidate patterns in \( \mathcal{M}^{k} \) by discarding every candidate pattern that cannot form a non-empty maximal pattern truss. In this way, Lines 2-12 iteratively generate the set of length-\( k \) qualified patterns \( \mathcal{P}^{k} \) from \( \mathcal{P}^{k-1} \) until no qualified patterns can be found. Last, the exact set of maximal pattern trusses \( \mathcal{C}(\alpha) \) is returned.

Comparing with the baseline TCS in Section 4.2, TCFA achieves a good efficiency improvement by effectively pruning a large number of unqualified patterns using the Apriori-like method in Algorithm 2. However, due to the well known limitation of Apriori [1], the set of candidate patterns \( \mathcal{M}^{k} \) is often very large and still contains many unqualified candidate patterns. Consequently, Lines 5-9 of Algorithm 3 become the bottleneck of TCFA. We solve this problem in the next subsection.

### 5.3 Theme Community Finder Intersection

The Theme Community Finder Intersection (TCFI) method significantly improves the efficiency of TCFA by further pruning unqualified patterns in \( \mathcal{M}^{k} \) using Proposition 5.3.

Consider pattern \( p^{k} \) of length \( k \) and patterns \( p^{k-1} \) and \( q^{k-1} \) of length \( k - 1 \). According to Proposition 5.3, if \( p^{k} = p^{k-1} \cup q^{k-1} \), then \( \mathcal{C}_{p^{k}}^{\star}(\alpha) \subseteq \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) \). Therefore, if \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) = \emptyset \), then \( \mathcal{C}_{p^{k}}^{\star}(\alpha) = \emptyset \). Thus, we can prune \( p^{k} \) immediately. If \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) \neq \emptyset \), we can induce theme network \( \mathcal{G}_{p^{k}} \) from \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) \) and find \( \mathcal{C}_{p^{k}}^{\star}(\alpha) \) within \( \mathcal{G}_{p^{k}} \) by MPTD.

Accordingly, TCFI improves TCFA by modifying only Line 6 of Algorithm 3. Instead of inducing \( \mathcal{G}_{p^{k}} \) from \( \mathcal{G}_{p^{k-1}} \), TCFI induces \( \mathcal{G}_{p^{k}} \) from \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) \) when \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) \neq \emptyset \). Here, \( p^{k-1} \) and \( q^{k-1} \) are qualified patterns in \( \mathcal{P}^{k-1} \) such that \( p^{k} = p^{k-1} \cup q^{k-1} \).

Using the graph intersection property in Proposition 5.3, TCFI efficiently prunes a large number of unqualified candidate patterns and dramatically improves the detection efficiency. First, TCFI prunes a large number of candidate patterns in \( \mathcal{M}^{k} \) by simply checking whether \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) = \emptyset \). Second, when \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) \neq \emptyset \), inducing \( \mathcal{G}_{p^{k}} \) from \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) \) is more efficient than inducing \( \mathcal{G}_{p^{k}} \) from \( \mathcal{G}_{p^{k-1}} \). Third, \( \mathcal{G}_{p^{k}} \) induced from \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) \) is often much smaller than \( \mathcal{G}_{p^{k-1}} \) from \( \mathcal{G}_{p^{k-1}} \), which significantly reduces the time cost of running MPTD on \( \mathcal{G}_{p^{k}} \). Fourth, according to Theorem 5.1, the size of a maximal pattern truss decreases when the length of the pattern increases. Thus, when a pattern grows longer, the size of \( \mathcal{C}_{p^{k-1}}^{\star}(\alpha) \cap \mathcal{C}_{q^{k-1}}^{\star}(\alpha) \) decreases rapidly, which significantly improves the pruning effectiveness of TCFI. Last, as to be discussed later in Section 7.2, most maximal pattern trusses are small local subgraphs in a database network. Such small subgraphs in different local regions of a large sparse database network generally do not intersect with each other.

### 6 THEME COMMUNITY INDEXING

When a user inputs a new cohesion threshold \( \alpha \), TCS, TCFA and TCFI have to recompute from scratch. Can we save the re-computation cost by decomposing and indexing all maximal pattern trusses to achieve fast user query answering? In this section, we propose the Theme Community Tree (TC-Tree) for fast user query answering. We first introduce how to decompose maximal pattern truss. Then, we illustrate how to build TC-Tree with decomposed maximal pattern trusses. Last, we present a querying method that efficiently answers user queries.

#### 6.1 Maximal Pattern Truss Decomposition

We first explore how to decompose a maximal pattern truss into multiple disjoint sets of edges.

**Theorem 6.1.** Given a theme network \( \mathcal{G}_{p} \), a cohesion threshold \( \alpha_{2} \) and a maximal pattern truss \( \mathcal{C}_{p}^{\star}(\alpha_{1}) \) in \( \mathcal{G}_{p} \) whose minimum edge...
cohesion is $\beta = \min_{e_{ij} \in E_p(a_i)} \text{eco}_{ij}(C_p^*(a_i))$, if $a_2 \geq \beta$, then $C_p^*(a_2) \subseteq C_p^*(a_1)$.

The proof of Theorem 6.1 is given in Appendix A.3.

Theorem 6.1 indicates that the size of maximal pattern truss $C_p^*(a_1)$ reduces only when cohesion threshold $a_2 \geq \min_{e_{ij} \in E_p(a_i)} \text{eco}_{ij}(C_p^*(a_i))$. Therefore, we can decompose a maximal pattern truss of $G_p$ into a sequence of disjoint sets of edges using a sequence of ascending cohesion thresholds $A_p = \alpha_0, \alpha_1, \cdots, \alpha_h$, where $\alpha_0 = 0, \alpha_k = \min_{e_{ij} \in E_p(a_{k-1})} \text{eco}_{ij}(C_p^*(a_{k-1}))$ and $k \in [1, h]$.

For $\alpha_0 = 0$, we call MPTD to calculate $C_p^*(\alpha_0)$, which is the largest maximal pattern truss in $G_p$. For $\alpha_1, \alpha_2, \cdots, \alpha_h$, we decompose $C_p^*(\alpha_0)$ into a sequence of removed sets of edges $R_p(\alpha_1), \ldots, R_p(\alpha_h)$, where $R_p(\alpha_k) = E_p^*(\alpha_{k-1}) \setminus E_p^*(\alpha_k)$ is the set of edges removed when $C_p^*(\alpha_{k-1})$ shrinks to $C_p^*(\alpha_k)$. The decomposition iterates until all edges in $C_p^*(\alpha_0)$ are removed.

The decomposition results are stored in a linked list $L_p = L_{p0}(\alpha_0), \ldots, L_{ph}(\alpha_h)$, where the $k$-th node stores $L_{pk}(\alpha_k) = (\alpha_k, R_p(\alpha_k))$. Since $L_p$ stores the same number of edges as in $E_p^*(\alpha_0)$, it does not incur much extra memory cost.

Using $L_p$, we can efficiently get the maximal pattern truss $C_p^*(\alpha)$ for any $\alpha \geq 0$ by first obtaining $E_p^*(\alpha)$ as

$$E_p^*(\alpha) = \bigcup_{\alpha_k \geq \alpha} R_p(\alpha_k)$$

and then inducing $V_p^*(\alpha)$ from $E_p^*(\alpha)$ according to Definition 3.3. $L_p$ also provides the nontrivial range of $\alpha$ for $G_p$. The upper bound of $\alpha$ in $G_p$ is $\alpha^* = \max A_p$, since $\forall a \geq \alpha^*$ we have $C_p^*(\alpha) = \emptyset$. Therefore, the nontrivial range of $\alpha$ for $G_p$ is $\alpha \in [0, \alpha^*)$. $\alpha^*$ can be easily obtained by visiting the last entry of $L_p$.

6.2 Theme Community Tree

A TC-Tree, denoted by $T$, is an extension of a set enumeration tree (SE-Tree) [23] and is carefully customized for efficient theme community indexing and query answering.

A SE-Tree is a basic data structure that enumerates all the subsets of a set $S$. A total order $<$ on the items in $S$ is assumed. Thus, any subset of $S$ can be written as a sequence of items in order $<$. Every node of a SE-Tree uniquely represents a subset of $S$. The root node represents $\emptyset$. For subsets $S_1$ and $S_2$ of $S$, the node representing $S_2$ is the child of the node representing $S_1$ if $S_1 \subseteq S_2$; $|S_2 \setminus S_1| = 1$; and $S_1$ is a prefix of $S_2$ when $S_1$ and $S_2$ are written as sequences of items in order $<$. Each node of a SE-Tree only stores the item in $S$ that is appended to the parent node to extend the child from the parent. In this way, the set of items represented by node $n_i$ is the union of the items stored in all the nodes along the path from the root to $n_i$. Figure 2 shows an example of the SE-tree of set $S = \{s_1, s_2, s_3, s_4\}$. For node $n_{13}$, the path from the root to $n_{13}$ contains nodes $\{n_0, n_1, n_6, n_{13}\}$, thus the set of items represented by $n_{13}$ is $\{s_1, s_3, s_4\}$.

A TC-Tree is an extension of a SE-Tree. In a TC-Tree, each node $n_i$ represents a pattern $p_i$, which is a subset of $S$. The item stored in $n_i$ is denoted by $n_i$. We also store the decomposed maximal pattern truss $\mathcal{L}_{p_i}$ in $n_i$. To save memory, we omit the nodes $n_i$ ($j \geq 1$) whose decomposed maximal pattern trusses are $\mathcal{L}_{p_i} = \emptyset$.

We can build a TC-Tree in a top-down manner efficiently. If $L_{p_i} = \emptyset$, we can prune the entire subtree rooted at $n_i$ immediately. This is because, for node $n_j$ and its descendant $n_k$, we have $p_j \subseteq p_k$. Since $L_{p_j} = \emptyset$, we can derive from Proposition 5.2 that $L_{p_k} = \emptyset$. As a result, all descendants of $n_j$ can be immediately pruned.

Algorithm 4 gives the details of building a TC-Tree $T$. Lines 2-5 generate the nodes at the first layer of $T$. Since the theme networks induced by different items in $S$ are independent, we can compute $L_{p_i}$ in parallel. Our implementation uses multiple threads for this step. Lines 6-12 iteratively build the rest of the nodes of $T$ in breadth-first order. Here, $n_f$'s siblings is the set of nodes that have the same parent as $n_f$. The children of $n_f$, denoted by $n_{ef}$, are built in Lines 8-11. In Line 9, we apply Proposition 5.3 to efficiently calculate $L_{p_{ef}}$. Since $p_e = p_f \cup p_b$, we have $p_e \subseteq p_f$ and $p_b \subseteq p_e$. From Proposition 5.3, we know $C_{p_{ef}}(\emptyset) \subseteq C_{p_f}(\emptyset) \cap C_{p_b}(\emptyset)$. Therefore, we can find $C_{p_{ef}}(\emptyset)$ within a small subgraph $C_{p_f}(\emptyset) \cap C_{p_b}(\emptyset)$ of MPTD, and then get $L_{p_{ef}}$ by decomposing $C_{p_{ef}}(\emptyset)$.

In summary, every node of a TC-Tree stores the decomposed maximal pattern truss $L_{p_i}$ of a unique pattern $p \subseteq S$. Since $L_{p_i}$ also stores the nontrivial range of $\alpha$ in $G_p$, we can easily use the TC-Tree to obtain the range of $\alpha$ for all theme networks in $G$. This range helps the users to set their queries.

6.3 Querying Theme Community Tree

In this subsection, we introduce how to query TC-Tree by a pattern $q$ and a cohesion threshold $\alpha_q$. The answer to query $(q, \alpha_q)$ is the set of maximal pattern trusses with respect to $\alpha_q$ for any sub-pattern of $q$, that is, $C_q(\alpha_q) = \{C_p^*(\alpha_q) \mid C_p^*(\alpha_q) \neq \emptyset, p \subseteq q\}$. With $C_q(\alpha_q)$, one can easily extract theme communities by finding the maximal connected subgraphs in all the retrieved maximal pattern trusses.

As shown in Algorithm 5, the querying method simply traverses the TC-Tree in breadth first order and collects maximal pattern trusses that satisfy the conditions of the answer.

The efficiency of Algorithm 5 comes from three factors. First, in Line 4, if $s_{n_q} \notin q$, then $p_e \subseteq q$ and the patterns of all descendants of $n_q$ are not sub-patterns of $q$. Therefore, we can prune the entire subtree rooted at $n_q$. Second, in Line 6, if $C_{p_{ef}}(\alpha_q) = \emptyset$, we can prune...
The entire subtree rooted at $n_c$, because, according to Proposition 5.2, no descendants of $n_c$ can induce a maximal pattern truss with respect to $a_q$. Last, in Line 5, getting $C_{p_c}(a_q)$ from $L_{p_c}$ is efficient using Equation 1.

In summary, TC-Tree enables fast user query answering. As demonstrated in Section 7.3, TC-Tree is easy to build and efficient to query, and scales well to index a large number of maximal pattern trusses using practical size of memory.

7 EXPERIMENTS

In this section, we evaluate the performance of Theme Community Scanner (TCS), Theme Community Finder Apriori (TCFA), Theme community Finder Intersection (TCFI) and Theme Community Tree (TC-Tree). TCS, TCFA and TCFI are implemented in Java. In order to efficiently process large database network, we implement TC-Tree in C++ and the parallel steps in Lines 2-5 of Algorithm 4 with 4 threads using OpenMP. All experiments are performed on a PC running Windows 7 with Core-i7-3370 CPU (3.40 GHz), 32GB memory and a 5400 rpm hard drive.

Since TC-Tree is an indexing method and it is not directly comparable with the other theme community detection methods, we compare the performance of TCS, TCFA and TCFI in Sections 7.1 and 7.2, and evaluate the performance of TC-Tree in Section 7.3.

The following datasets are used.

**Brightkite (BK)** The Brightkite dataset is a public check-in dataset produced by the location-based social networking website Brightkite.com [6]. It includes a friendship network of 58,228 users and 4,491,143 user check-ins that contain the check-in time and location. We construct a database network using this data set by taking the user friendship network as the network of the database network. Moreover, to create the vertex database for a user, we treat each check-in location as an item, and cut the check-in history of a user into periods of 2 days. The set of check-in locations within a period is transformed into a transaction. A theme community in this database network represents a group of friends who frequently visit the same set of places.

**Gowalla (GW)** The Gowalla dataset is a public dataset produced by the location-based social networking website Gowalla.com [6]. It includes a friendship network of 196,591 users and 6,442,890 user check-ins that contain the check-in time and location. We transform this dataset into a database network in the same way as BK.

**AMINER** The AMiner dataset is built from the Citation network v2 (CNV2) dataset [3]. CNV2 contains 1,397,240 papers. We transform it into a database network in the following two steps. First, we treat each author as a vertex and build an edge between a pair of authors who co-author at least one paper. Second, to build the vertex database for an author, we treat each abstract of a paper as an item, and all the keywords in the abstract of a paper are turned into a transaction. A theme community in this dataset represents a group of authors who collaboratively share the same research interest that is described by the same set of keywords.
Figure 3: The effects of user input $\alpha$ and threshold $\epsilon$ on BK, GW and AMINER. In (f)-(h) and (j)-(l), the performance of NP, NE and NV are all zero when $\alpha = 2.0$, however, we could not draw zero in the figure since the y-axes are in log scale.

**Synthetic (SYN) dataset.** The synthetic dataset is built to evaluate the scalability of TC-Tree. We first generate a network with 1 million vertices using the Java Universal Network/Graph Framework (JUNG) [18]. Then, in order to make the vertex databases of neighbour vertices share some common patterns, we generate the transaction databases on each vertex in three steps. First, we randomly select 1000 seed vertices. Then, to build the transaction database of each seed vertex, we randomly sample multiple itemsets from $S$ and store each sampled itemset as a transaction in the transaction database. Last, to build the transaction database of each non-seed vertex, we first sample multiple transactions from the transaction databases of the neighbor vertices, then randomly change 10% of the items in each sampled transaction to different items randomly picked in $S$. In this way, we iteratively generate the transaction databases of all vertices by a breadth first search of the network. For each vertex $v_i$ with degree $d(v_i)$, the number of transactions in vertex database $d_i$ is set to $[e^{0.13d(v_i)}]$, the length of each transaction in $d_i$ is set to $[e^{0.13d(v_i)}]$. The statistics of all datasets are given in Table 2.

### 7.1 Effect of Parameters

In this subsection, we analyze the effects of the cohesion threshold $\alpha$ and the frequency threshold $\epsilon$ for TCS in the real world database networks. The settings of parameters are $\alpha \in \{0.0, 0.1, 0.2, 0.3, 0.5, 1.0, 1.5, 2.0\}$ and $\epsilon \in \{0.1, 0.2, 0.3\}$. We do not evaluate the performance of TCS for $\epsilon = 0.0$ and $\epsilon > 0.3$, because TCS is too slow to stop in reasonable time when $\epsilon = 0.0$ and it loses too much accuracy when $\epsilon > 0.3$. Since TCS with $\epsilon \in \{0.1, 0.2, 0.3\}$ still run too slow on the original database networks of BK, GW and AMINER, we use small database networks that are sampled from the original database networks by performing a breadth first search of a randomly picked seed vertex. From BK and GW, we obtain sampled database networks with 10,000 edges. For AMINER, we sample a database network of 5,000 edges.

Figures 3(a), 3(e) and 3(i) show the time cost of all methods on BK, GW and AMINER, respectively. The cost of TCS does not change when $\alpha$ increases. This is because the cost of TCS is largely dominated by the size of the set of candidate patterns $P$ (see Section 4.2), which is not affected by $\alpha$. However, when $\epsilon$ increases, the size of $P$ reduces, thus the cost of TCS decreases. When $\alpha$ increases, the cost of TCFA and TCFI both decreases. This is because, for both TCFA and TCFI, a larger $\alpha$ reduces the size of the set of qualified patterns $P^{k-1}$, thus reduces the number of generated candidate patterns in $M^k$. This improves the effectiveness of the early pruning of TCFA and TCFI. The cost of TCFA is sensitive to $\alpha$. This is because the cost of TCFI is largely dominated by the number
of candidate patterns in $M_k$, which is generated by taking the length-$k$ union of the patterns in $p^{k-1}$. When $\alpha$ decreases, the size of $p^{k-1}$ increases rapidly, and the number of candidate patterns in $M_k$ becomes very large. In contrast, the cost of TCFI is stable with respect to $\alpha$ and is much lower than the cost of TCFA when $\alpha$ is small. The reason is that many maximal pattern trusses are small subgraphs that do not intersect with each other, thus many unqualified patterns in $M_k$ are easily pruned by TCFI using the graph intersection property in Proposition 5.3.

According to the experimental results on the small database network of AMINER of 3,000 edges, when $\alpha = 0$, TCFA calls MPTD 622,852 times, TCFI calls MPTD 152,396 times. This indicates that TCFI effectively prunes 75.5% of the candidate patterns. However, in Figure 3(i), TCFI is nearly 3 orders of magnitudes faster than TCFA when $\alpha = 0$. This is because, for each run of MPTD, TCFA computes the maximal pattern truss in the large theme network induced from the entire database network, however, TCFI computes the maximal pattern truss within the small theme network induced from the intersection of two maximal pattern trusses.

In Figures 3(a), 3(c) and 3(e), when $\alpha \geq 1$, the cost of TCFA is comparable with TCFI in all database networks. This is because, when $\alpha \geq 1$, GW and AMINER contain only one maximal pattern truss each, BK contains no more than three maximal pattern trusses that intersect with each other. In this case, TCFA does not generate many unqualified candidate patterns and TCFI does not prune any candidate patterns by the graph intersection property.

Figures 3(b)-(d), 3(f)-(h) and 3(j)-(l) show the performance in NP, NV of all methods on BK, GW and AMINER, respectively. TCFA and TCFI produce the same exact results for all values of $\alpha$ in all database networks. Whether TCS produces the exact results highly depends on the frequency threshold $\epsilon$, cohesion threshold $\alpha$ and the database network. For example, in Figures 3(b)-(d) and Figures 3(f)-(h), TCS ($\epsilon = 0.1$) cannot produce the same results as TCFA and TCFI unless $\alpha \geq 0.2$. For TCS ($\epsilon = 0.2$) and TCS ($\epsilon = 0.3$), in order to produce the same results as TCFA and TCFI, the proper values of $\alpha$ varies in different database networks. The reason is that vertices with small pattern frequencies can still form a good maximal pattern truss with large edge cohesion if they form a densely connected subgraph. Such maximal pattern trusses may be lost if the patterns with low frequencies are dropped by the pre-filtering step of TCS. This clearly shows that TCS performs a trade-off between efficiency and accuracy.

In summary, TCFI produces the best detection results of maximal pattern trusses and achieves the best efficiency performance for all values of user input $\alpha$ on all database networks.
7.2 Efficiency of Theme Community Finding

In this subsection, we analyze how the runtime of all methods changes when the size of the database network increases. For each database network, we generate a series of database networks with different sizes by sampling the original database network using the breath first search sampling method introduced in Section 7.1. Since TCS and TCFA run too slow on large database networks, we stop reporting the performance of TCS and TCFA when they cost more than one day. The performance of TCFI is evaluated on all sizes of database networks including the original ones. To evaluate the worst-case performance of all methods, we set $\alpha = 0$.

Figures 4(a), 4(e) and 4(i) show the time cost of all methods in BK, GW and AMINER, respectively. The cost of all methods increases when the number of sampled edges increases. This is because increasing the size of the database network increases the number of maximal pattern trusses. The cost of TCFI grows much slower than that of TCS and TCFA. The reason is that TCS generates candidate patterns by enumerating the patterns of all vertex databases, TCFA generates candidate patterns by pairwise unions of the patterns of the detected maximal pattern trusses. They both generate a large number of unqualified candidate patterns. TCFI generates candidate patterns by the pairwise unions of the patterns of two intersecting maximal pattern trusses, and runs MPTD on the small intersection of two maximal pattern trusses. This effectively reduces the number of candidate patterns and significantly reduces the time cost. As a result, TCFI achieves the best scalability and is more than two orders of magnitude faster than TCS and TCFA on large database networks.

Figures 4(b), 4(f) and 4(j) show the performance in NP of all methods. When the number of sampled edges increases, the NPs of all methods increase. This is because, increasing the size of database network increases the number of maximal pattern trusses, which is equal to NP. Both TCFI and TCFA produce the same exact results. However, due to the accuracy loss caused by pre-filtering the patterns with low frequencies, TCS cannot produce the same results as TCFI and TCFA.

In Figures 4(c)-(d), 4(g)-(h) and 4(k)-(l), we show the average number of vertices and edges in detected maximal pattern trusses by NV/NP and NE/NP, respectively. The trends of the curves of NV/NP and NE/NP are different in different database networks. This is because each database network is sampled by conducting breath first search from a randomly selected seed vertex, and the distributions of maximal pattern trusses are different in different database networks. If more smaller maximal pattern trusses are sampled earlier than larger maximal pattern trusses, NV/NP and NE/NP increase when the number of sampled edges increases. In contrast, if more smaller maximal pattern trusses are sampled later than larger maximal pattern trusses, NV/NP and NE/NP decrease. We can also see that the average numbers of vertices and edges in detected maximal pattern trusses are always small. This demonstrates that most maximal pattern trusses are small local subgraphs in a database network. Such small subgraphs in different local regions of a large sparse database network generally do not intersect with each other. Therefore, using the graph intersection property, TCFI can efficiently prune a large number of unqualified patterns and achieve much better scalability.

7.3 Efficiency of Theme Community Indexing

Now we analyze the indexing scalability and query efficiency of TC-Tree in both the real and synthetic database networks.

The indexing performance of TC-Tree in all database networks is shown in Table 3. “Indexing Time” is the cost to build a TC-Tree; “Memory” is the peak memory usage when building a TC-Tree; “#Nodes” is the number of nodes in a TC-Tree, which is also the number of maximal pattern trusses in a database network, since every TC-Tree node stores a unique maximal pattern truss.

Building a TC-Tree is efficient in both Indexing Time and Memory. For large database networks of AMINER and SYN, TC-Tree is much faster than TCS and TCFA. However, TCFI and TCFA are still faster than TCS on large database networks. This is because TCFI and TCFA efficiently prune a large number of unqualified patterns.
scales up pretty well in indexing more than 130 million nodes. TC-Tree costs more time on AMINER than SYN, because the database network of AMINER contains more unique items, which produces a larger set of candidate patterns.

We evaluate the performance of the TC-Tree querying method (Algorithm 5) under two settings: 1) Query by Alpha (QBA), which queries a TC-Tree with a threshold \( q \) by setting \( q = S \). 2) Query by Pattern (QBP), which queries a TC-Tree with pattern \( q \) by setting \( q = 0 \). The results are shown in Figure 5, where “Query Time” is the cost of querying a TC-Tree. “Retrieved Nodes (RN)” is the number of nodes retrieved from a TC-Tree.

To evaluate how QBA performance changes when \( q \) increases, we use \( q = \{0, 0.1, 0.2, \ldots, q_{\text{max}}\} \), which is a finite sequence that starts from 0.0 and is increased by 0.1 per step until Algorithm 5 returns \( 0, q_{\text{max}} \). \( q_{\text{max}} \) is the largest \( q \) when Algorithm 5 does not return \( 0 \). For each \( q \), the Query Time is the average of 1,000 runs.

In Figures 5(a)-(d), when \( q \) increases, both RN and Query Time decrease. This is because a larger \( q \) reduces the number of maximal pattern trusses, thus decreases RN and Query Time. Interestingly, in Figure 5(c), we have \( q_{\text{max}} = 106.9 \) in the database network of AMINER. This is because the CNV2 dataset [3] contains a paper about the “IBM Blue Gene/L super computer” that is co-authored by 115 authors.

Figures 5(c)-(d) show the excellent QBA performance of the proposed querying method (Algorithm 5) on the large database networks of AMINER and SYN. The proposed querying method can retrieve 1 million maximal pattern trusses within 1 second.

Figures 5(e)-(h) show how the performance of QBP changes when query pattern length increases. To generate query patterns with different length, we randomly sample 1,000 nodes from each layer of the TC-Tree and use the patterns of the sampled nodes as query patterns. Setting the query pattern length larger than the maximum depth of the TC-Tree does not make sense, since such patterns do not correspond to any maximal pattern trusses in the database network. Each Query Time reported is an average of 1,000 runs using different query patterns of the same length. As shown in Figures 5(e)-(h), both RN and Query Time increase when the Query Pattern Length increases. This is because querying the TC-Tree with a longer query pattern visits more TC-Tree nodes and retrieves more maximal pattern trusses.

In summary, TC-Tree is scalable in both time and memory when indexing large database networks.

### 7.4 A Case Study

In this subsection, we present some interesting theme communities discovered in the database network of AMINER using the proposed TC-Tree method. Each detected theme community represents a group of co-working scholars who share the same research interest characterized by a set of keywords. We present 6 interesting theme communities in Figure 6 and show the corresponding sets of keywords in Table 4.

Take Figures 6(a)-(b) as an example, the research interest of the theme community in Figure 6(a) is "data mining" and "sequential pattern". If we narrow down the research interest of this theme community by an additional keyword “intrusion detection”, the theme community in Figure 6(a) reduces to the theme community in Figure 6(b). This result demonstrates the fact that the size of a theme community reduces when the length of the pattern increases, which is consistent with Theorem 5.1.

The results in Figures 6(a)-(d) show that four researchers, Philip S. Yu, Jiawei Han, Jian Pei and Ke Wang, actively coauthor with different groups of researchers in different sub-disciplines of data mining, such as sequential pattern mining, intrusion detection, frequent pattern mining and privacy protection. These results demonstrate that the proposed TC-Tree method can discover arbitrarily overlapped theme communities with different themes.

We are surprised to see that TC-Tree also discovers the interdisciplinary research communities that are formed by researchers from different research areas. As shown in Figures 6(e)-(f), the research activities of Jiawei Han and Jian Pei are not limited in data mining. Figure 6(e) indicates that Jiawei Han collaborated with some researchers in linear discriminant analysis. Figure 6(f) shows that Jian Pei collaborated with some researchers in image retrieval. More interestingly, both Jiawei Han and Jian Pei collaborated with Jun

### Table 3: Indexing performance of TC-Tree.

| Indexing Time | Memory | #Nodes |
|---------------|--------|--------|
| BK            | 179 seconds | 0.3 GB | 18,581 |
| GW            | 1,594 seconds | 2.6 GB | 11,750,761 |
| AMINER        | 41,068 seconds | 28.3 GB | 152,067,019 |
| SYN           | 35,836 seconds | 26.6 GB | 132,985,944 |

### Table 4: The sets of keywords for theme communities.

| \( p_1 \) | \( p_2 \) | \( p_3 \) | \( p_4 \) | \( p_5 \) | \( p_6 \) |
|-----------|-----------|-----------|-----------|-----------|-----------|
| data mining, sequential pattern | data mining, sequential pattern, intrusion detection, | data mining, search space, complete set, pattern mining | data mining, sensitive information, privacy protection | principal component analysis, linear discriminant analysis, dimensionality reduction, component analysis | image retrieval, image database, relevance feedback, semantic gap |
Zhao, Xiaofei He, et al. The two theme communities in Figures 6(e)–(f) have a heavy overlap in vertices, but are different in themes.

In summary, the proposed theme community finding method allows arbitrary overlap between theme communities with different themes, and it is able to efficiently and accurately discover meaningful theme communities from large database networks.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we tackle the novel problem of finding theme communities from database networks. We first introduce the novel concept of database network, which is a natural abstraction of many real-world networks. Then, we propose TCFI and TC-Tree that efficiently discovers and indexes millions of theme communities in large database networks. As demonstrated by extensive experiments in both synthetic and real-world database networks, TCFI and TC-Tree are highly efficient and scalable. As future works, we will extend TCFI and TC-Tree to find theme communities from edge database network, where each edge is associated with a transaction database that describes complex relationships between vertices.

REFERENCES

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. VLDB 12/5 (1994), 487–499.

[2] Leman Akoglu, Hanghang Tong, Brendan Meeder, and Christos Faloutsos. 2012. PICs: Parameter-free Identification of Cohesive Subgraphs in Large Attributed Graphs. In SDM. 439–450.

[3] AMINER. 2010. https://aminer.org/citation. (2010).

[4] Ramnath Balasubramanyan and William W Cohen. 2011. Block-LDA: Jointly modeling entity-annotated text and entity-entity links. In SDM. Vol. 11. 450–461.

[5] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. TKDE 29, 12. 2090–2104.
This means that

\[
\min(f_i(p_1), f_j(p_1), f_k(p_1)) \geq \min(f_i(p_2), f_j(p_2), f_k(p_2))
\]  

(2)

Since \( V_{p_1} = V_{p_1}(\alpha) \) and \( E_{p_1} = E_{p_1}(\alpha) \), it follows that the set of triangles in \( E_{p_1}(\alpha) \) is exactly the same as the set of triangles in \( H_{p_1} \).

Therefore, we can derive from Equation 2 and Definition 3.1 that

\[
\forall e_{ij} \in E_{p_1}, eco_{ij}(H_{p_1}) \geq eco_{ij}(E_{p_1}(\alpha))
\]

Since \( E_{p_1}(\alpha) \) is the maximal pattern truss with respect to cohesion threshold \( \alpha \) in \( G_{p_1} \), it follows

\[
\forall e_{ij} \in E_{p_1}(\alpha), eco_{ij}(E_{p_1}(\alpha)) > \alpha
\]

Recall that \( E_{p_1} = E_{p_1}(\alpha) \), it follows

\[
\forall e_{ij} \in E_{p_1}, eco_{ij}(H_{p_1}) \geq eco_{ij}(E_{p_1}(\alpha)) > \alpha
\]

This means \( H_{p_1} \) is a pattern truss with respect to threshold \( \alpha \) in \( G_{p_1} \).

Recall that \( H_{p_1} = E_{p_1}(\alpha) \), it follows that \( E_{p_1}(\alpha) \) is a pattern truss in \( G_{p_1} \). The Theorem follows.

\[\Box\]

A.3 The Proof of Theorem 6.1

\[\text{Theorem A.3. Given a theme network } G_p, \text{ a cohesion threshold } \alpha_2 \text{ and a maximal pattern truss } C_p^*(\alpha_1) \text{ in } G_p \text{ whose minimum edge cohesion is } \beta = \min_{e_{ij} \in E_p(\alpha_1)} \text{eco}_{ij}(C_p(\alpha_1)), \text{ if } \alpha_2 \geq \beta, \text{ then } C_p^*(\alpha_2) \subseteq C_p^*(\alpha_1).
\]

Proof. First, we prove \( \alpha_2 > \alpha_1 \). Since \( E_{p_1}(\alpha_1) \) is a maximal pattern truss with respect to threshold \( \alpha_1 \), from Definition 3.3, we have \( \forall e_{ij} \in E_{p_1}(\alpha_1), eco_{ij}(C_p^*(\alpha_1)) > \alpha_1 \). Since \( \beta \) is the minimum edge cohesion of all the edges in \( C_p^*(\alpha_1) \), \( \beta > \alpha_1 \). Since \( \alpha_2 \geq \beta, \alpha_2 > \alpha_1 \).

Second, we prove \( C_p^*(\alpha_2) \subseteq C_p^*(\alpha_1) \). Since \( \alpha_2 > \alpha_1 \), we know from Definition 3.3 that \( \forall e_{ij} \in E_{p_1}(\alpha_2), eco_{ij}(C_p^*(\alpha_2)) > \alpha_2 > \alpha_1 \). This means that \( C_p^*(\alpha_2) \) is a pattern truss with respect to cohesion threshold \( \alpha_1 \) in \( G_p \). Since \( C_p^*(\alpha_1) \) is the maximal pattern truss with respect to cohesion threshold \( \alpha_1 \) in \( G_p \), from Definition 3.4 we have \( C_p^*(\alpha_2) \subseteq C_p^*(\alpha_1) \).

Last, we prove \( C_p^*(\alpha_2) \neq C_p^*(\alpha_1) \). Let \( e_{ij}^* \) be the edge with minimum edge cohesion \( \beta \) in \( E_{p_1}(\alpha_1) \). Since \( \alpha_2 \geq \beta \). According to Definition 3.3, \( e_{ij}^* \notin E_{p_1}(\alpha_2) \). Thus, \( E_{p_1}(\alpha_1) \neq E_{p_1}(\alpha_2) \) and \( C_p^*(\alpha_2) \neq C_p^*(\alpha_1) \).

Recall that \( C_p^*(\alpha_2) \subseteq C_p^*(\alpha_1) \). The theorem follows. \[\Box\]