Finding Theme Communities from Database Networks

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

Zhefeng Wang
Huawei Technologies
China
wangzhefeng@huawei.com

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

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

Yu Yang
Simon Fraser University
Burnaby, 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 TCFI, 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 a ranked list of theme communities from a TC-Tree of hundreds of millions of theme communities takes less than 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.

PVLDB Reference Format:
Lingyang Chu, Zhefeng Wang, Jian Pei, Yanyan Zhang, Yu Yang, Enhong Chen. Finding Theme Communities from Database Networks. *PVLDB*, 12(10): 1071-1084, 2019.
DOI: https://doi.org/10.14778/3339490.3339492

1. INTRODUCTION

Finding communities from large networks is a fundamental data mining problem that enjoys various applications, such as targeted advertisement in e-commerce networks [3, 21], friend recommendation in social networks [20, 23] and research group discovery in co-author networks [36].

Conventional community detection methods, such as graph partitioning [31, 26], dense subgraph mining [7, 27] and truss detection [9, 33], model real world networks as *simple networks* that contain only graph structures. To enhance the model of simple network, *vertex attributed networks* further profile the attributes of each vertex by a set of items [32, 38]. Community detection in vertex attributed networks aims to find communities such that all vertices in the same community contain the same set of items and are densely connected [5, 28]. Due to the homogeneity of vertex attributes, these communities are usually more meaningful and accurate [17, 25].

However, in most real world networks, the items of a vertex are not equally important and often do not co-occur all together. The valuable vertex information, such as item co-occurrence and the frequency of co-occurring items, is much beyond the limited descriptive power of the single set of items associated with the vertex. To tackle this problem, in this paper, we propose to model real world networks as *DataBase Networks* (DBN), where every vertex is associated with a transaction database named *vertex database*.

DBN is a natural descriptive model for many real world networks. For example, in e-commerce networks where each vertex represents a customer, every set of items purchased together by the customer is recorded as a transaction, and a vertex database stores all transactions of the customer. In location-based social networks where each vertex represents a user, the set of locations that the user checks in during a period (e.g., a day, a week or a month) can be recorded as a transaction, and the user's all transactions form a vertex database. The co-author network can also be enhanced by associating with each author a vertex database, where each transaction stores the keywords in an article published by the author.

The vertex databases of DBN accurately describe the co-occurrences and frequencies of different sets of items. A set of co-occurring items is called a *pattern* [1, 16], and a pattern with high frequency in a vertex database dominates the properties of the vertex. Therefore, in DBNs, it is more interesting to consider item co-occurrences and pattern frequencies, and find communities such that all vertices in the same community share the same dominant pattern and are densely connected. We call such communities *theme communities*, where each *theme* is the dominant pattern of a community.

For example, in a co-author network, authors are vertices and two authors are linked if they collaborated before. Each author is associated with a transaction database where each transaction is the set of keywords in an article published by the author. In this DBN, a pattern is a set of keywords that describes a research topic, and a theme community represents a group of closely collaborating authors who frequently publish papers in the same research topic.

Proceedings of the VLDB Endowment, Vol. 12, No. 10
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3339490.3339492

Can we adapt existing methods straightforwardly to find theme communities in DBNs? Unfortunately, the answer is no due to the following challenges.

First, a vertex database may contain an exponential number of patterns. Since the conventional methods that work on simple networks can only detect communities of one pattern at a time, it is computationally intractable to call those methods for each of an exponential number of patterns.

Second, the existing methods that work on vertex attributed networks only consider the case where each vertex is associated with a single set of items. In such a case, all items of a vertex occur together and thus the pattern frequencies for all patterns of the same vertex are trivially the same. Therefore, these methods cannot distinguish the different frequencies of different patterns in DBNs.

Last but not least, theme community finding can be a fast query answering service – different users can easily use the service to efficiently explore a DBN and quickly retrieve theme communities of their own interest in real time. Providing this service requires enumerating and indexing all theme communities in a DBN, which is challenging because a large DBN usually contains a huge number of arbitrarily overlapping theme communities, and even counting the number of theme communities is #P-hard.

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

First, we introduce the notion of DBN to model real networks in a natural and expressive manner. A DBN contains rich information about item co-occurrencess, pattern frequencies and graph/subgraph structures.

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

Third, we propose to find theme communities by enumerating *maximal* $(\mathbf{p}, \alpha)$*-trusses* in DBNs. A $(\mathbf{p}, \alpha)$-truss is a subgraph of a DBN such that the vertex databases of all vertices in the subgraph contain pattern $\mathbf{p}$, and the cohesion of every edge in the subgraph passes a non-negative threshold α. Here, the cohesion of an edge incorporates the rich information about item co-occurrencess and pattern frequencies comprehensively, and is closely related to the structure of the $(\mathbf{p}, \alpha)$-truss.

We propose a greedy algorithm and two effective pruning methods to enumerate maximal $(\mathbf{p}, \alpha)$-trusses. In our experiments, the pruning methods reduce the time cost of the greedy algorithm by over two orders of magnitude without any sacrifice in detection accuracy.

Fourth, we advocate the construction of a data warehouse of maximal $(\mathbf{p}, \alpha)$-trusses. To facilitate indexing and query answering in the data warehouse, we show that a maximal $(\mathbf{p}, \alpha)$-truss can be efficiently decomposed and stored in a linked list. We use the decomposition to design an efficient indexing tree, and develop a query answering method that takes less than 1 second to retrieve a ranked list of theme communities from the indexing tree storing hundreds of millions of theme communities. Moreover, to ensure usability and avoid user disappointment due to queries that retrieve no valid theme community, we develop a query recommendation method that efficiently explores the indexing tree to recommend to the user a ranked list of new queries, which are highly similar to the user query, and can retrieve meaningful theme communities with large cohesiveness.

Last, we report extensive experimental results that demonstrate the accuracy and efficiency of the proposed methods in the enumeration, indexing and query answering of theme communities. A case study shows that, by exploring the indexing tree that stores hundreds of millions of theme communities in a large co-author network, the query recommendation method quickly discovers meaningful research topics from an exponential number of combinations of user-interested keywords, and the query answering method efficiently retrieves the communities of closely collaborating scholars who frequently publish papers in those research topics discovered.

The rest of the paper is organized as follows. We review related works in Section 2 and formulate the theme community finding problem in Section 3. We present a baseline method and a maximal $(\mathbf{p}, \alpha)$-truss detection method in Section 4. We develop our major theme community finding algorithms in Section 5, and the indexing and querying answering algorithms in Section 6. 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, systematically finding theme communities from DBNs is novel and has not been formulated or tackled in literature. Broadly, it is related to frequent pattern mining, truss detection and vertex attributed network clustering.

Frequent pattern mining is to find frequent patterns from a transaction database. Some typical methods include Apriori [1] and FP-Growth [16]. Since frequent pattern mining methods do not handle densely connected graph structures, they cannot find communities in DBNs.

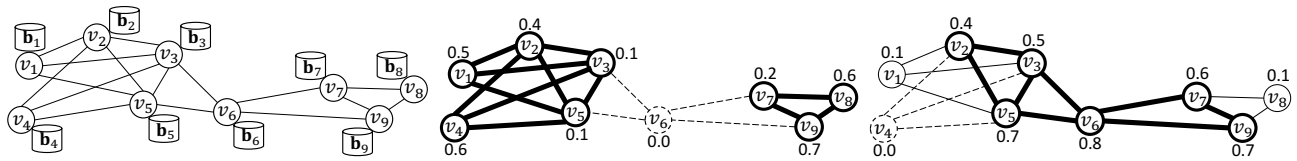
Truss detection aims to detect k-trusses from unweighted simple networks. Cohen [9] defined k-truss as a subgraph S where each edge is contained in at least $k-2$ triangles. As demonstrated by many studies [8, 9, 10], k-truss naturally models cohesive communities in social networks and is elegantly related to some other graph structures, such as k-core [30] and k-clique [22].

The elegance of k-truss attracts much research attention. Wang *et al.* [33] proposed two memory efficient methods to find k-trusses for all possible values of k in unweighted simple networks. Huang *et al.* [35] proposed (k, γ)-truss to extend the concept of k-truss from deterministic networks to probabilistic networks. Huang *et al.* [18] designed an online community search method to query k-trusses by vertices.

The above methods do not consider attributes of vertices, thus the detected communities may be hard to interpret due to the heterogeneity of vertex attributes [19]. One may also wonder whether we can enumerate theme communities by finding k-trusses in each of the simple networks induced by a pattern. However, this is impractical because a DBN may contain an exponential number of patterns.

Vertex attributed network clustering methods aim to find communities such that all vertices in the same community contain the same pattern and are densely connected. ABACUS [5] finds multi-dimensional communities by mining frequent itemsets. CoPaM [25] uses pruning methods to find maximal cohesive communities. Prado *et al.* [28] designed interestingness measures to find cohesive communities. Moosavi *et al.* [24] used frequent pattern mining to find cohesive groups of users sharing similar features. Huang *et al.* [19] formulated the ATC problem, which finds a community containing a query vertex. The ATC problem is NP-hard [19] and is substantially different from our problem, because our goal is to enumerate all theme communities, and even counting the number of theme communities in a DBN is #P-hard. There are also effective methods that detect communities in vertex attributed networks by graph weighting [32, 12], structural embedding [11, 13], statistical inference [4, 37] and subspace clustering [14, 34].

Since the above methods cannot distinguish different item co-occurrences and the corresponding pattern frequencies in DBNs, they cannot be directly applied to enumerate theme communities in DBNs. One may also wonder whether we



(a) Database network (b) Theme network $G_{\mathbf{p}}$ (c) Theme network $G_{\mathbf{q}}$

Figure 1: A toy example of DBN, theme network and theme community. The pattern frequencies are labeled beside each vertex. The theme communities marked bold in (b) are valid when $\alpha \in [0, 0.2)$. The theme community marked in bold in (c) is valid when $\alpha \in [0.2, 0.4)$.

can transform a DBN into a vertex attributed network by treating every transaction as a set-valued item or by taking the union of all transactions per vertex database. Unfortunately, these transformations are ineffective because they lose the valuable information about item co-occurrencess and pattern frequencies.

3. PROBLEM DEFINITION

In this section, we first introduce the notions of DBN, 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* $\mathbf{x}$ is a subset of S. A *transaction* $\mathbf{t}$ is an itemset. Transaction $\mathbf{t}$ is said to *contain* itemset $\mathbf{x}$ if $\mathbf{x} \subseteq \mathbf{t}$. The *length* of transaction $\mathbf{t}$, denoted by $|\mathbf{t}|$, is the number of items in $\mathbf{t}$. A *vertex database* $\mathbf{b} = \{\mathbf{t}_1, \ldots, \mathbf{t}_h\}$ $(h \geq 1)$ is a multi-set of transactions, that is, an itemset may appear multiple times as transactions in a vertex database.

A *database network* (DBN) is an undirected graph denoted by $G = (V, E, B, S)$, where each vertex is associated with a vertex database. Specifically, $V = \{v_1, \ldots, v_n\}$ is a set of vertices; $E = \{e_{ij} = (v_i, v_j) \mid v_i, v_j \in V\}$ is a set of edges; $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ is a set of vertex databases, where $\mathbf{b}_i$ is the vertex database associated with vertex v_i; and $S = \{s_1, \ldots, s_m\}$ is the set of items that constitute all vertex databases in B. That is, $S = \cup_{\mathbf{b}_i \in B} \cup_{\mathbf{t} \in \mathbf{b}_i} \mathbf{t}$.

Figure 1(a) gives a toy DBN, where the details of vertex databases are omitted due to the limit of space.

A *pattern* is an itemset $\mathbf{p} \subseteq S$ [1, 16]. The *length* of $\mathbf{p}$, denoted by $|\mathbf{p}|$, is the number of items in $\mathbf{p}$. The *absolute frequency* of $\mathbf{p}$ in vertex database $\mathbf{b}_i$ is the number of transactions in $\mathbf{b}_i$ that contain $\mathbf{p}$. The *relative frequency* of $\mathbf{p}$ in $\mathbf{b}_i$ is the proportion of such transactions in $\mathbf{b}_i$ [1, 16]. Our method works well for both absolute frequency and relative frequency. For the sake of clarity, we use relative frequency by default in the rest of the paper, and write the relative frequency of $\mathbf{p}$ in $\mathbf{b}_i$ as $f_i(\mathbf{p})$.

Given a pattern $\mathbf{p}$, the *theme network* $G_{\mathbf{p}}$ is a subgraph induced from G by the set of vertices satisfying $f_i(\mathbf{p}) > 0$, denoted by $G_{\mathbf{p}} = (V_{\mathbf{p}}, E_{\mathbf{p}})$, where $V_{\mathbf{p}} = \{v_i \in V \mid f_i(\mathbf{p}) > 0\}$ is the set of vertices and $E_{\mathbf{p}} = \{e_{ij} \in E \mid v_i, v_j \in V_{\mathbf{p}}\}$ is the set of edges. A subgraph $C_{\mathbf{p}}$ of $G_{\mathbf{p}}$ is written as $C_{\mathbf{p}} \subseteq G_{\mathbf{p}}$.

Figures 1(b)-(c) show two theme networks induced by different patterns $\mathbf{p}$ and $\mathbf{q}$, respectively. The edges and vertices in dashed lines are not contained in the theme networks.

We can induce a theme network by each pattern $\mathbf{p} \subseteq S$. A DBN G can induce at most $2^{|S|}$ theme networks, where G itself is the theme network of $\mathbf{p} = \emptyset$.

3.2 Theme Community

A *theme community* is a subgraph of a theme network such that the vertices form a cohesively connected subgraph. We define a theme community by extending the well-defined k-truss [9] from simple networks to DBN, such that it naturally models communities that are highly cohesive in both graph structure and pattern frequency, and is elegantly related to some well-established graph structures such as k-core [30] and k-clique [22]. Moreover, theme communities in different theme networks may arbitrarily overlap with each other, which reflects the application scenarios where a vertex may participate in communities of different themes.

The intuition of k-truss is that, if every edge of a community is contained in more triangles, then the vertices in the community are more densely connected [9]. Here, a *triangle*, denoted by $\triangle_{ijk} = \{v_i, v_j, v_k\}$, is a clique containing vertices v_i, v_j and v_k. If a triangle $\triangle_{ijk}$ is a subgraph of a graph T, we say $\triangle_{ijk}$ is in T.

In simple networks, Cohen [9] first measured the cohesion of an edge e_{ij} in a subgraph T by the number of triangles in T that contain e_{ij}; then he defines k-truss as a subgraph T such that the cohesion of every edge in T is at least $k-2$. Essentially, every triangle $\triangle_{ijk}$ in a subgraph T contributes a weight of 1.0 to the cohesion of each edge in $\triangle_{ijk}$, and the cohesion of an edge e_{ij} in T is the sum of the weights contributed by all triangles in T that contain e_{ij}.

Next, we illustrate how to integrate the rich information about item co-occurrencess and pattern frequencies with the edge cohesion in DBNs.

In a DBN, to incorporate item co-occurrences and the corresponding pattern frequencies, the contribution of a triangle $\triangle_{ijk}$ to the cohesion of each edge in $\triangle_{ijk}$ should be relevant to two factors: 1) a pattern $\mathbf{p} \subseteq S$, that is, a set of co-occurring items; and 2) the frequencies of $\mathbf{p}$ in the vertex databases $\mathbf{b}_i$, $\mathbf{b}_j$ and $\mathbf{b}_k$.

Intuitively, if $\mathbf{p}$ has higher frequencies in $\mathbf{b}_i$, $\mathbf{b}_j$ and $\mathbf{b}_k$, then all vertices in the triangle $\triangle_{ijk}$ are dominated by the pattern $\mathbf{p}$, thus $\triangle_{ijk}$ should contribute a heavier weight to the cohesion of each edge in $\triangle_{ijk}$. If the frequency of $\mathbf{p}$ is zero in any of $\mathbf{b}_i$, $\mathbf{b}_j$ and $\mathbf{b}_k$, then at least one vertex in $\triangle_{ijk}$ is not associated with $\mathbf{p}$, therefore $\triangle_{ijk}$ should not contribute any weight to the cohesion of any edge of $\triangle_{ijk}$.

Following the above intuition, for a pattern $\mathbf{p}$, we define the contribution of a triangle $\triangle_{ijk}$ to the cohesion of each edge in $\triangle_{ijk}$ as

$$\min\left(f_i(\mathbf{p}), f_j(\mathbf{p}), f_k(\mathbf{p})\right).$$

For any pattern $\mathbf{p}$, a triangle $\triangle_{ijk}$ contributes a none-zero weight to the cohesion of its edges if and only if $\triangle_{ijk}$ is in the theme network $G_{\mathbf{p}}$. The reason is that if $\triangle_{ijk}$ is not in $G_{\mathbf{p}}$, then the frequency of $\mathbf{p}$ is zero in at least one of the vertex databases $\mathbf{b}_i$, $\mathbf{b}_j$ and $\mathbf{b}_k$. In this case, $\triangle_{ijk}$ does not contribute any weight to the cohesion of its edges. Accordingly, we define the *edge cohesion* in DBNs as follows.

Definition 1 (Edge Cohesion). *Consider a pattern $\mathbf{p}$ and the theme network $G_{\mathbf{p}}$, for a subgraph $C_{\mathbf{p}} \subseteq G_{\mathbf{p}}$ and*

an edge e_{ij} in $C_{\mathbf{p}}$, the ***edge cohesion*** *of e_{ij} in $C_{\mathbf{p}}$ is*

$$eco_{ij}(C_{\mathbf{p}}) = \sum_{\triangle_{ijk} \subseteq C_{\mathbf{p}}} \min\left(f_i(\mathbf{p}), f_j(\mathbf{p}), f_k(\mathbf{p})\right)$$

EXAMPLE 1. *In Figure 1(b), for subgraph $C_{\mathbf{p}}$ induced by the set of vertices $\{v_1, v_2, v_3, v_4, v_5\}$, edge e_{12} is contained in $\triangle_{123}$ and $\triangle_{125}$, thus the edge cohesion of e_{12} is $eco_{12}(C_{\mathbf{p}}) = \min(f_1(\mathbf{p}),\ f_2(\mathbf{p}), f_3(\mathbf{p})) + \min(f_1(\mathbf{p}), f_2(\mathbf{p}),\ f_5(\mathbf{p})) = 0.2$.*

The edge cohesion in simple networks [9] is a special case of the edge cohesion in Definition 1. Because, when $f_i(\mathbf{p}) = 1$ for the vertex database of every vertex v_i in $C_{\mathbf{p}}$, $eco_{ij}(C_{\mathbf{p}})$ is exactly the number of triangles in $C_{\mathbf{p}}$ that contain e_{ij}.

Based on the edge cohesion in DBNs, we smoothly extend the notion of k-truss [9] in simple networks to the notion of $(\mathbf{p}, \alpha)$-*truss* in DBNs as follows.

DEFINITION 2 ($(\mathbf{p}, \alpha)$-TRUSS). *Given a pattern $\mathbf{p}$ and a minimum cohesion threshold $\alpha \geq 0$, a* $(\mathbf{p}, \alpha)$***-truss****, denoted by $C_{\mathbf{p},\alpha}$, is a subgraph of the theme network $G_{\mathbf{p}}$ such that the edge cohesion $eco_{ij}(C_{\mathbf{p},\alpha})$ of every edge in $C_{\mathbf{p},\alpha}$ is* ***larger than*** *α.*

A $(\mathbf{p}, \alpha)$-truss $C_{\mathbf{p},\alpha}$ is elegantly related to some well-established graph structures, such as k-truss [9], k-core [30] and k-clique [22], when $f_i(\mathbf{p}) = 1$ for the vertex database of every vertex v_i in $C_{\mathbf{p},\alpha}$. First, if $\alpha = k - 3$, $C_{\mathbf{p},\alpha}$ becomes a k-truss. Second, if $\alpha = k - 2$ and $C_{\mathbf{p},\alpha}$ is a maximal connected subgraph in $G_{\mathbf{p}}$, it is a k-core. Last, if $\alpha = k - 3$ and $C_{\mathbf{p},\alpha}$ contains k vertices, it is a k-clique.

A $(\mathbf{p}, \alpha)$-truss is not necessarily a connected subgraph, and the union of multiple $(\mathbf{p}, \alpha)$-trusses is still a $(\mathbf{p}, \alpha)$-truss. For example, consider the edges in bold in Figure 1(b). When $\alpha \in [0, 0.2)$, the two subgraphs induced by $\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_7, v_8, v_9\}$ are both $(\mathbf{p}, \alpha)$-trusses. The union of the two subgraphs is still a $(\mathbf{p}, \alpha)$-truss, but it is not a connected subgraph.

DEFINITION 3 (MAXIMAL $(\mathbf{p}, \alpha)$-TRUSS). *A* ***maximal*** $(\mathbf{p}, \alpha)$***-truss****, denoted by $C^*_{\mathbf{p},\alpha}$, is a $(\mathbf{p}, \alpha)$-truss in $G_{\mathbf{p}}$ such that any proper superset of $C^*_{\mathbf{p},\alpha}$ is not a $(\mathbf{p}, \alpha)$-truss in $G_{\mathbf{p}}$.*

Since the union of multiple $(\mathbf{p}, \alpha)$-trusses is still a $(\mathbf{p}, \alpha)$-truss, a maximal $(\mathbf{p}, \alpha)$-truss is the union of all $(\mathbf{p}, \alpha)$-trusses in $G_{\mathbf{p}}$. Apparently, a maximal $(\mathbf{p}, \alpha)$-truss is still not necessarily a connected subgraph.

Now we are ready to define theme community.

DEFINITION 4 (THEME COMMUNITY). *Every maximal connected subgraph in a maximal $(\mathbf{p}, \alpha)$-truss is a* ***theme community****.*

EXAMPLE 2. *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 $G_{\mathbf{p}}$. 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 $G_{\mathbf{q}}$, and partially overlaps with the two theme communities in $G_{\mathbf{p}}$.*

We write the set of all theme communities in the maximal $(\mathbf{p}, \alpha)$-truss as $\mathbb{T}_{\mathbf{p},\alpha} = \{T^1_{\mathbf{p},\alpha}, \ldots, T^m_{\mathbf{p},\alpha}\}$, where $T^i_{\mathbf{p},\alpha}$ is the i-th theme community in $\mathbb{T}_{\mathbf{p},\alpha}$ ($i \in \{1, \ldots, m\}$). Every theme community $T^i_{\mathbf{p},\alpha} \in \mathbb{T}_{\mathbf{p},\alpha}$ is also a $(\mathbf{p}, \alpha)$-truss. We define the cohesiveness of a theme community as follows.

DEFINITION 5 (COHESIVENESS OF THEME COMMUNITY). *The* ***cohesiveness*** *of a theme community is the minimum cohesion of its edges.*

There are several important benefits from modeling theme communities using maximal $(\mathbf{p}, \alpha)$-trusses. First, there exist polynomial time algorithms to find maximal $(\mathbf{p}, \alpha)$-trusses. Second, maximal $(\mathbf{p}, \alpha)$-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 $(\mathbf{p}, \alpha)$-trusses have many desirable properties that enable us to design efficient mining and indexing algorithms for theme community finding.

3.3 Problem Definition and Complexity

DEFINITION 6 (THEME COMMUNITY FINDING). *Given a DBN G and a minimum cohesion threshold α, the* ***problem of theme community finding*** *is to enumerate all theme communities in G.*

THEOREM 1 (COMPLEXITY). *Given a DBN G and a minimum cohesion threshold α, the problem of counting the number of theme communities in G is #P-hard.*

PROOF. We prove by a reduction from the *Frequent Pattern Counting* (FPC) problem, which is #P-complete [15].

Given a vertex database $\mathbf{b}$ and a minimum support threshold $\alpha \geq 0$, an instance of the FPC problem is to count the number of patterns $\mathbf{p}$ in $\mathbf{b}$ such that $f(\mathbf{p}) > \alpha$. Here, $f(\mathbf{p})$ is the frequency of $\mathbf{p}$ in $\mathbf{b}$.

We construct a DBN $G = (V, E, B, S)$, where $V = \{v_1, v_2, v_3\}$. $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$ forms a triangle; $B = \{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \mid \mathbf{b}_1 = \mathbf{b}_2 = \mathbf{b}_3 = \mathbf{b}\}$; and S is the set of items appearing in $\mathbf{b}$. Apparently, G can be constructed in $O(|\mathbf{b}|)$ time.

For any pattern $\mathbf{p} \subseteq S$, since $\mathbf{b}_1 = \mathbf{b}_2 = \mathbf{b}_3 = \mathbf{b}$, it follows $f_1(\mathbf{p}) = f_2(\mathbf{p}) = f_3(\mathbf{p}) = f(\mathbf{p})$. According to Definition 1, $eco_{12}(G_{\mathbf{p}}) = eco_{13}(G_{\mathbf{p}}) = eco_{23}(G_{\mathbf{p}}) = f(\mathbf{p})$. By Definition 4, $G_{\mathbf{p}}$ is a theme community in G if and only if $f(\mathbf{p}) > \alpha$. Therefore, for any threshold $\alpha \geq 0$, the number of theme communities in G is equal to the number of patterns in $\mathbf{b}$ satisfying $f(\mathbf{p}) > \alpha$, which is exactly the answer to the FPC problem. □

The theme community finding problem is challenging because a DBN G can induce up to $2^{|S|} - 1$ theme networks, and each theme network may contain many theme communities. Denote by n the number of vertices in G. Since the smallest theme community is a triangle that is not connected to any other vertices, the maximum number of theme communities in a theme network is $\lfloor \frac{n}{3} \rfloor$ for a fixed threshold $\alpha \geq 0$. Therefore, the maximum number of theme communities in G is $\lfloor \frac{n}{3} \rfloor (2^{|S|} - 1)$. However, this theoretical worst case seldom occurs in real-world datasets, since a large proportion of patterns do not induce any theme community. This is because a longer pattern $\mathbf{p}$ is less likely to appear in a vertex database, and the sparse structure of G makes it even harder for a small number of vertices containing $\mathbf{p}$ to form a theme community. As demonstrated by our experiments in section 7, the proposed methods efficiently enumerate and index theme communities in large real-world DBNs.

Since extracting theme communities (i.e., maximal connected subgraphs) from a maximal $(\mathbf{p}, \alpha)$-truss is straightforward, the core of the theme community finding problem is to identify the maximal $(\mathbf{p}, \alpha)$-trusses of all theme networks. In the rest of the paper, we develop an exact algorithm to find maximal $(\mathbf{p}, \alpha)$-truss and investigate various techniques to speed up the search.

4. BASELINE

In this section, we first introduce *Maximal $(\mathbf{p}, \alpha)$-Truss Detector* (MTD), which finds the maximal $(\mathbf{p}, \alpha)$-truss of a

given theme network $G_{\mathbf{p}}$. Then, we present a baseline for theme community finding.

4.1 Maximal $(\mathbf{p}, \alpha)$-Truss Detector

Given $G_{\mathbf{p}}$ and α, an edge in $G_{\mathbf{p}}$ is called an *unqualified edge* if the edge cohesion is not larger than α. The key idea of MTD is to iteratively remove all unqualified edges so that the remaining edges and connected vertices constitute the maximal $(\mathbf{p}, \alpha)$-truss. By Definition 1, the cohesion of an edge e_{ij} depends on the structure of subgraph $C_{\mathbf{p}} \subseteq G_{\mathbf{p}}$ that contains e_{ij}, thus removing an edge from $C_{\mathbf{p}}$ reduces the cohesion of some remaining edges. As a result, MTD iteratively updates the cohesion of all remaining edges after removing each unqualified edge. The iteration continues until there is no unqualified edge to remove.

As shown in Algorithm 1, MTD 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) iteratively removes the unqualified edges in Q from $E_{\mathbf{p}}$. Since removing e_{ij} also breaks $\triangle_{ijk}$, we update $eco_{ik}(G_{\mathbf{p}})$ and $eco_{jk}(G_{\mathbf{p}})$ in Lines 12-13. If consequently e_{ik} or e_{jk} become unqualified, they are pushed into Q (Lines 14-15). Finally, the surviving edges and connected vertices are returned as the maximal $(\mathbf{p}, \alpha)$-truss.

We show the correctness of MTD as follows. If $C^*_{\mathbf{p},\alpha} = \emptyset$, then all edges in $E_{\mathbf{p}}$ are removed as unqualified edges and MTD returns $\emptyset$. If $C^*_{\mathbf{p},\alpha} \neq \emptyset$, then only the edges in $E_{\mathbf{p}}$ that are not contained in $C^*_{\mathbf{p},\alpha}$ are removed as unqualified edges, and MTD returns exactly $C^*_{\mathbf{p},\alpha}$.

The time complexity of Algorithm 1 is dominated by the complexity of triangle enumeration for each edge e_{ij} in $E_{\mathbf{p}}$. This requires checking all neighboring vertices of v_i and v_j, which costs $\mathcal{O}(d(v_i) + d(v_j))$ time, where $d(v_i)$ and $d(v_j)$ are the degrees of v_i and v_j, respectively. Since all edges in $E_{\mathbf{p}}$ are checked, the cost for Lines 1-8 in Algorithm 1 is $\mathcal{O}(\sum_{e_{ij} \in E_{\mathbf{p}}} (d(v_i) + d(v_j))) = \mathcal{O}(\sum_{v_i \in V_{\mathbf{p}}} d^2(v_i))$. The cost of Lines 9-18 is also $\mathcal{O}(\sum_{v_i \in V_{\mathbf{p}}} d^2(v_i))$. The worst case happens when all edges are removed. Therefore, the time complexity of MTD is $\mathcal{O}(\sum_{v_i \in V_{\mathbf{p}}} d^2(v_i))$. In many real networks, most vertices have very small degrees. Thus, MTD can efficiently find the maximal $(\mathbf{p}, \alpha)$-truss of a sparse theme network.

4.2 Theme Community Scanner: A Baseline

Since a DBN G may induce up to $2^{|S|}$ theme networks, running MTD on all theme networks is impractical. Intuitively, patterns with low frequencies may be less likely to induce a theme community. Thus, a simple idea is to first filter out the patterns whose maximum frequencies in all vertex databases fail a minimum frequency threshold ϵ, then apply MTD to detect the maximal $(\mathbf{p}, \alpha)$-truss on each theme network induced by the remaining patterns.

Following the above idea, we introduce a baseline method, called *Theme Community Scanner* (TCS). Given a frequency threshold ϵ, TCS first obtains the set of candidate patterns $\mathcal{P} = \{\mathbf{p} \mid \exists v_i \in V, f_i(\mathbf{p}) \geq \epsilon\}$ by enumerating all patterns in each vertex database. Then, for each candidate pattern $\mathbf{p} \in \mathcal{P}$, we induce theme network $G_{\mathbf{p}}$ and find the maximal $(\mathbf{p}, \alpha)$-truss by MTD. The final result is a set of maximal $(\mathbf{p}, \alpha)$-trusses, denoted by $\mathbb{C}(\alpha) = \{C^*_{\mathbf{p},\alpha} \mid C^*_{\mathbf{p},\alpha} \neq \emptyset, \mathbf{p} \in \mathcal{P}\}$.

The filtering step of TCS improves the detection efficiency of theme communities, however, it may miss some theme communities, since a pattern $\mathbf{p}$ with relatively small frequencies on all vertex databases can still form a good theme community, if a large number of vertices containing $\mathbf{p}$ form a densely connected subgraph. As a result, TCS trades accuracy for efficiency, which, however, is not as effective as one may expect according to our experiments in Section 7.2.

Algorithm 1: Maximal $(\mathbf{p}, \alpha)$-Truss Detector

Input: A theme network $G_{\mathbf{p}}$ and a user input α.
Output: The maximal $(\mathbf{p}, \alpha)$-truss $C^*_{\mathbf{p},\alpha}$ in $G_{\mathbf{p}}$.

1: Initialize: $Q \leftarrow \emptyset$.
2: **for** each $e_{ij} \in E_{\mathbf{p}}$ **do**
3: $\quad eco_{ij}(G_{\mathbf{p}}) \leftarrow 0$.
4: $\quad$ **for** each $v_k \in \triangle_{ijk}$ **do**
5: $\quad\quad eco_{ij}(G_{\mathbf{p}}) \leftarrow eco_{ij}(G_{\mathbf{p}}) + \min(f_i(\mathbf{p}), f_j(\mathbf{p}), f_k(\mathbf{p}))$.
6: $\quad$ **end for**
7: $\quad$ **if** $eco_{ij}(G_{\mathbf{p}}) \leq \alpha$ **then** $Q.push(e_{ij})$.
8: **end for**
9: **while** $Q \neq \emptyset$ **do**
10: $\quad e_{ij} \leftarrow Q.pop()$.
11: $\quad$ **for** each $v_k \in \triangle_{ijk}$ **do**
12: $\quad\quad eco_{ik}(G_{\mathbf{p}}) \leftarrow eco_{ik}(G_{\mathbf{p}}) - \min(f_i(\mathbf{p}), f_j(\mathbf{p}), f_k(\mathbf{p}))$.
13: $\quad\quad eco_{jk}(G_{\mathbf{p}}) \leftarrow eco_{jk}(G_{\mathbf{p}}) - \min(f_i(\mathbf{p}), f_j(\mathbf{p}), f_k(\mathbf{p}))$.
14: $\quad\quad$ **if** $eco_{ik}(G_{\mathbf{p}}) \leq \alpha$ **then** $Q.push(e_{ik})$.
15: $\quad\quad$ **if** $eco_{jk}(G_{\mathbf{p}}) \leq \alpha$ **then** $Q.push(e_{jk})$.
16: $\quad$ **end for**
17: $\quad$ Remove e_{ij} from $G_{\mathbf{p}}$.
18: **end while**
19: **return** $C^*_{\mathbf{p},\alpha} = G_{\mathbf{p}}$.

5. THEME COMMUNITY FINDING

In this section, we first explore several fundamental properties of maximal $(\mathbf{p}, \alpha)$-truss, then apply them to develop two fast and exact theme community finding methods.

5.1 Properties of Maximal $(\mathbf{p}, \alpha)$-Truss

THEOREM 2 (GRAPH ANTI-MONOTONICITY). *If patterns $\mathbf{p}_1 \subseteq \mathbf{p}_2$, then maximal $(\mathbf{p}, \alpha)$-trusses $C^*_{\mathbf{p}_2,\alpha} \subseteq C^*_{\mathbf{p}_1,\alpha}$.*

PROOF. Since $C^*_{\mathbf{p}_1,\alpha}$ is the union of all $(\mathbf{p}_1, \alpha)$-trusses, we prove $C^*_{\mathbf{p}_2,\alpha} \subseteq C^*_{\mathbf{p}_1,\alpha}$ by proving $C^*_{\mathbf{p}_2,\alpha}$ is also a $(\mathbf{p}_1, \alpha)$-truss.

First, we prove that there exists a subgraph $H_{\mathbf{p}_1}$ in $G_{\mathbf{p}_1}$ such that $H_{\mathbf{p}_1} = C^*_{\mathbf{p}_2,\alpha}$, that is, $H_{\mathbf{p}_1}$ and $C^*_{\mathbf{p}_2,\alpha}$ have exactly the same sets of vertices and edges. Since $\mathbf{p}_1 \subseteq \mathbf{p}_2$, it follows the anti-monotonicity [1, 16] that $\forall v_i \in V, f_i(\mathbf{p}_1) \geq f_i(\mathbf{p}_2)$. Thus, $G_{\mathbf{p}_2} \subseteq G_{\mathbf{p}_1}$. Since $C^*_{\mathbf{p}_2,\alpha} \subseteq G_{\mathbf{p}_2}$, $C^*_{\mathbf{p}_2,\alpha} \subseteq G_{\mathbf{p}_1}$. Therefore, $H_{\mathbf{p}_1}$ exists.

Next, we prove $H_{\mathbf{p}_1}$ and $C^*_{\mathbf{p}_2,\alpha}$ are both $(\mathbf{p}_1, \alpha)$-trusses. Since $\forall v_i \in V, f_i(\mathbf{p}_1) \geq f_i(\mathbf{p}_2)$, the following inequality holds for every triangle $\triangle_{ijk}$ in $H_{\mathbf{p}_1}$.

$$\min(f_i(\mathbf{p}_1), f_j(\mathbf{p}_1), f_k(\mathbf{p}_1)) \geq \min(f_i(\mathbf{p}_2), f_j(\mathbf{p}_2), f_k(\mathbf{p}_2))$$

Since $H_{\mathbf{p}_1} = C^*_{\mathbf{p}_2,\alpha}$, it follows the above inequality that $eco_{ij}(H_{\mathbf{p}_1}) \geq eco_{ij}(C^*_{\mathbf{p}_2,\alpha})$ for every edge e_{ij} in $H_{\mathbf{p}_1}$.

Since $C^*_{\mathbf{p}_2,\alpha}$ is the maximal $(\mathbf{p}_2, \alpha)$-truss, $eco_{ij}(C^*_{\mathbf{p}_2,\alpha}) > \alpha$ for every edge e_{ij} in $C^*_{\mathbf{p}_2,\alpha}$.

Now we can conclude from the above that $eco_{ij}(H_{\mathbf{p}_1}) \geq eco_{ij}(C^*_{\mathbf{p}_2,\alpha}) > \alpha$ for every edge e_{ij} in $H_{\mathbf{p}_1}$. This means both $H_{\mathbf{p}_1}$ and $C^*_{\mathbf{p}_2,\alpha}$ are $(\mathbf{p}_1, \alpha)$-trusses. The theorem follows. □

PROPOSITION 1 (PATTERN ANTI-MONOTONICITY). *For patterns $\mathbf{p}_1 \subseteq \mathbf{p}_2$ and a cohesion threshold α,*

1. *If $C^*_{\mathbf{p}_2,\alpha} \neq \emptyset$, then $C^*_{\mathbf{p}_1,\alpha} \neq \emptyset$.*
2. *If $C^*_{\mathbf{p}_1,\alpha} = \emptyset$, then $C^*_{\mathbf{p}_2,\alpha} = \emptyset$.*

PROOF. According to Theorem 2, since $\mathbf{p}_1 \subseteq \mathbf{p}_2$, $C^*_{\mathbf{p}_2,\alpha} \subseteq C^*_{\mathbf{p}_1,\alpha}$. The proposition follows immediately. □

PROPOSITION 2 (GRAPH INTERSECTION PROPERTY). *If $\mathbf{p}_1 \subseteq \mathbf{p}_3$ and $\mathbf{p}_2 \subseteq \mathbf{p}_3$, then $C^*_{\mathbf{p}_3,\alpha} \subseteq C^*_{\mathbf{p}_1,\alpha} \cap C^*_{\mathbf{p}_2,\alpha}$.*

PROOF. By Theorem 2, since $\mathbf{p}_1 \subseteq \mathbf{p}_3$, $C^*_{\mathbf{p}_3,\alpha} \subseteq C^*_{\mathbf{p}_1,\alpha}$. Similarly, $C^*_{\mathbf{p}_3,\alpha} \subseteq C^*_{\mathbf{p}_2,\alpha}$. The proposition follows. □

Algorithm 2: Generate Apriori Candidate Patterns

Input: The length-$(k-1)$ qualified patterns $\mathcal{P}^{k-1}$.
Output: The set of length-k candidate patterns $\mathcal{M}^k$.

1: Initialize: $\mathcal{M}^k \leftarrow \emptyset$.
2: **for** $\{\mathbf{p}, \mathbf{q}\} \subset \mathcal{P}^{k-1} \wedge |\mathbf{p} \cup \mathbf{q}| = k$ **do**
3: $\quad \mathbf{h} \leftarrow \mathbf{p} \cup \mathbf{q}$.
4: $\quad$ **if** all length-$(k-1)$ sub-patterns of $\mathbf{h}$ are qualified **then** $\mathcal{M}^k \leftarrow \mathcal{M}^k \cup \mathbf{h}$.
5: **end for**
6: **return** $\mathcal{M}^k$.

Algorithm 3: Theme Community Finder Apriori

Input: A DBN G and a user input α.
Output: The set of maximal $(\mathbf{p}, \alpha)$-trusses $\mathbb{C}$ in G.

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

5.2 Theme Community Finder Apriori

In this subsection, we introduce algorithm *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 $\mathbf{p}$ is said to be *unqualified* if $C^*_{\mathbf{p},\alpha} = \emptyset$, and to be *qualified* if $C^*_{\mathbf{p},\alpha} \neq \emptyset$. For two patterns $\mathbf{p}_1$ and $\mathbf{p}_2$, if $\mathbf{p}_1 \subseteq \mathbf{p}_2$, $\mathbf{p}_1$ is called a *sub-pattern* of $\mathbf{p}_2$.

According to the second item in Proposition 1, for two patterns $\mathbf{p}_1$ and $\mathbf{p}_2$, if $\mathbf{p}_1 \subseteq \mathbf{p}_2$ and $\mathbf{p}_1$ is unqualified, then $\mathbf{p}_2$ is unqualified, either, thus $\mathbf{p}_2$ can be immediately pruned. 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 all length-$(k-1)$ sub-patterns are qualified.

Algorithm 3 introduces the details of TCFA. Line 1 computes the set of length-1 qualified patterns $\mathcal{P}^1 = \{\mathbf{p} \subset S \mid C^*_{\mathbf{p},\alpha} \neq \emptyset, |\mathbf{p}| = 1\}$ and the corresponding set of maximal $(\mathbf{p}, \alpha)$-trusses $\mathbb{C}^1 = \{C^*_{\mathbf{p},\alpha} \mid \mathbf{p} \in \mathcal{P}^1\}$. This requires to run MTD 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 $(\mathbf{p}, \alpha)$-truss. In this way, we 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 $(\mathbf{p}, \alpha)$-trusses $\mathbb{C}$ 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. However, due to the 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 next.

5.3 Theme Community Finder Intersection

The *Theme Community Finder Intersection* (TCFI) method significantly improves the efficiency of TCFA by pruning unqualified patterns in $\mathcal{M}^k$ using Proposition 2.

Consider pattern $\mathbf{h}$ of length k and patterns $\mathbf{p}$ and $\mathbf{q}$ both of length $k-1$. According to Proposition 2, if $\mathbf{h} = \mathbf{p} \cup \mathbf{q}$, then $C^*_{\mathbf{h},\alpha} \subseteq C^*_{\mathbf{p},\alpha} \cap C^*_{\mathbf{q},\alpha}$. Therefore, let $Z = C^*_{\mathbf{p},\alpha} \cap C^*_{\mathbf{q},\alpha}$, if $Z = \emptyset$, then $C^*_{\mathbf{h},\alpha} = \emptyset$. That is, we can prune $\mathbf{h}$ immediately. If $Z \neq \emptyset$, we can induce theme network $G_{\mathbf{h}}$ from Z and find $C^*_{\mathbf{h},\alpha}$ within $G_{\mathbf{h}}$ by MTD.

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

TCFI dramatically improves the detection efficiency. First, TCFI prunes a large number of candidate patterns in $\mathcal{M}^k$ by efficiently checking whether $Z = \emptyset$. Second, when $Z \neq \emptyset$, inducing $G_{\mathbf{h}}$ from Z is more efficient than inducing $G_{\mathbf{h}}$ from G, since Z is often much smaller than G. Third, $G_{\mathbf{h}}$ induced from Z is often much smaller than $G_{\mathbf{h}}$ induced from G, which significantly reduces the time cost of running MTD on $G_{\mathbf{h}}$. Last, according to Theorem 2, the size of a maximal $(\mathbf{p}, \alpha)$-truss decreases when the length of the pattern increases. Thus, when a pattern grows longer, the size of Z decreases rapidly, which significantly improves the pruning effectiveness of TCFI.

6. THEME COMMUNITY INDEXING

In practice, different users may be interested in theme communities in different maximal $(\mathbf{p}, \alpha)$-trusses. Unfortunately, for every new threshold α, TCS, TCFA and TCFI have to recompute from scratch. Can we save the recomputation cost by providing a fast query answering service that allows users to efficiently explore a DBN and quickly retrieve theme communities of their own interest? In this section, we propose *Theme Community Tree* (TC-Tree) to provide fast query answering service by decomposing and indexing all maximal $(\mathbf{p}, \alpha)$-trusses in a DBN.

We first introduce how to decompose maximal $(\mathbf{p}, \alpha)$-truss. Then, we illustrate how to build TC-Tree with decomposed maximal $(\mathbf{p}, \alpha)$-trusses. Last, we present a query answering method that can efficiently retrieve a ranked list of theme communities to answer a user query, and can also recommend a ranked list of meaningful new queries to address user disappointment when a query does not return any community.

6.1 Maximal $(\mathbf{p}, \alpha)$-Truss Decomposition

In this subsection, we introduce how to decompose a maximal $(\mathbf{p}, \alpha)$-truss into multiple disjoint sets of edges.

THEOREM 3. *Given a maximal $(\mathbf{p}, \alpha_1)$-truss $C^*_{\mathbf{p},\alpha_1}$ with minimum edge cohesion $\beta_{\mathbf{p},\alpha_1}$, for any cohesion threshold $\alpha_2 \geq \beta_{\mathbf{p},\alpha_1}$, $C^*_{\mathbf{p},\alpha_2} \subset C^*_{\mathbf{p},\alpha_1}$.*

PROOF. First, we prove $\alpha_2 > \alpha_1$. By Definition 2, for any edge e_{ij} in $C^*_{\mathbf{p},\alpha_1}$, $eco_{ij}(C^*_{\mathbf{p},\alpha_1}) > \alpha_1$. Since $\beta_{\mathbf{p},\alpha_1}$ is the minimum edge cohesion of $C^*_{\mathbf{p},\alpha_1}$, $\beta_{\mathbf{p},\alpha_1} > \alpha_1$. Since $\alpha_2 \geq \beta_{\mathbf{p},\alpha_1}$, $\alpha_2 > \alpha_1$.

Second, we prove $C^*_{\mathbf{p},\alpha_2} \subseteq C^*_{\mathbf{p},\alpha_1}$. Since $\alpha_2 > \alpha_1$, it follows Definition 2 that, for any edge e_{ij} in $C^*_{\mathbf{p},\alpha_2}$, $eco_{ij}(C^*_{\mathbf{p},\alpha_2}) > \alpha_2 > \alpha_1$. This means $C^*_{\mathbf{p},\alpha_2}$ is also a $(\mathbf{p}, \alpha_1)$-truss. Since $C^*_{\mathbf{p},\alpha_1}$ is the maximal $(\mathbf{p}, \alpha_1)$-truss, $C^*_{\mathbf{p},\alpha_2} \subseteq C^*_{\mathbf{p},\alpha_1}$.

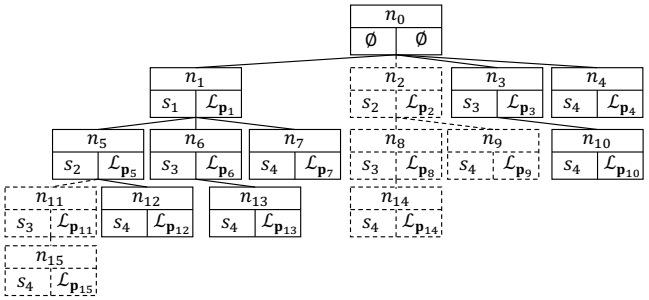


Figure 2: An example of SE-Tree and TC-Tree when $S = \{s_1, s_2, s_3, s_4\}$ and $\mathcal{L}_{\mathbf{p}_0} = \mathcal{L}_{\mathbf{p}_2} = \mathcal{L}_{\mathbf{p}_8} = \mathcal{L}_{\mathbf{p}_9} = \mathcal{L}_{\mathbf{p}_{11}} = \mathcal{L}_{\mathbf{p}_{14}} = \mathcal{L}_{\mathbf{p}_{15}} = \emptyset$. SE-Tree includes all nodes marked in solid and dashed lines. TC-tree contains only the nodes in solid line.

Last, we prove $C^*_{\mathbf{p},\alpha_2} \neq C^*_{\mathbf{p},\alpha_1}$. Let e^*_{ij} be the edge that has the minimum edge cohesion $\beta_{\mathbf{p},\alpha_1}$ in $C^*_{\mathbf{p},\alpha_1}$. Since $\alpha_2 \geq \beta_{\mathbf{p},\alpha_1}$, e^*_{ij} is not an edge of $C^*_{\mathbf{p},\alpha_2}$. Thus, $C^*_{\mathbf{p},\alpha_2} \neq C^*_{\mathbf{p},\alpha_1}$. Recall that $C^*_{\mathbf{p},\alpha_2} \subseteq C^*_{\mathbf{p},\alpha_1}$, the theorem follows. □

Theorem 3 indicates that the size of $C^*_{\mathbf{p},\alpha_1}$ is smaller than $C^*_{\mathbf{p},\alpha_2}$ only when $\alpha_2 \geq \beta_{\mathbf{p},\alpha_1}$. Thus, we can iteratively compute a sequence of ascending cohesion thresholds $\mathcal{A}_\mathbf{p} = \alpha_0, \alpha_1, \ldots, \alpha_h$, where $\alpha_0 = 0$, $\alpha_k = \beta_{\mathbf{p},\alpha_{k-1}}$ for $k \in \{1, \ldots, h\}$, and α_h is the largest α in $G_\mathbf{p}$ such that $C^*_{\mathbf{p},\alpha} = \emptyset$ for all $\alpha \geq \alpha_h$.

We use $\mathcal{A}_\mathbf{p}$ to decompose a maximal $(\mathbf{p}, \alpha)$-truss as follows. First, we call MTD to compute $C^*_{\mathbf{p},\alpha_0}$, which is the largest maximal $(\mathbf{p}, \alpha)$-truss in $G_\mathbf{p}$. Then, for $\alpha_1, \ldots, \alpha_h$, we decompose $C^*_{\mathbf{p},\alpha_0}$ into a sequence of sets of edges $R_{\mathbf{p},\alpha_1}, \ldots, R_{\mathbf{p},\alpha_h}$, where $R_{\mathbf{p},\alpha_k}$ is the set of edges that are contained in $C^*_{\mathbf{p},\alpha_{k-1}}$ but not in $C^*_{\mathbf{p},\alpha_k}$.

The decomposition results are stored in a linked list $\mathcal{L}_\mathbf{p} = \mathcal{L}_{\mathbf{p},\alpha_1}, \ldots, \mathcal{L}_{\mathbf{p},\alpha_h}$, where the k-th node stores $\mathcal{L}_{\mathbf{p},\alpha_k} = (\alpha_k, R_{\mathbf{p},\alpha_k})$. Since $\mathcal{L}_\mathbf{p}$ stores the same number of edges as $C^*_{\mathbf{p},\alpha_0}$, it does not incur much extra memory cost.

Using $\mathcal{L}_\mathbf{p}$, we can efficiently get the set of theme communities $\mathbb{T}_{\mathbf{p},\alpha} = \{T^1_{\mathbf{p},\alpha}, \ldots, T^m_{\mathbf{p},\alpha}\}$ in two steps. Denote by $E^*_{\mathbf{p},\alpha}$ the set of edges of $C^*_{\mathbf{p},\alpha}$, in the first step, we compute $E^*_{\mathbf{p},\alpha} = \bigcup_{\alpha_k > \alpha} R_{\mathbf{p},\alpha_k}$, and induce $C^*_{\mathbf{p},\alpha}$ from $E^*_{\mathbf{p},\alpha}$. In the second step, we obtain $\mathbb{T}_{\mathbf{p},\alpha}$ by finding the set of maximal connected subgraphs in $C^*_{\mathbf{p},\alpha}$.

Next, we introduce how to compute the cohesiveness of every theme community in $\mathbb{T}_{\mathbf{p},\alpha}$.

Theorem 4. *Given a theme community $T^i_{\mathbf{p},\alpha} \in \mathbb{T}_{\mathbf{p},\alpha}$, denote by γ the cohesiveness of $T^i_{\mathbf{p},\alpha}$, if α_k is the smallest cohesion threshold in $\mathcal{A}_\mathbf{p}$ such that $T^i_{\mathbf{p},\alpha} \not\subseteq C^*_{\mathbf{p},\alpha_k}$, then $\gamma = \alpha_k$.*

Proof. First, we prove $\gamma \leq \alpha_k$. By Definition 5, γ is the minimum cohesion of all edges in $T^i_{\mathbf{p},\alpha}$. Since $T^i_{\mathbf{p},\alpha} \not\subseteq C^*_{\mathbf{p},\alpha_k}$, it follows Definition 2 that $\gamma \leq \alpha_k$.

Second, we prove $\gamma \geq \alpha_k$. Since α_k is the smallest cohesion threshold in $\mathcal{A}_\mathbf{p}$ such that $T^i_{\mathbf{p},\alpha} \not\subseteq C^*_{\mathbf{p},\alpha_k}$, we have $T^i_{\mathbf{p},\alpha} \subseteq C^*_{\mathbf{p},\alpha_{k-1}}$, and thus $\gamma \geq \beta_{\mathbf{p},\alpha_{k-1}}$. Since $\alpha_k = \beta_{\mathbf{p},\alpha_{k-1}}$, $\gamma \geq \alpha_k$. The theorem follows. □

According to Theorem 4, $\mathcal{A}_\mathbf{p}$ is exactly the set of the cohesiveness of all theme communities in $G_\mathbf{p}$. To compute the cohesiveness of a theme community $T^i_{\mathbf{p},\alpha} \in \mathbb{T}_{\mathbf{p},\alpha}$, we simply use $\mathcal{L}_\mathbf{p}$ to find the smallest $\alpha_k \in \mathcal{A}_\mathbf{p}$ such that $T^i_{\mathbf{p},\alpha} \not\subseteq C^*_{\mathbf{p},\alpha_k}$.

Next, we introduce how to use the decomposition property of maximal $(\mathbf{p}, \alpha)$-truss to build a TC-Tree.

6.2 Theme Community Tree

A TC-Tree, denoted by $\mathcal{T}$, is an extension of a *set enumeration tree* (SE-Tree) [29] 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 $\prec$ on the items in S is assumed. Thus, any subset of S can be written as a sequence of items in the order of $\prec$.

Every node of a SE-Tree uniquely represents a subset of S. The root node represents empty set $\emptyset$. For subsets S_1 and S_2 of S, the node representing S_2 is the child of the node representing S_1, if $S_1 \subset S_2$, $|S_2 \setminus S_1| = 1$, and S_1 is a prefix of S_2 when S_1 and S_2 are written as sequences of items in order $\prec$. 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 $\mathbf{p}_i$, which is a subset of S. The item stored in n_i is denoted by s_{n_i}. We also store the decomposed maximal $(\mathbf{p}, \alpha)$-truss $\mathcal{L}_{\mathbf{p}_i}$ in n_i. To save memory, we omit the nodes n_j ($j \geq 1$) whose decomposed maximal $(\mathbf{p}, \alpha)$-trusses are $\mathcal{L}_{\mathbf{p}_j} = \emptyset$.

We can build a TC-Tree in a top-down manner efficiently. If $\mathcal{L}_{\mathbf{p}_j} = \emptyset$, we can prune the entire subtree rooted at n_j immediately. This is because, for node n_j and its descendant n_d, we have $\mathbf{p}_j \subset \mathbf{p}_d$. Since $\mathcal{L}_{\mathbf{p}_j} = \emptyset$, we can derive from Proposition 1 that $\mathcal{L}_{\mathbf{p}_d} = \emptyset$. As a result, all descendants of n_j can be immediately pruned.

Algorithm 4 gives the details of building a TC-Tree $\mathcal{T}$. Lines 2-5 generate the nodes at the first layer of $\mathcal{T}$. Since the theme networks induced by different items in S are independent, we can compute $\mathcal{L}_{\mathbf{p}_i}$ in parallel. Our implementation uses multiple threads for this step. Lines 6-12 iteratively build the rest of the nodes of $\mathcal{T}$ in breadth first order. Here, $n_f.siblings$ is the set of nodes that have the same parent as n_f. The children of n_f, denoted by n_c, are built in Lines 8-11. In Line 9, we apply Proposition 2 to efficiently calculate $\mathcal{L}_{\mathbf{p}_c}$. Since $\mathbf{p}_c = \mathbf{p}_f \cup \mathbf{p}_b$, we have $\mathbf{p}_f \subset \mathbf{p}_c$ and $\mathbf{p}_b \subset \mathbf{p}_c$. From Proposition 2, we know $C^*_{\mathbf{p}_c,\alpha_0} \subseteq C^*_{\mathbf{p}_f,\alpha_0} \cap C^*_{\mathbf{p}_b,\alpha_0}$. Therefore, we can find $C^*_{\mathbf{p}_c,\alpha_0}$ within a small subgraph $C^*_{\mathbf{p}_f,\alpha_0} \cap C^*_{\mathbf{p}_b,\alpha_0}$ using MTD, and then get $\mathcal{L}_{\mathbf{p}_c}$ by decomposing $C^*_{\mathbf{p}_c,\alpha_0}$.

In summary, every node of a TC-Tree stores the decomposed maximal $(\mathbf{p}, \alpha)$-truss $\mathcal{L}_\mathbf{p}$ of a unique pattern $\mathbf{p} \subseteq S$. Next, we introduce how to efficiently query a TC-Tree.

6.3 Querying and Query Recommendation

In this subsection, we first introduce how to query a TC-Tree $\mathcal{T}$ by a **query pattern q**, then we illustrate how to recommend new queries based on a user-provided query pattern $\mathbf{q}$.

When querying a TC-Tree $\mathcal{T}$ by a query pattern $\mathbf{q}$, the *answer* to the query, denoted by $\mathcal{R}_\mathbf{q}$, is a ranked list of all the theme communities in $G_\mathbf{q}$, that is, $\cup_{\alpha_i \in \mathcal{A}_\mathbf{p}} \mathbb{T}_{\mathbf{q},\alpha_i}$.

We obtain $\mathcal{R}_\mathbf{q}$ in the following steps: First, we find the node n_i in $\mathcal{T}$ such that the pattern of n_i is $\mathbf{p}_i = \mathbf{q}$. Second, we obtain the set of theme communities in $G_\mathbf{q}$ using the $\mathcal{L}_\mathbf{q}$ stored in n_i. Last, we obtain $\mathcal{R}_\mathbf{q}$ by sorting the theme communities in the descending order of their cohesiveness.

From time to time, a user-provided query pattern $\mathbf{q}$ may not lead to any answer. The cohesiveness and size of the retrieved theme communities may be too small or the answer to the query can even be $\mathcal{R}_\mathbf{q} = \emptyset$. In such a case, we

can explore the TC-Tree to recommend a ranked list of new query patterns, denoted by $\mathcal{U} = \mathbf{q}_1, \ldots, \mathbf{q}_k$.

Those recommended query patterns in $\mathcal{U}$ should satisfy three conditions. First, querying $\mathcal{T}$ by any query pattern $\mathbf{q}_i \in \mathcal{U}$ leads to a non-empty set of theme communities, that is, $\mathcal{L}_{\mathbf{q}_i} \neq \emptyset$. Second, the recommended query patterns should be contained in $\mathbf{q}$, because querying $\mathcal{T}$ by a pattern containing $\mathbf{q}$ only retrieves those theme communities with smaller cohesiveness and size. Third, every query pattern $\mathbf{q}_i \in \mathcal{U}$ should be the most similar to the original query pattern $\mathbf{q}$, that is, among all query patterns that satisfy the previous two conditions, $\mathbf{q}_i$ has the minimum size of set difference $|\mathbf{q} \setminus \mathbf{q}_i|$.

According to the third condition, the set differences of $\mathbf{q}$ and every query pattern in $\mathcal{U}$ must have the same size. Since a query pattern $\mathbf{q}$ contains at most $\binom{|\mathbf{q}|}{m}$ patterns that have the same set difference size m with respect to $\mathbf{q}$, the number of query patterns in $\mathcal{U}$ is at most $\binom{|\mathbf{q}|}{\lfloor |\mathbf{q}|/2 \rfloor}$. In practice, since many query patterns contained in $\mathbf{q}$ retrieve no theme community from $\mathcal{T}$, the actual volume of $\mathcal{U}$ is very small.

Now, we illustrate how to rank all the recommended query patterns in $\mathcal{U}$. For each query pattern $\mathbf{q}_i \in \mathcal{U}$, we first access the last entry of $\mathcal{L}_{\mathbf{q}_i}$ to obtain $\alpha^*_{\mathbf{q}_i}$, which is the maximum cohesiveness of all theme communities in $G_{\mathbf{q}_i}$. Then, we rank the new query patterns in $\mathcal{U}$ in the descending order of $\alpha^*_{\mathbf{q}_i}$, so that the user can first explore the query patterns that retrieve theme communities with large cohesiveness.

As shown in Algorithm 5, the query recommendation method simply traverses the TC-Tree in the breadth first manner and collects the set of new queries that satisfy the above conditions for the new query patterns in $\mathcal{U}$.

In summary, TC-Tree enables fast user query answering and efficient query recommendation. As demonstrated by the case study in Section 7.1 and the experiments in Section 7.4, TC-Tree is efficient to build, easy to query, and scales well to index a large number of theme communities using practical size of memory.

Algorithm 4: Build Theme Community Tree

Input: A DBN G.
Output: The TC-Tree $\mathcal{T}$ with root node n_0.

1: Initialization: $Q \leftarrow \emptyset$, $s_{n_0} \leftarrow \emptyset$, $\mathcal{L}_{\mathbf{p}_0} \leftarrow \emptyset$.
2: **for** each item $s_i \in S$ **do**
3: $s_{n_i} \leftarrow s_i$, $\mathbf{p}_i \leftarrow s_i$ and compute $\mathcal{L}_{\mathbf{p}_i}$.
4: **if** $\mathcal{L}_{\mathbf{p}_i} \neq \emptyset$ **then** $n_0.addChild(n_i)$ and $Q.push(n_i)$.
5: **end for**
6: **while** $Q \neq \emptyset$ **do**
7: $n_f \leftarrow Q.pop()$.
8: **for** each node $n_b \in n_f.siblings$ **do**
9: **if** $s_{n_f} \prec s_{n_b}$ **then** $s_{n_c} \leftarrow s_{n_b}$, $\mathbf{p}_c \leftarrow \mathbf{p}_f \cup \mathbf{p}_b$, and compute $\mathcal{L}_{\mathbf{p}_c}$.
10: **if** $\mathcal{L}_{\mathbf{p}_c} \neq \emptyset$ **then** $n_f.addChild(n_c)$ and $Q.push(n_c)$.
11: **end for**
12: **end while**
13: **return** The TC-Tree $\mathcal{T}$ with root node n_0.

Algorithm 5: Query Recommendation

Input: A TC-Tree $\mathcal{T}$ and a query $\mathbf{q}$.
Output: A ranked list $\mathcal{U} = \mathbf{q}_1, \ldots, \mathbf{q}_k$.

1: Initialization: $Q \leftarrow n_0$, $\mathcal{U} \leftarrow \emptyset$.
2: **while** $Q \neq \emptyset$ **do**
3: $n_f \leftarrow Q.pop()$.
4: **for** each node $n_c \in n_f.children \wedge s_{n_c} \in \mathbf{q}$ **do**
5: **if** $\mathcal{L}_{\mathbf{p}_c} \neq \emptyset$ **then** $\mathcal{U} \leftarrow \mathcal{U} \cup \mathbf{p}_c$; $Q.push(n_c)$.
6: **end for**
7: **end while**
8: Keep each query pattern $\mathbf{q}_i \in \mathcal{U}$ that has the smallest size of set difference $|\mathbf{q} \setminus \mathbf{q}_i|$, and remove the others.
9: Rank all patterns in $\mathcal{U}$ by $\alpha^*_{\mathbf{q}_i}$.
10: **return** The ranked list $\mathcal{U}$.

7. EXPERIMENTS

In this section, we first present a case study to demonstrate how theme community finding is useful. Then, we comprehensively evaluate the performance of Theme Community Scanner (TCS), Theme Community Finder Apriori (TCFA), Theme community Finder Intersection (TCFI) and Theme Community Tree (TC-Tree). Last, we compare the theme community detection performance of TCFI and that of two vertex attributed network methods, CESNA [37] and SCI [34].

For each of our proposed methods, the final output is the set of theme communities obtained by finding the maximal connected subgraphs in the detected maximal $(\mathbf{p}, \alpha)$-trusses. We use the absolute frequency for TC-Tree. However, since the absolute frequency is unnormalized, which makes it difficult to set the frequency threshold ϵ for TCS, we adopt the relative frequency for TCS, and also use the relative frequency for TCFA and TCFI to fairly compare with TCS. Since TC-Tree is an indexing method, it is not directly comparable with TCS, TCFA and TCFI.

We implement TCS, TCFA and TCFI in Java. In order to efficiently index the theme communities in large DBNs, we implement TC-Tree in C++ and parallelize the steps in Lines 2-5 of Algorithm 4 with 4 threads using OpenMP. The source code of CESNA and SCI was provided by their authors. All experiments are performed on a Windows 7 PC with Core-i7 CPU, 32GB RAM and a 5400 rpm hard drive.

The following 3 data sets are used.

The **Brightkite (BK)** data set is a public check-in data set 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; every user check-in contains the check-in time and location. We construct a DBN using this data set by taking the user friendship network as the network of the DBN. 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 a transaction. A theme community in this DBN is a group of friends who frequently visit the same set of places.

The **Gowalla (GW)** data set is a public data set 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 data set into a DBN in the same way as BK.

The **AMINER** data set is built from the Citation network v2 (CNV2) data set [2]. CNV2 contains 1,397,240 papers. We transform it into a DBN 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 keyword in the abstract of a paper as an item, and all the keywords in the abstract of a paper are turned into a transaction. An author vertex is associated with a vertex database of all papers the author publishes. In this DBN, a theme community represents a group of authors who collaborate closely and share the same research interest that is described by the same set of keywords.

The statistics of all data sets are given in Table 1.

Table 1: Statistics of the DBNs. #Trans. is the number of transactions. #Items is the number of items stored in all vertex databases.

Data sets	#Vertices	#Edges	#Trans.	#Items
BK	5.1×10^4	2.1×10^5	1.2×10^6	1.7×10^6
GW	1.1×10^5	9.5×10^5	2.0×10^6	3.5×10^6
AMINER	1.1×10^6	2.6×10^6	3.1×10^6	9.2×10^6

	Query Pattern	α^*
a_1	Data mining, face recognition, clustering algorithm, image retrieval, principal component analysis, gene expression, linear discriminant analysis, hierarchical clustering, dimensionality reduction	0
b_1	Data mining, principal component analysis, gene expression, dimensionality reduction	7
b_2	Image retrieval, principal component analysis, linear discriminant analysis, dimensionality reduction	3
b_3	Clustering algorithm, gene expression, linear discriminant analysis, hierarchical clustering	3
b_4	Face recognition, principal component analysis, linear discriminant analysis, dimensionality reduction	3
b_5	Data mining, principal component analysis, hierarchical clustering, dimensionality reduction	2
b_6	Clustering algorithm, gene expression, hierarchical clustering, dimensionality reduction	2
b_7	Data mining, principal component analysis, linear discriminant analysis, dimensionality reduction	1
b_8	Face recognition, image retrieval, principal component analysis, linear discriminant analysis	1
c_1	Face Recognition, principal component analysis, linear discriminant analysis	6
c_2	Image retrieval, principal component analysis, linear discriminant analysis	3

Figure 3: The query patterns for the theme communities in Figure 4 and Figure 5. $\mathbf{a}_1$ is the user provided query pattern. $\{\mathbf{b}_1,\ldots,\mathbf{b}_8\}$ are the recommended query patterns based on $\mathbf{a}_1$. $\{\mathbf{c}_1,\mathbf{c}_2\}$ are the top-1 and top-2 recommended query patterns for $\mathbf{b}_8$, respectively. α^* is the maximum cohesiveness of all theme communities induced by a query pattern.

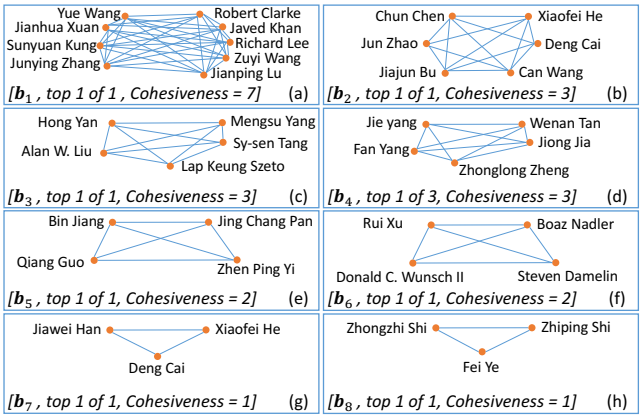


Figure 4: The top-1 theme communities retrieved by the query patterns $\{\mathbf{b}_1,\ldots,\mathbf{b}_8\}$ in Figure 3.

Figure 5: The top-1 and top-2 theme communities retrieved by the query patterns $\mathbf{c}_1$ and $\mathbf{c}_2$ in Figure 3.

7.1 A Case Study

In this subsection, we demonstrate the query recommendation and query answering process of the TC-Tree that indexes all the theme communities in the DBN of AMINER. Each theme community retrieved by a query pattern represents a group of co-working scholars who share the same research interest characterized by the set of keywords contained in the query pattern.

We start our case study with a user who wants to find theme communities that apply the classic methods, such as "hierarchical clustering" (HC), "principal component analysis" (PCA), "linear discriminant analysis" (LDA) and "dimensionality reduction" (DR), in the research areas of "data mining" (DM), "face recognition" (FR), "image retrieval" (IR) and "gene expression" (GE).

Since the user may have little prior knowledge about the theme communities in the DBN, it is difficult for him to design an appropriate query that successfully retrieves valid theme communities. Therefore, he simply put all his interested key words together and starts querying the TC-Tree using the query pattern $\mathbf{a}_1$ in Figure 3.

Unfortunately, no theme community in the DBN of AMINER is induced by $\mathbf{a}_1$, therefore querying the TC-Tree by $\mathbf{a}_1$ returns no theme community. In this case, we use the query recommendation method in Algorithm 5 to recommend to the user a ranked list of query patterns $\{\mathbf{b}_1,\ldots,\mathbf{b}_8\}$ in Figure 3. Each of $\{\mathbf{b}_1,\ldots,\mathbf{b}_8\}$ is a meaningful combination of the user-interested keywords that describes an interesting research direction. For example, $\mathbf{b}_1$ characterizes the research direction that applies PCA and DR in the area of DM and GE, and $\mathbf{b}_2$ is the research direction that applies PCA, LDA and DR in the area of IR.

Figure 4 shows the top-1 theme communities retrieved by the recommended query patterns in $\{\mathbf{b}_1,\ldots,\mathbf{b}_8\}$. Most of the top-1 theme communities are cliques, because a clique is a special case of $(\mathbf{p},\alpha)$-truss, and the dense connection between the vertices of a clique produces a high cohesiveness. Interestingly, the theme community in Figure 4(b) is not a clique. This demonstrates the flexibility of the proposed $(\mathbf{p},\alpha)$-truss in detecting communities that are not cliques but still have large cohesiveness.

We can also see that the theme communities in Figures 4(b) and 4(g) both contain "Xiaofei He" and "Deng Cai". This shows that the proposed theme community finding allows arbitrary overlap between communities of different themes.

As shown in Figure 4(h), the query pattern $\mathbf{b}_8$ retrieves only one theme community that is small in both size and cohesiveness. This is because FR and IR are separate subdisciplines in the research field of computer vision, and it is rare for a paper to focus on both of them at the same time.

If the user is not happy with the small theme community retrieved by $\mathbf{b}_8$, we can further apply Algorithm 5 to recommend to the user the query patterns $\mathbf{c}_1$ and $\mathbf{c}_2$ in Figure 3. Obviously, $\mathbf{c}_1$ describes the sub-discipline that applies PCA and LDA in FR, and $\mathbf{c}_2$ characterizes the sub-discipline that uses PCA and LDA in IR.

Figure 5 shows the theme communities retrieved by $\mathbf{c}_1$ and $\mathbf{c}_2$. Since $\mathbf{c}_1$ and $\mathbf{c}_2$ characterize the two sub-disciplines more precisely than $\mathbf{b}_8$, $\mathbf{c}_1$ and $\mathbf{c}_2$ retrieve more theme communities than $\mathbf{b}_8$, and the cohesiveness of the theme communities retrieved by $\mathbf{c}_1$ and $\mathbf{c}_2$ are much larger than the one retrieved by $\mathbf{b}_8$.

In summary, the proposed theme community finding discovers meaningful theme communities and allows arbitrary

overlap between theme communities. The proposed query recommendation method effectively recommends meaningful query patterns, and the proposed TC-Tree makes it possible to efficiently retrieve ranked lists theme communities from large DBNs.

7.2 Effect of Parameters

In this subsection, we analyze the effect of the cohesion threshold α and the frequency threshold ϵ. 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 cannot 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 DBNs of BK, GW and AMINER, we use smaller DBNs that are sampled from the original DBNs by doing a breadth first search from a random seed vertex. From each of BK and GW, we sample one DBN with 10,000 edges. For AMINER, we sample a DBN of 5,000 edges.

Figures 6(a), 6(c) and 6(e) show the **number of theme communities**, denoted by "#TCs", found by TCFA, TCFI and TCS from BK, GW and AMINER, respectively. As it is shown, TCFA and TCFI detect the same number of theme communities for all values of α on all DBNs. This is because both TCFA and TCFI produce the exact result in finding all theme communities in a DBN. However, TCS does not always produce the exact result. The reason is that vertices with small pattern frequencies can still form a good theme community with large edge cohesion if they form a densely connected subgraph. Such theme communities may be missed if the patterns with low frequencies are dropped by the pre-filtering step of TCS.

Figures 6(b), 6(d) and 6(f) show the time cost of TCFA, TCFI and TCS on BK, GW and AMINER, respectively. The time cost of both TCFA and TCFI decreases when α increases, because increasing α reduces the size of $\mathcal{P}^{k-1}$, which further reduces the size of $\mathcal{M}^k$.

When α is small, the time cost of TCFI is much lower than the time cost of TCFA. This is due to the following two effects of applying the graph intersection property in Proposition 2. First, most maximal $(\mathbf{p}, \alpha)$-trusses are small local subgraphs that do not intersect with each other, thus many unqualified patterns in $\mathcal{M}^k$ are pruned by TCFI using the graph intersection property. Second, for each call of MTD, TCFA computes the maximal $(\mathbf{p}, \alpha)$-truss in the large theme network induced from the entire DBN, however, TCFI operates on the small theme network induced from the intersection of two maximal $(\mathbf{p}, \alpha)$-trusses.

When α is large, the time cost of TCFI and TCFA becomes comparable. This is because increasing α reduces the size of $\mathcal{M}^k$ and the size of the maximal $(\mathbf{p}, \alpha)$-truss for each pattern in $\mathcal{M}^k$. Therefore, the effectiveness of applying the graph intersection property is reduced.

Take the sampled DBN of AMINER as an example. When $\alpha = 0$, TCFA calls MTD 622,852 times and TCFI calls MTD 152,396 times. TCFI effectively prunes 75.5% of the candidate patterns used by TCFA. Moreover, as shown in Figure 6(f), TCFI is nearly 3 orders of magnitude faster than TCFA when $\alpha = 0$. This is because TCFI always operates on small theme networks. We can also see that, when $\alpha \geq 1$, the time cost of TCFI and TCFA becomes comparable. This is because, when $\alpha \geq 1$, AMINER contains only one maximal $(\mathbf{p}, \alpha)$-truss, thus TCFI cannot prune any candidate pattern in $\mathcal{M}^k$ or induce any smaller theme network by applying the graph intersection property.

The time cost of TCS is not affected by α, because it is largely dominated by the size of $\mathcal{P}$ (see Section 4.2), which

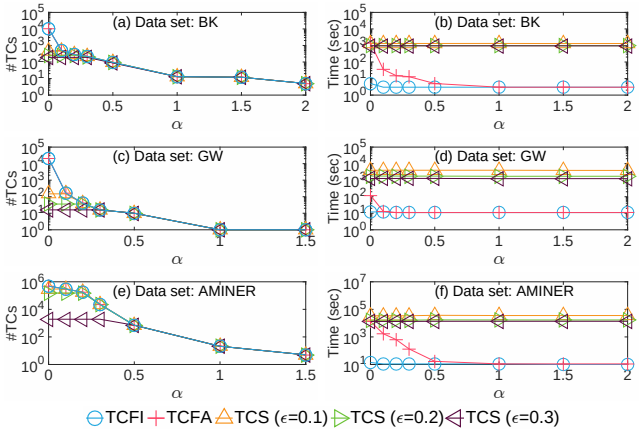


Figure 6: The effects of parameters α and ϵ. In (c) and (e), #TCs is zero when $\alpha = 2.0$.

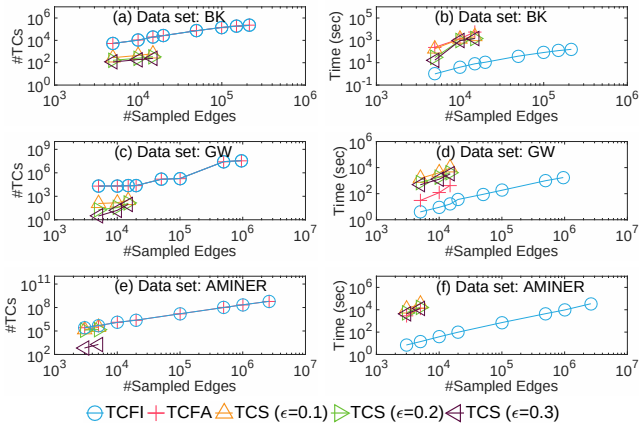


Figure 7: The #TCs and the time cost of TCS, TCFA and TCFI on different sizes of networks.

is irrelevant to α. Increasing ϵ reduces the size of $\mathcal{P}$ and improves the efficiency of TCS. However, since the size of $\mathcal{P}$ is still too large, TCS is orders of magnitude slower than TCFI on all DBNs.

In summary, TCFI produces the best detection results of theme communities and achieves the best efficiency performance for all values of α on all DBNs.

7.3 Scalability of Theme Community Finding

In this subsection, we analyze the runtime of all methods with respect to the size of the DBNs. For each DBN, we generate a series of DBNs with different sizes by sampling the original DBN using the sampling method introduced in Section 7.2. Since TCS and TCFA run too slow on large DBNs, 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 DBNs including the original ones. To evaluate the worst case performance of all methods, we set $\alpha = 0$.

Figures 7(a), 7(c) and 7(e) show the number of theme communities found by TCFA, TCFI and TCS from BK, GW and AMINER, respectively. When the number of sampled edges increases, the numbers of theme communities reported by all methods increase. Increasing the size of the DBN increases the number of maximal $(\mathbf{p}, \alpha)$-trusses, which further increases the number of theme communities. Both TCFI

Table 2: The performance of indexing of TC-Tree on the full data sets of BK, GW and AMINER.

	Indexing Time	Memory	#Nodes
BK	215 seconds	0.35 GB	18,581
GW	1,812 seconds	2.66 GB	11,750,761
AMINER	41,958 seconds	22.84 GB	122,337,700

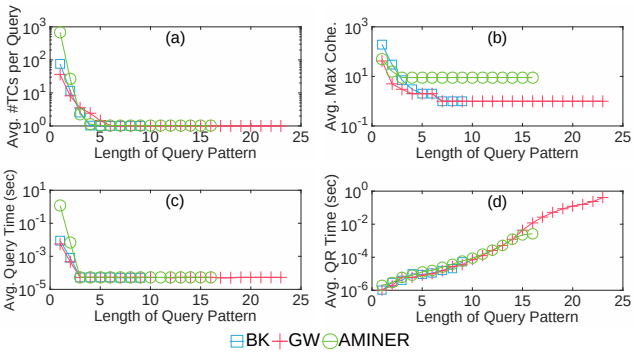


Figure 8: The effect of the length of query pattern. "Avg.", "Cohe." and "QR" are abbreviations for "Average", "Cohesiveness" and "Query Recommendation", respectively.

and TCFA consistently produce the exact results. Due to the accuracy loss caused by pre-filtering the patterns with low frequencies, TCS cannot produce the same results as TCFI and TCFA.

Figures 7(b), 7(d) and 7(f) show the performance of time cost. When the number of sampled edges increases, the time cost of all methods increases, because increasing the size of the DBN increases the number of theme communities. The time cost of TCS and TCFA grows much faster than that of TCFI. This is because TCS generates a large number of unqualified candidate patterns by enumerating the patterns of all vertex databases, and TCFA also generates many unqualified candidate patterns by taking the pairwise unions of the patterns of the detected maximal $(\mathbf{p}, \alpha)$-trusses. By applying the graph intersection property, TCFI efficiently generates a substantially smaller number of candidate patterns by the pairwise unions of the patterns of two intersecting maximal $(\mathbf{p}, \alpha)$-trusses, and only runs MTD on the small intersection of two maximal $(\mathbf{p}, \alpha)$-trusses. This 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 DBNs.

7.4 Performance of Theme Community Tree

In this subsection, we analyze the performance of Theme Community Tree (TC-Tree), including the indexing efficiency of TC-Tree, the effect of the length of query pattern and the distribution of retrieved theme communities.

The indexing efficiency of TC-Tree in all DBNs is shown in Table 2. "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.

Building a TC-Tree is efficient in both Indexing Time and Memory. For the largest DBN AMINER, TC-Tree scales up well and indexes more than 120 million nodes.

To analyze the effect of the length of query pattern on the querying performance, we generate a set of query patterns $\mathcal{B}$ by randomly sampling 1,000 nodes from the TC-Tree and using the patterns of the sampled nodes as query patterns. Denote by $\mathcal{B}^l = \{\mathbf{q} \mid \mathbf{q} \in \mathcal{B}, |\mathbf{q}| = l\}$ the set of length-l query patterns in $\mathcal{B}$, and by L the maximum length of all query patterns in $\mathcal{B}$, we have $\mathcal{B} = \mathcal{B}^1 \cup \cdots \cup \mathcal{B}^L$. We use the query patterns in $\mathcal{B}$ to query the TC-Tree and analyze the following querying performance.

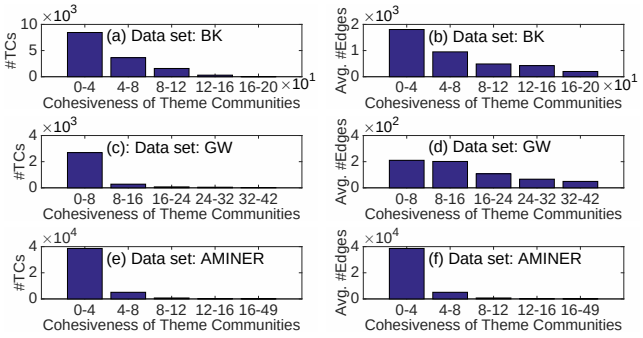


Figure 9: The distribution of retrieved theme communities. Each bin on the x-axis represents an interval of the cohesiveness of theme communities. "#TCs" is the number of theme communities in a bin. "Avg. #Edges" is the average number of edges of all theme communities in a bin.

The "Avg. #TCs per Query" in Figure 8(a) is the average number of theme communities retrieved by the query patterns in each set of $\mathcal{B}^l, l \in \{1, \ldots, L\}$. In Figure 8(a), the Avg. #TCs per Query decreases quickly when the length of query pattern increases, this verifies our analysis in Section 3.3 that a longer pattern is less likely to induce a theme community. For query patterns with length larger than 1, the average number of retrieved theme communities is less than 100. It is not a heavy burden for users to analyze such a small number of theme communities.

In Figure 8(b), the "Avg. Max Cohe." is the average cohesiveness of the top-1 theme communities retrieved by the query patterns in each set of $\mathcal{B}^l, l \in \{1, \ldots, L\}$. The Avg. Max Cohe. decreases when the length of query pattern increases, because a longer pattern has a lower frequency, which limits the cohesiveness of the corresponding theme communities.

Figure 8(c) shows the average query time for the query patterns in each set of $\mathcal{B}^l, l \in \{1, \ldots, L\}$. For all DBNs, the average query time is less than 1 second when the length of pattern is 1, and decreases quickly to less than 0.1 second when the length of query pattern increases. The query time is dominated by the time to compute the theme communities and their cohesiveness using the decomposed maximal $(\mathbf{p}, \alpha)$-truss stored in a node of the TC-Tree. When the length of query pattern increases, the size of the maximal $(\mathbf{p}, \alpha)$-truss reduces, thus the query time decreases.

Figure 8(d) shows the average query recommendation time for the query patterns in each set of $\mathcal{B}^l, l \in \{1, \ldots, L\}$. When the length of query pattern increases, the average query recommendation time increases, because a longer query pattern requires Algorithm 5 to visit more TC-Tree nodes. The query recommendation is highly efficient, and the time cost is less than 1 second on all DBNs.

Next, we query the TC-Tree by the query patterns in $\mathcal{B}$, and show the distribution of all retrieved theme communities in Figure 9. A large proportion of the retrieved theme communities have very small cohesiveness, and contain a larger number of edges. These theme communities are trivially induced by the patterns that are contained in a large number of vertex databases with low pattern frequencies, and they have low ranks in the retrieved ranked list of theme communities due to their small cohesiveness.

There are a lot of theme communities that have large cohesiveness and contain a small number of edges. Those theme communities are usually induced by the patterns that frequently appear in the vertex databases of a small number of strongly connected vertices. As demonstrated by the case study in Section 7.1, these theme communities are ranked high in the retrieved ranked list and often reveal interesting patterns and communities in DBNs.

7.5 Comparison with Two Vertex Attributed Network Methods

In this subsection, we evaluate the theme community detection performance of TCFI and two community detection methods for Vertex Attributed Networks (VAN), such as CESNA [37] and SCI [34]. We set $\alpha = 0$ for TCFI, and use the default settings of CESNA and SCI, respectively. Since SCI cannot efficiently process large networks, we sample one DBN with 10,000 edges from each of BK, GW and AMINER using the sampling method in Section 7.2. Both CESNA and SCI cannot directly process a DBN, thus we extend them by first converting a DBN into a VAN, then applying them to detect communities from the VAN.

We adopt the following three types of **extensions**.

Extension A: we convert each vertex database into a set of items by taking the union of all transactions in it. The pattern of a detected community is a set of items, which is used as the theme of the community.

Extension B: we regard a transaction as a set-valued item, and treat each vertex database as a set of set-valued items. The pattern of a detected community is a set of transactions, where each transaction is used as a theme of the community.

Extension C: we convert a DBN in the same way as Extension B. Instead of using each transaction as a theme, we treat the union of the set of transactions as the theme of the detected community.

Denote by D a detected community with a theme $\mathbf{p}$. We measure the quality of D by the following metrics.

Average Edge Strength (AES): denote by $\mathcal{E}$ and $\mathcal{V}$ the sets of edges and vertices of D, respectively. The **edge strength** of an edge $e_{ij} \in \mathcal{E}$ is measured by $ES_{ij} = f_i(\mathbf{p}) \times f_j(\mathbf{p})$. The AES of D is

$$AES = \frac{2\sum_{e_{ij}\in\mathcal{E}} ES_{ij}}{|\mathcal{V}|(|\mathcal{V}|-1)},$$

which is the ratio between the sum of edge strength of all edges in $\mathcal{E}$ and the number of edges of a clique containing a number of $|\mathcal{V}|$ vertices. A large AES means the vertices in $\mathcal{V}$ have a high frequency of $\mathbf{p}$, and are densely connected by the edges in $\mathcal{E}$. Obviously, a larger AES indicates a higher quality of the theme community.

Cohesiveness (COHE): as defined in Definition 5, the cohesiveness of D is the minimum cohesion of its edges. If COHE is 0, then either at least one vertex of D does not contain $\mathbf{p}$ in its vertex database, or at least one edge of D is not contained in any triangle. Thus, we say D is an **invalid** theme community if its COHE is 0, and say D is **valid** if its COHE is positive. Obviously, a larger COHE indicates a higher quality of D.

Both AES and COHE are computed based on the absolute frequency of the theme $\mathbf{p}$.

Figure 10 shows the Average (Avg.) COHE and the Avg. AES of the top-k detected theme communities with the largest COHE and AES, respectively. Both CESNA and SCI do not achieve a good performance of Avg. COHE or Avg. AES for all types of extensions, because all the extensions lose the valuable information about item co-occurrences and pattern frequencies when converting a vertex database into a set of items. Since TCFI effectively explores the item co-occurrence and pattern frequency of DBNs, it achieves the best performance on all data sets.

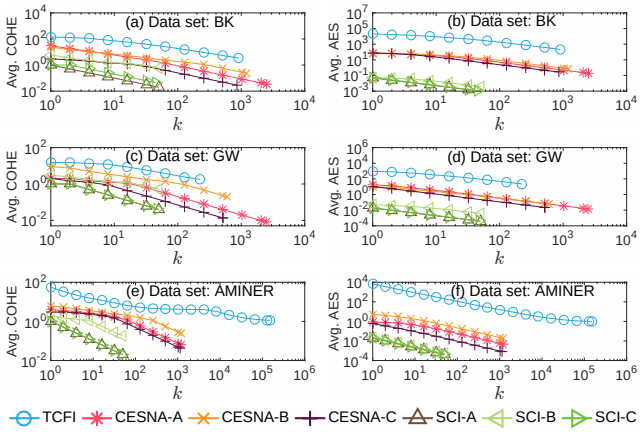


Figure 10: The Average (Avg.) COHE and Avg. AES of top-k detected communities.

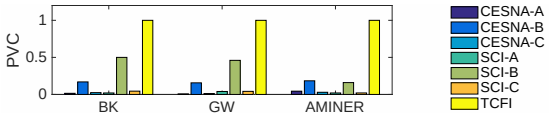


Figure 11: The PVC performance.

Figure 11 shows the **Proportion of Valid Community (PVC)** of all compared methods. Here, PVC is the proportion of valid theme communities in the set of all detected communities. The PVC of CESNA and SCI is low for Extensions A and C, because both of them take the union of transactions, which generates a lot of new patterns that do not exist in the DBN. These new patterns induce many detected communities, which are invalid in the DBN. The extension B does not take the union of transactions, thus it achieves a higher PVC than Extensions A and C. However, since CESNA and SCI do not strictly require every vertex in the same community to have exactly the same pattern, nor do they require every edge of a community to be contained in one or more triangles, many communities detected by CESNA-B and SCI-B are invalid, thus their PVC is still much lower than TCFI. The PVC of TCFI is always 1.0, which demonstrates the superior performance of TCFI in accurately finding valid theme communities.

In sum, a DBN naturally models the rich and valuable vertex information, such as item co-occurrence and pattern frequency, which is way beyond the limited descriptive power of VANs. As a result, it is difficult to straightforwardly extend VAN-based methods, such as CESNA and SCI, to effectively enumerate the theme communities in DBNs.

8. CONCLUSIONS AND FUTURE WORK

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

9. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *PVLDB*, 1215:487–499, 1994.

[2] AMINER. `https://aminer.org/citation`, 2010.

[3] A. F. A. Assal, R. M. M. Lim, and M.-H. L. Lee. Multi-user on-line real-time virtual social networks based upon communities of interest for entertainment, information or e-commerce purposes, Dec. 2010. US Patent 7,853,881.

[4] R. Balasubramanyan and W. W. Cohen. Block-LDA: Jointly modeling entity-annotated text and entity-entity links. In *Proceedings of the SIAM International Conference on Data Mining*, volume 11, pages 450–461, 2011.

[5] M. Berlingerio, F. Pinelli, and F. Calabrese. ABACUS: frequent pattern mining-based community discovery in multidimensional networks. *Data Mining and Knowledge Discovery*, 27(3):294–320, 2013.

[6] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1082–1090, 2011.

[7] L. Chu, S. Wang, S. Liu, Q. Huang, and J. Pei. ALID: Scalable dominant cluster detection. *PVLDB*, 8(8):826–837, 2015.

[8] J. Cohen. Barycentric graph clustering. *Oregon Health Science University*, 2008.

[9] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, page 16, 2008.

[10] J. Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29–41, 2009.

[11] D. Combe, C. Largeron, E. Egyed-Zsigmond, and M. Géry. Combining relations and text in scientific network clustering. In *International Conference on Advances in Social Networks Analysis and Mining*, pages 1248–1253, 2012.

[12] J. D. Cruz, C. Bothorel, and F. Poulet. Semantic clustering of social networks using points of view. In *CORIA*, pages 175–182, 2011.

[13] T. Dang and E. Viennet. Community detection based on structural and attribute similarities. In *International Conference on Digital Society*, pages 7–12, 2012.

[14] S. Günnemann, B. Boden, and T. Seidl. DB-CSC: a density-based approach for subspace clustering in graphs with feature vectors. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 565–580, 2011.

[15] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2):140–174, 2003.

[16] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 29, pages 1–12, 2000.

[17] A. L. Hu and K. C. Chan. Utilizing both topological and attribute information for protein complex identification in ppi networks. *IEEE Transactions on Computational Biology and Bioinformatics*, 10(3):780–792, 2013.

[18] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1311–1322, 2014.

[19] X. Huang and L. V. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, 2017.

[20] Y. Huang, Y. Tang, C. Li, Z. Wu, and H. Dong. A method for latent-friendship recommendation based on community detection in social network. In *Web Information System and Application Conference*, pages 3–8. IEEE, 2015.

[21] Z. Huang and M. Benyoucef. From e-commerce to social commerce: A close look at design features. *Electronic Commerce Research and Applications*, 12(4):246–259, 2013.

[22] R. D. Luce. Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, 15(2):169–190, 1950.

[23] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are who you know: inferring user profiles in online social networks. In *International Conference on Web Search and Data Mining*, pages 251–260. ACM, 2010.

[24] S. A. Moosavi, M. Jalali, N. Misaghian, S. Shamshirband, and M. H. Anisi. Community detection in social networks using user frequent pattern mining. *Knowledge and Information Systems*, pages 1–28, 2016.

[25] F. Moser, R. Colak, A. Rafiey, and M. Ester. Mining cohesive patterns from graphs with feature vectors. In *Proceedings of the SIAM International Conference on Data Mining*, volume 9, pages 593–604, 2009.

[26] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2002.

[27] M. Pavan and M. Pelillo. Dominant sets and pairwise clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):167–172, 2007.

[28] A. Prado, M. Plantevit, C. Robardet, and J.-F. Boulicaut. Mining graph topological patterns: Finding covariations among vertex descriptors. *IEEE Transactions on Knowledge and Data Engineering*, 25(9):2090–2104, 2013.

[29] R. Rymon. Search through systematic set enumeration. *Technical Reports (CIS)*, page 297, 1992.

[30] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[31] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[32] K. Steinhaeuser and N. V. Chawla. Community detection in a large real-world social network. In *Social computing, behavioral modeling, and prediction*, pages 168–175. Springer, 2008.

[33] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[34] X. Wang, D. Jin, X. Cao, L. Yang, and W. Zhang. Semantic community identification in large attribute networks. In *the AAAI Conference on Artificial Intelligence*, 2016.

[35] H. Xin, L. Wei, and L. V. S. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 77–90, 2016.

[36] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213,

2015.

[37] J. Yang, J. McAuley, and J. Leskovec. Community detection in networks with node attributes. In *International Conference on Data Mining*, pages 1151–1156, 2013.

[38] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.