Mining Frequent Neighborhood Patterns in a Large Labeled Graph *

Jialong Han
Renmin University of China
jialonghan@gmail.com

Ji-Rong Wen
Renmin University of China
jirong.wen@gmail.com

ABSTRACT

Over the years, frequent subgraphs have been an important kind of targeted pattern in pattern mining research, where most approaches deal with databases holding a number of graph *transactions*, e.g., the chemical structures of compounds. These methods rely heavily on the *downward-closure property* (DCP) of the support measure to ensure an efficient pruning of the candidate patterns. When switching to the emerging scenario of single-graph databases such as Google's Knowledge Graph and Facebook's social graph, the traditional support measure turns out to be trivial (either 0 or 1). However, to the best of our knowledge, all attempts to redefine a single-graph support have resulted in measures that either lose DCP, or are no longer semantically intuitive.

This paper targets pattern mining in the single-graph setting.

We propose mining a new class of patterns called frequent neighborhood patterns, which is free from the "DCP-intuitiveness" dilemma of mining frequent subgraphs in a single graph. A neighborhood is a specific topological pattern in which a vertex is embedded, and the pattern is frequent if it is shared by a large portion (above a given threshold) of vertices. We show that the new patterns not only maintain DCP, but also have equally significant interpretations as subgraph patterns. Experiments on real-life datasets support the feasibility of our algorithms on relatively large graphs, as well as the capability of mining interesting knowledge that is not discovered by prior methods.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

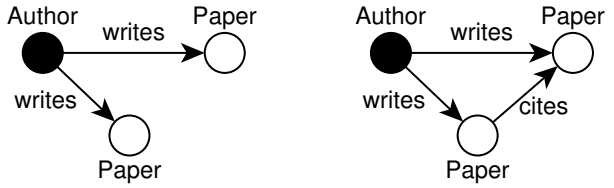
Keywords

Pattern Mining, Frequent Neighborhood Mining

*The work was done when the first author was visiting Microsoft Research Asia.

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2505515.2505530.



(a) 47.0% of authors have at least two papers.

(b) 9.1% of authors once cited their own paper.

Figure 1: Neighborhood patterns with support ratios, mined from a public citation network dataset

1. INTRODUCTION

Since Agrawal et al. introduced the concept of Association Rule Mining[1] in 1993, frequent itemset mining, which is the core subtask of association rule mining, has resulted in fruitful follow-up work. Among the horizontal explorations that target mining substructures more expressive than a subset, including subsequences, subtrees, and subgraphs[11], the Frequent Subgraph Mining (we refer to it as FSM later for short) problem turns out to be the most expressive.

In the typical *graph-transaction setting*, the database consists of a large number of *transactions*, e.g., chemical structures of small molecules. The measure of frequency, a.k.a. support, is then naturally defined as how many transactions a given pattern is observed to be a subgraph of. This definition provides clear semantics in applications. For example, an atom group commonly found among a set of organic compounds may indicate that they potentially share some properties. Moreover, it also satisfies the *downward-closure property* (DCP), which requires that the support of a pattern must not exceed that of its sub-patterns. This property is essential to all frequent pattern mining algorithms, as it enables safely pruning a branch of infrequent patterns in the search space for efficiency.

Nevertheless, when switching to a *single-graph setting*, i.e., the database is itself a large graph and the knowledge inside the single graph is of major concern, the definition of support by counting transactions easily fails because the support of any pattern is simply 0 or 1. In other words, this definition cannot quantify our intuition that a subgraph occurs "frequently" in a large graph.

Indeed, it is straightforward to define the support as the number of "distinct" *embeddings* of the pattern in the graph. However DCP simply does not hold for this case. Consider Figure 1(a), which describes the event "author X writes pa-

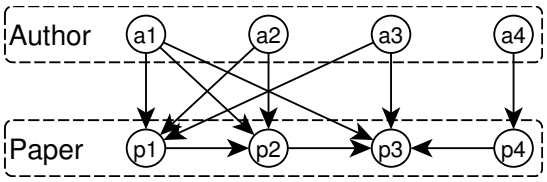


Figure 2: A toy database. "writes" and "cites" labels are omitted.

per Y and Z." When it is matched to a toy database consisting of exactly one author writing n papers, the number of different embeddings for the three vertices is $n(n-1)$. For the sub-pattern "author X writes one paper Y," the count is $n < n(n-1)$. Even if we consider the automorphism of the pattern, and regard two embeddings as identical if one differs from the other by only swapping the papers that Y and Z are matched to, it is still the case that $n < \binom{n}{2}$.

Intuitively, complicated patterns have larger support counts because they tend to reuse elements in the database. Inspired by this observation, Vanetik et al.[29] and Kuramochi et al.[18] redefined the support in the single-graph setting to be the maximum number of edge-disjoint embeddings, which satisfies DCP. According to them, the support of the "an-author-writes-two-papers" pattern on the toy database should be $\lfloor n/2 \rfloor$ ($< n$), since we can find at most that many embeddings without reusing any $\xrightarrow{writes}$ edge. In addition to the increase in problem complexity, we argue that this definition introduces non-determinism to the support computation, which disobeys the human sense that counting is a "one-by-one" procedure.

Since it cannot be avoided that traditional embeddings reuse elements, it seems to be a fact that DCP and intuitiveness can never both be achieved in any subgraph support that counts embeddings of the entire pattern. However, if we assign the count operator to a fixed vertex in a pattern, and treat two embeddings as identical if they match the fixed vertex to the same vertex in the database, we obtain a support measure that regains DCP. Consider Figure 1(a) again, where the author vertex is painted solid and should be counted. On the toy database in Figure 2, though (a_1, p_1, p_2) and (a_1, p_1, p_3) are both embeddings for (X, Y, Z), they only contribute one to the support because they share the same author a_1. Moreover, a_2 and a_3 may also serve to compose legal embeddings, so the overall support is 3. Similarly, Figure 1(b) is a super-pattern of Figure 1(a). Only by matching the author vertex to a_1 or a_2 can we appropriately arrange the two paper vertices given that one should cite the other. So the support is 2 (<3). Best of all, our new support ensures that the new patterns induced have clear interpretations. E.g., the two patterns in Figure 1 actually describe "authors who have at least two papers" and "authors who once cited their own paper," respectively. Since this new class of patterns characterizes vertices that are embedded in a specific local topology, we denote them as *neighborhood patterns*, and the corresponding mining problem as **F**requent **N**eighborhood **M**ining (denoted as FNM for short). By neighborhood we refer to not only other vertices directly linked to the counted vertex as defined in the graph theory terminology, but also to the vertices and edges indirectly connected, along with their labels.

Previously, [10, 15] studied similar problems by defining the number of such "partial matches" as the support of a graph structure. However, only tree-like patterns were addressed as their mining targets. Instead, we try to remove the constraint that cycles are not allowed, and investigate the new class of pattern in the same generalized way that the FSM problem was studied. Our contribution lies in that we established rich and deep connections between the two problems in terms of basic definitions, problem complexity, solutions, and possible optimizations. By testing on a real-life dataset we confirm that trading the problem complexity for better expressivity is worthwhile, as patterns with cycles can lead to more informative and interesting discoveries in the data being investigated. E.g., taking Figure 1(b), 1(a), and the support ratios in their captions into consideration, we can conclude that among all authors who are "able" to cite their own paper (having at least two papers), one out of five will do so.

The rest of this paper is organized as follows: In Section 2 we formalize the FNM problem, where the *Pivoted Subgraph Isomorphism* problem is identified as the core of FNM, similar to what subgraph isomorphism is to the FSM problem. Section 3 discusses our basic solution and further optimization for FNM. We prove that the building blocks of FNM are not as trivial as those of FSM, while some of the ways to optimize the latter can still be adapted for ours. In Section 4 we conduct experiments on real datasets to verify the performance of our solution and the utility of the mined neighborhood patterns. After introducing related and future research we finally conclude the paper.

2. PROBLEM FORMULATION

In this section, we first introduce basic notations to describe a labeled graph and a neighborhood pattern. With the notations we then formulate the decision problem of checking whether a neighborhood pattern matches a given vertex in a large graph as the Pivoted Subgraph Isomorphism problem. We prove that, as the name indicates, this problem is NP-complete, making our problem as difficult as FSM. Finally, after defining the support of a neighborhood pattern as the number of vertices in the database it can be matched to, we briefly justify its downward closure property.

2.1 Labeled Graphs

Definition 1. A (directed) **labeled graph** is a 5-tuple $G = \langle V, L_V, E, \Sigma_V, \Sigma_E \rangle$, where

- V is the set of all vertices;
- Σ_V and Σ_E denote label names used to form vertex and edge labels, respectively;
- $L_V \subseteq V \times \Sigma_V$ is the set of all vertex labels;
- $E \subseteq V \times V \times \Sigma_E$ is the set of labeled edges;

Note that unlike [14, 31, 17], we allow an arbitrary number of labels on a single vertex. This is a reasonable generalized assumption for possible applications. For example, in a knowledge base consisting of objects and their relationships, an object may be a father, a politician, and a vegetarian at the same time. It's also possible that a vertex has no label, i.e., we know nothing about the object, except its existence. On the other hand, parallel edges carrying distinct label

names may link a pair of vertices to model multiple relationships simultaneously existing between two objects. We do not allow edges with no label because we do not process an "arbitrary" or "universal" relationship. Without losing any generality, we do not allow loops, i.e., edges starting and ending with the same vertex. Instead, we can use a vertex label with a specially designed name to indicate the existence of a loop on a vertex. We use "elements" as the joint name of vertex labels and labeled edges, and define $(|L_V| + |E|)$, i.e., the number of elements, as the size of a labeled graph.

2.2 Pivoted Subgraph Isomorphism

Definition 2. A **pivoted graph** is a tuple $\mathcal{G} = \langle G, v_p \rangle$, where

- G is a labeled graph;
- $v_p \in V(G)$ is called the *pivot* of $\mathcal{G}$.

By introducing the concept of "pivot," we aim to characterize the semantics of fixing a vertex in a subgraph to form a neighborhood pattern (e.g., Figure 1), or selecting a vertex in the database to match a pattern to. Previous to us, this notation has at least been used by Vacic et. al [28]. However they were using it to address a different problem.

Definition 3. A pivoted graph $\mathcal{G}_1$ is **pivoted subgraph isomorphic** to $\mathcal{G}_2$, denoted as $\mathcal{G}_1 \subseteq_p \mathcal{G}_2$, if and only if there exists an injective $f : V(\mathcal{G}_1) \to V(\mathcal{G}_2)$ such that

- $\forall (v, l) \in L_V(\mathcal{G}_1)$, $(f(v), l) \in L_V(\mathcal{G}_2)$;
- $\forall (v_1, v_2, l) \in E(\mathcal{G}_1)$, $(f(v_1), f(v_2), l) \in E(\mathcal{G}_2)$;
- $f(v_p(\mathcal{G}_1)) = v_p(\mathcal{G}_2)$.

The first two descriptions indicate that the isomorphic function preserves both vertex labels and edge labels. In addition, the special isomorphism between pivoted graphs requires that the isomorphic function maps the pivot of $\mathcal{G}_1$ to that of $\mathcal{G}_2$. As a subtask of FNM, the problem of deciding whether a pivoted graph is pivoted subgraph isomorphic to another is NP-complete.

THEOREM 1. *The problem of testing pivoted subgraph isomorphism between two arbitrary pivoted graphs is* NP*-complete.*

We prove this in the appendix by reducing it to the classical subgraph isomorphism problem.

PROPERTY 1. *The relation $\subseteq_p$ is transitive.*

PROOF. (Sketch) Given $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$, where $\mathcal{P}_1 \subseteq_p \mathcal{P}_2$ and $\mathcal{P}_2 \subseteq_p \mathcal{P}_3$. Let $f_{12} : V(\mathcal{G}_1) \to V(\mathcal{G}_2)$ be the isomorphic function between $\mathcal{P}_1$ and $\mathcal{P}_2$, and $f_{23} : V(\mathcal{G}_2) \to V(\mathcal{G}_3)$ that for $\mathcal{P}_2$ and $\mathcal{P}_3$. It is clear that there exists $f_{13}(v) = f_{23}(f_{12}(v))$, which is a legal isomorphic function between $\mathcal{P}_1$ and $\mathcal{P}_3$ (it is injective and maps pivot to pivot). Therefore, $\mathcal{P}_1 \subseteq_p \mathcal{P}_3$ holds. □

2.3 Support Measure and its PCP

Definition 4. Given a large labeled graph G, a neighborhood pattern $\mathcal{P}$ **matches** $v \in V(G)$, if $\mathcal{P} \subseteq_p \langle G, v \rangle$. Denoting the set of all vertices in G that $\mathcal{P}$ matches as $M_G(\mathcal{P}) = \{v \in V(G) | \mathcal{P} \subseteq_p \langle G, v \rangle\}$, we define the support of $\mathcal{P}$ in G as the size of $M_G(\mathcal{P})$, and call $\mathcal{P}$ a **frequent neighborhood pattern** of G, if its support is above a given threshold τ.

With the support measure defined, the frequent neighborhood mining problem is simply finding all frequent neighborhood patterns in a large graph, with respect to a given threshold. To control the problem complexity, we further require the mined patterns be connected, i.e., paths exist between the pivot and every other vertex. In later discussions, sometimes we consider the operation of removing a labeled edge from a pattern. If the removal leads to an *redundant* vertex, i.e., a vertex without any vertex label or edge associated to it, we further remove the vertex to make the resulting pattern legal, without affecting its size. If we adopt $\subseteq_p$ to describe the sub-pattern/super-pattern relationship between neighborhood patterns, the fact that a pattern cannot be more frequent than any of its sub-patterns is directly derived via Property 1.

THEOREM 2. *The support measure defined in Definition 4 satisfies the downward closure property.*

PROOF. (Sketch) Given G, $\mathcal{P}_1$, and $\mathcal{P}_2$, where $\mathcal{P}_1 \subseteq_p \mathcal{P}_2$. For any $v \in G$, if $\mathcal{P}_2 \subseteq_p \langle G, v \rangle$, according to Property 1 we have $\mathcal{P}_1 \subseteq_p \langle G, v \rangle$. Therefore, $M_G(\mathcal{P}_1) \supseteq M_G(\mathcal{P}_2)$ and it holds that $|M_G(\mathcal{P}_1)| \geq |M_G(\mathcal{P}_2)|$. □

3. MINING ALGORITHMS

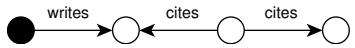
In this section, we describe the algorithm for mining frequent neighborhood patterns, which follows the apriori breadth-first search paradigm. We reveal the major technical difference between mining subgraph and neighborhood patterns. That is, the latter task has more complicated "building blocks." We prove that the building blocks in our task are no longer all frequent size-1 patterns. Instead, they consist of all *frequent paths*, and require special treatments. Additionally, similarities between the solutions and optimizations of FNM and FSM are also discussed.

3.1 Building Blocks

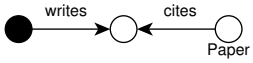
Typically, the traditional FSM algorithm generates subgraph patterns in order of size. First, all frequent subgraphs of size 1 are pre-computed as "building blocks." Then, candidates of size K are obtained by joining pattern pairs of size $(K-1)$ that differ by only one vertex or edge, after which false positives are filtered by a verification against the database. We indicate that the join ensures a complete result because every candidate of size $K \geq 2$ is *decomposable*, that is, we can always find two distinguished elements, and after removing either one we obtain a connected, thus legal, sub-pattern of size $(K-1)$. They may be isomorphic to each other, but their join takes the candidate into consideration.

For our neighborhood mining problem, however, this is not the case. Consider the path-like neighborhood patterns in Figure 3. Obviously, to find a connected sub-pattern of size $(K-1)$ for Figure 3(a), the only choice is to remove the edge and vertex at the end of the path. Meanwhile, in the case of Figure 3(b), only the vertex label to the right can be removed. Otherwise, the resulting patterns will be illegally unconnected. Since they are not *decomposable*, it's impossible to derive them by joining two smaller patterns. Luckily, the following theorem clarifies that these special patterns are only limited to what is described in Figure 3 and Definition 5, which enables us to treat them as building blocks and pre-process them in advance.

Definition 5. A neighborhood pattern is a **path pattern** if the following statements holds:

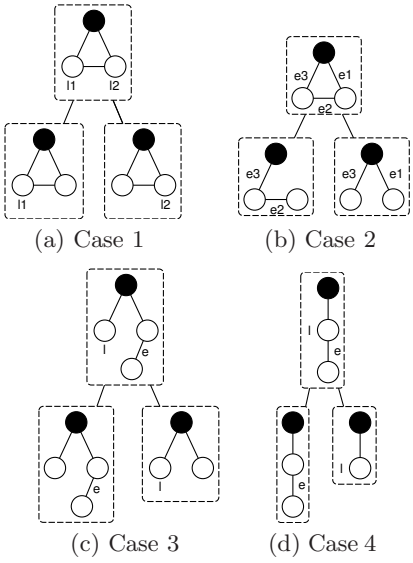


(a) Path pattern without vertex labels



(b) Path pattern with a vertex label

Figure 3: Two variations of path patterns



(a) Case 1 (b) Case 2

(c) Case 3 (d) Case 4

Figure 4: Decomposable cases in the proof of Theorem 3. Labels with no direct influence on the proof are omitted.

- It is a path of labeled edges (directions are ignored) where the pivot is on one end of the path.
- It contains at most one vertex label, which (if it exists) must appear on the other end of the path.

THEOREM 3. *A neighborhood pattern is not decomposable, iff. it is a path pattern.*

PROOF. The sufficiency of the theorem is apparent and has been briefly discussed above. Therefore we only concentrate on proving the necessity.

If a neighborhood pattern $\mathcal{P}$ is not decomposable, it must have at most one vertex label. Otherwise, we can arbitrarily choose two of them as l_1 and l_2, and decompose the pattern as $\mathcal{P} \setminus \{l_1\}$ and $\mathcal{P} \setminus \{l_2\}$, as illustrated in Figure 4(a). Moreover, it must not contain cycles. Otherwise, we arbitrarily choose two edges on the cycle as e_1 and e_2, and decompose it as $\mathcal{P} \setminus \{e_1\}$ and $\mathcal{P} \setminus \{e_2\}$ (Figure 4(b)). Note that this does not harm the connectivity of the patterns since edges on a cycle are not cutting edges.

So far, the shape of $\mathcal{P}$ has been limited to being a tree with at most one label. We transform it to a rooted tree, where the root is the pivot of the pattern. This tree must have only one leaf. If there are two leaves, we can again remove them with associated edges respectively (in the case where the leaf possesses the only vertex label, we only remove the label instead) to decompose the pattern (Figure 4(c)).

Algorithm 1 Building block construction

Input: The single-graph database G, minimum support τ
Output: All frequent path patterns

```
1: queue ← {ε}
2: repeat
3:     path ← queue.Dequeue()
4:     count[].Clear()
5:     for all v ∈ V(G) do
6:         for all nextStep ∈ v.Traverse(path) do
7:             count[nextStep] ← count[nextStep] + 1
8:         end for
9:     end for
10:    for all nextStep ∈ count.Keys() do
11:        if count[nextStep] ≥ τ then
12:            newPath ← path.Append(nextStep)
13:            R ← R ∪ {newPath}
14:            if nextStep.IsEdgeStep() then
15:                queue.Enqueue(newPath)
16:            end if
17:        end if
18:    end for
19: until queue.Empty() = true
20: return R
```

Now the tree with only one leaf is actually a path. However, we still have to prove that if the tree contains a vertex label, it must be on the only leaf: if any vertex other than the leaf carries the label, removing the label and removing the leaf with its associated edge respectively will cause the pattern to decompose (Figure 4(d)). □

3.2 Constructing Building Blocks

As a special case of the general neighborhood patterns, path patterns can still be organized into a level-wise structure, or more exactly, a hierarchical one, which preserves the downward closure property. The parent of each path pattern is uniquely found, by removing the vertex label or the vertex on the other end of the path than the pivot. Thus, the level-wise search algorithm on such a structure deserves an "extending" approach to generate larger patterns from small ones, rather than the "joining" one used for non-building-blocks.

In Algorithm 1, we describe the basic algorithm for finding frequent paths. First, a queue used for the bread-first search is initialized with an empty path ϵ. When extending a path on the front of the queue, we traverse according to the pattern, each time with one vertex in G as the starting point. Note that for each starting point, we should not visit a vertex more than once. Each traversal returns all possible moves when we arrive at the ending point(s) and try to take one more step. E.g., we traverse along a "$\bullet \xrightarrow{writes} \circ \xleftarrow{cites} \circ$" path starting from vertex "Jiawei Han," and stop at the vertex of paper [18] (it cites [31] of Jiawei Han), then all possible moves on [18] may be to follow a "cites" edge to another unvisited paper it cites (such as paper [14]) to produce Figure 3(a), or to terminate the path with a vertex label "Paper" to end up with Figure 3(b). Each time a new move for the current starting point is discovered, the counter for this move is increased by 1. When all traversals are over, those moves with a count of more than τ are used to extend the path. After saving all extended paths to the result set, non-terminated paths, i.e., new paths obtained by appending

Algorithm 2 Frequent neighborhood mining

Input: The single-graph database G, minimum support τ
Output: All frequent neighborhood patterns

```
1: fPaths ← FrequentPaths(G)
2: f_1 ← fPaths.SelectSize(1)
3: k ← 2
4: while f_{k−1}.Empty() = false do
5:     for all 1 ≤ i ≤ j ≤ f_{k−1}.Count() do
6:         c_k ← c_k ∪ Join(f_{k−1}[i], f_{k−1}[j])
7:     end for
8:     for all P ∈ c_k do
9:         if G.CountSupport(P) ≥ τ then
10:            f_k ← f_k ∪ {P}
11:        end if
12:    end for
13:    f_k ← f_k ∪ fPaths.SelectSize(k)
14:    k ← k + 1
15: end while
16: return ⋃_{k≥1} f_k
```

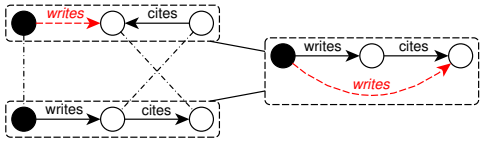
an edge rather than a vertex label such as Figure 3(a), are added to the queue for further expansion. The algorithm terminates when all extendable paths in the queue are consumed.

3.3 Joining and Verifying

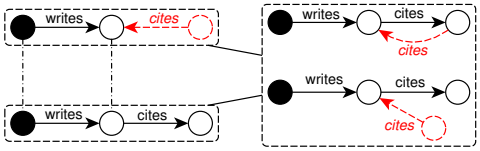
As stated above, what distinguishes our problem from traditional ones solved by a "join-verify-join-..." scheme is the fact that large path patterns cannot be derived by joining smaller patterns, no matter whether these smaller ones are paths, trees, or else. Therefore, the working flow of Algorithm 2 differs from other apriori-based subgraph mining algorithms only by Line 13. This line adds path patterns of the current size to F_k, the frequent non-building-blocks of the same size, to ensure that larger patterns relying on them are not lost due to their absence.

Additionally, our join operation at Line 6 is also worth an detailed explanation. Roughly speaking, we determine whether two patterns should be joined via deleting one element from the first, and check whether the remaining structure is pivoted subgraph isomorphic to the second. Notice that there may be multiple isomorphic mappings so the number of results produced by a single join may be more than one. For each mapping, the deleted element is mapped to the correct position in the second pattern, and inserted to produce a joining result. If the removed element is a vertex label, the join is relatively easy. But if it is an edge, the operation is a bit tricky.

On the one hand, the remaining structure is not necessarily connected after the deletion of an edge. Consider Figure 5(a), where we are going to join the patterns "authors having a cited paper" and "authors having a paper citing another." For the sake of the example and w.l.o.g., we assume that this join is in a branch of the search space where vertex labels are not introduced yet, while readers can still infer by the context that the pivots are author vertices, and the non-pivot vertices represent papers. After deleting the $\xrightarrow{writes}$ edge marked with a dotted line and italic label in the first pattern we obtain an unconnected structure. The remaining structure is pivoted subgraph isomorphic to the second pattern, where the mapping is illustrated by dotted



(a) Deletions may produce unconnected medium results.



(b) Handling dangling edges

Figure 5: Cases of a single join

lines. Since the paper vertex in the first pattern that the deleted edge points to is mapped to the second paper vertex in the second pattern, we restore the mapped $\xrightarrow{writes}$ edge between the author vertex and the second paper vertex to generate the result on the right. Obviously, this pattern is the skeleton of Figure 1(b).

On the other hand, new vertices may be introduced when handling dangling edges. In Figure 5(b) we again try joining the same patterns as in Figure 5(a), but this time we delete the $\xleftarrow{cites}$ edge. As is required in Section 2.3, this deletion makes the second paper vertex redundant, so the vertex is also removed. When the pivoted subgraph isomorphism from the remaining structure to the second pattern is established, the vertex that the deleted edge was associated with is mapped to the first paper vertex in the second pattern, which is the new ending point of the restored edge. But be aware that the new starting point may be the other unmatched paper vertex, as well as an additionally introduced vertex. Neglecting this case will cause the bottom-right pattern to be lost, which is the representative of all tree-like patterns.

Moreover, it should be noted that Line 10 actually embeds a procedure of checking for duplicated patterns. If neglected, they will cause more duplicated patterns, joins, and support computations in later computations. To efficiently check for duplicates, we can hash each produced pattern with the vertex labels on, and the associated edges of the pivot. When a new pattern is produced, we first use the hash table to find potential duplicates, and further verify them with a series of isomorphism checks.

Finally, the last performance overhead of this algorithm lies in the pivoted subgraph isomorphism checker at Line 9 and 10. At this stage, we have not considered adapting any advanced heuristic optimizations of the original subgraph isomorphism problem to ours. In the experiments we simply implemented a depth-first search checker, utilizing an index built on all label names of the large graph G.

3.4 Optimization via VID-Lists

In [17], Kuramochi et al. used TID (Transaction Identifier) lists to optimize their FSM algorithm under the graph transaction setting. Analogously, we propose VID (Vertex Identifier) lists to improve our efficiency both in the building block construction phase and the joining phase. Both of our

Dataset	$\|V\|$	$\|L_V\|$	$\|E\|$	$\|\Sigma_V\|$	$\|\Sigma_E\|$
EntityCube	4,685,439	165,533	75,831	288	207
ArnetMiner	2,495,972	0	7,791,406	0	3

Table 1: Two datasets used in the experiments

optimizations originate from the fact that for any patterns $\mathcal{P}_1 \subseteq_p \mathcal{P}_2$, the set of vertices (transactions in Kuramochi's cases) matching $\mathcal{P}_1$, i.e., $M_G(\mathcal{P}_1)$, must be a superset of $M_G(\mathcal{P}_2)$. This is essentially a reinforced version of the DCP, which enables us to reduce the number of vertices considered when counting the support of a candidate. To utilize it, we have to maintain the IDs of all vertices in $M_G(\mathcal{P})$ as an ordered list for any $\mathcal{P}$, instead of recording only its size.

Specifically, at Line 5 in Algorithm 1, when extending *path*, we only need to consider $M_G(path)$ instead of all vertices in G. In Algorithm 2, for each enumerated pair of patterns at Line 5, we first intersect $M_G(F_{k-1}[i])$ and $M_G(F_{k-1}[j])$ in linear time. If the number of results is below τ, they need not be joined because vertices matching their shared super-patterns must be within the intersection. If they pass the test, $M_G(F_{k-1}[i]) \cap M_G(F_{k-1}[j])$ is saved, and at Line 9 we only need to verify the intersection instead of the whole $V(G)$ to count the support of the size-K patterns. Let's take the join in Figure 5(a) on the toy database in Figure 2 for example. The upper left pattern matches author a_1,a_2, and a_3, and the lower left one matches a_1,a_2,a_3, and a_4. Since a_4 does not appear in the intersection, it will not be checked when computing the support of the joined pattern. Vertices matching the pattern under consideration are stored again in VID lists at Line 7 of Algorithm 1 and Line 9 of Algorithm 2 for the use of larger patterns.

In our experiments, the VID-optimization reduced the running time by up to two orders of magnitude. In Section 4 we will discuss in detail the experimental results and the feasibility of this optimization.

4. EXPERIMENTS

Our experiments were performed on two real datasets: EntityCube[1] and ArnetMiner Citation Network[2] [25, 27, 24, 26], the statistics of which are presented in Table 1. Due to their intrinsic characteristics, the efficiency of our algorithm was mainly tested on the first dataset, while the second was used to showcase the distinctive form of knowledge our method is able to discover. The algorithm was implemented in C# and run on a 2.4G 16-core Intel Xeon PC with 72GB of main memory. The code optimization option was turned on in the compiler. All reported times are in seconds.

4.1 Datasets

4.1.1 EntityCube

The EntityCube system is a research prototype for exploring object-level search technologies, which automatically summarizes the Web for entities (such as people, locations, and organizations). We utilized the relationship network between person entities extracted by the system. Specifically, with a list of people names as seeds, we queried the system using one name each time, and got related persons and the corresponding relationship names in return. On the one hand, the seed people and returned people were used to form the vertex set V. On the other hand, the system returned two types of relationships. The name of one was in the plural form, such as "politicians(Barack Obama, Bill Cliton)," indicating the connection that they're both politicians. Thus, the relationship name naturally served as vertex labels for the two associated entities. The other type of relationship appeared in the singular form, e.g., "wife(Michelle Obama, Barack Obama)". They were interpreted as labeled edges between the corresponding vertices.

4.1.2 ArnetMiner

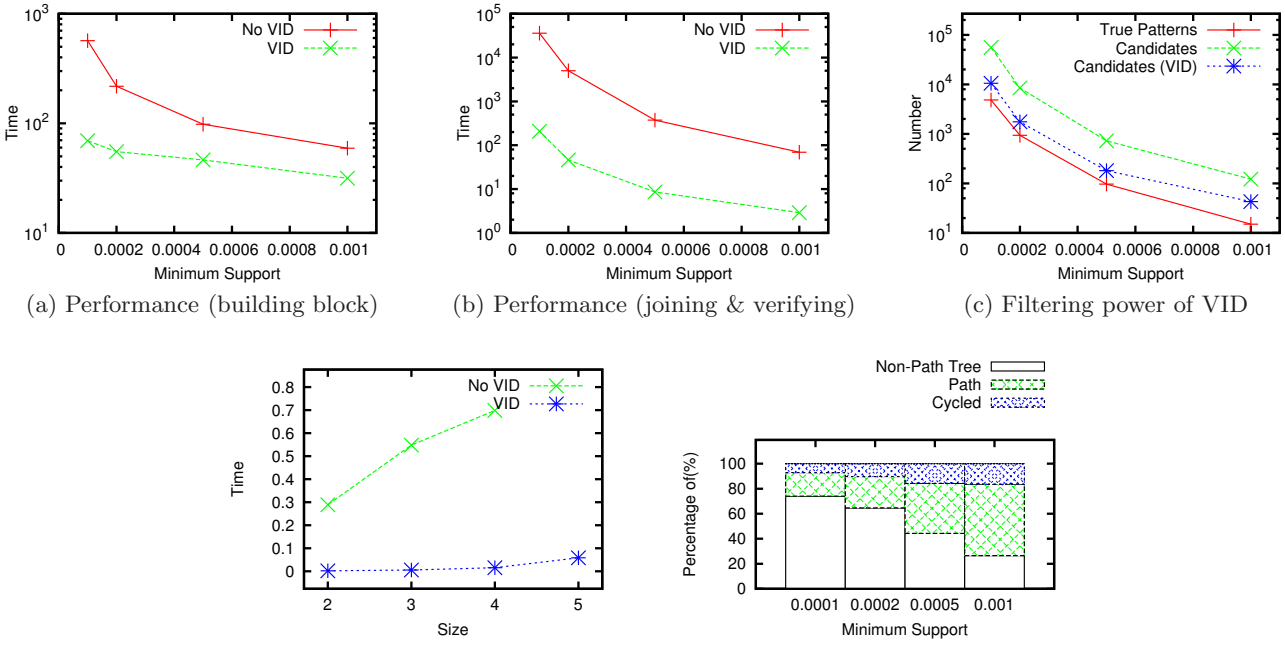
The ArnetMiner Citation Network dataset contains many papers with associated attribute information, as well as their citation relationship. The dataset consists of five versions and we use the fifth one. We extracted all papers, authors, and conferences appearing in the data, and constructed the vertex set with them. Conferences of the same series but in different years were treated as identical. There were only three types of labeled edges, i.e., "writes" between an author and a paper, "accepts" between a conference and a paper, and "cites" between one paper and another. Because these edge label names actually imply the type of both the starting and ending vertices of an edge, we didn't employ any vertex labels to avoid redundancy. In the data, IDs were provided to uniquely denote papers and form citations, which were adopted by us. However in the author and conference sections of each paper, only texts were presented. Therefore, when converting them into the IDs in our algorithm, we required an exact text-match and didn't perform any cleaning operation involving external data.

4.2 Performance

In this section, we report the performance of our algorithms for frequent neighborhood mining. Since no previous work has addressed exactly the same problem, our experiments were dedicated to validating the feasibility of our VID optimization. In practice, the running time of such a pattern-mining algorithm is heavily influenced by the size of the result set. Therefore, we decided to conduct the experiments on the EntityCube data with rich label names, for it has a potentially larger result set and the running time is more sensitive to parameters such as the minimum support τ. In all experiments, τ was chosen from $\{0.0001, 0.0002, 0.0005, 0.001\}$ and all reported times were an average of five consecutive runs.

In Figures 6(a) and 6(b), we terminated the search after all frequent patterns below size 4 were discovered. It is clear that our VID optimization successfully accelerates both the building block construction and the join-verify phases by up to one and two orders of magnitude, respectively. Analogous to the TID optimization [17] for FSM, the advantages of the VID optimization are two-fold. First, two patterns will not be joined if the intersection of their VID lists is smaller than τ. Therefore, the algorithm successfully avoids verifying false positives caused by joining unpromising pattern pairs, which is a vital overhead to the overall performance. In Figure 6(c), the number of candidates with/without the VID-list-pruning, and the number of true patterns are illustrated. This figure shows that the pruning helps narrow

[1]http://entitycube.research.microsoft.com/
[2]http://arnetminer.org/citation



(a) Performance (building block) (b) Performance (joining & verifying) (c) Filtering power of VID

(d) Verification time per candidate (e) Percentage of different pattern types

Figure 6: Experiments on the EntityCube dataset

down the number of candidates by several times. Second, for each pair of patterns that passes the pruning, the time spent on counting the support of the joining results is also reduced because vertices that are not in the intersection won't contribute to the count, and therefore are not checked. Figure 6(d) presents the average verification time for each non-path candidate where $\tau = 0.0001$. The x-axis denotes different stages of the search procedure, where candidates of size 2 to 5 are verified. Obviously the VID optimization is significantly effective for verifying candidates of all sizes. Particularly, without the optimization, the algorithm didn't finish the verification of size 5 in a reasonable time.

4.3 Interpretation of Mined Patterns

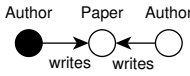
As mentioned above, the major superiority of FNM over [10, 12, 15] is that it discovers patterns with cycles that are not targeted by others. With the hands-on experience of experimenting on both datasets, we realize that a cyclic pattern can be viewed as a set of constraints with a lower degree of freedom than a tree-like one of the same size. Figure 6(e) shows the constitution of all patterns in EntityCube data, whose size are below 5. Patterns with cycles actually make up around 10% among all three types. The trend also shows that when we decrease the support ratio and specify a pattern into its super-patterns, it is more difficult for a cycle to form, than a fork to appear.

However, once formed, patterns with cycles serve as a good complement to tree-like patterns. Introducing them does not linearly increase our knowledge about the data being investigated, but actually makes a mutual reinforcement with tree-like ones. As patterns from the ArnetMiner dataset have better interpretability, we selected some interesting neighborhood patterns mined from this dataset to demonstrate our points. In addition to the example given in Figure 1, more patterns are displayed in Figure 7. By combining two or more of them, we can make very interesting discoveries.

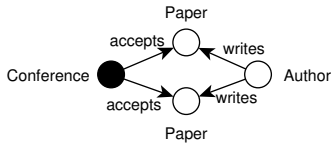
For example, the support ratio of Figure 7(g) is lower than those of Figures 1(a) and 7(f), which reflects the common sense that it is more difficult to get one's paper cited than to write more papers. Also, the small gap between the ratios of Figures 1(a) and 7(e) reveals the fact that most writers are willing to maintain a co-authoring relationship. On the other hand, the ratios of Figures 1(a) and 7(h) together prove that an average author relatively favors a conference that once accepted his paper. Moreover, Figure 7(c) alone points out that most of us (assuming that we all have papers) have a paper with no less than three authors. Surprisingly, as Figure 7(l) indicates, there are even papers citing each other! By checking the data we find two cases of such a phenomenon. One is caused by the dataset itself. The data treats books as conferences, and their chapters as papers. Of course, chapters from the same book can cite each other. This case is rare. The other case is more common: an author simultaneously submitted two papers to the same conference and got them both accepted. When preparing the camera-ready versions, he had them cite each other.

When browsing all mined patterns, we also find that, despite those interesting ones, the results tend to be flooded by a significant number of uninteresting patterns. Most of them are superpositions of some basic components. E.g., "authors citing his own paper, while having a third paper". We believe that this is caused by the vagueness of "interestingness" and the gap between "frequent" and "truly interesting". One possible way out may be to compress the results (e.g., require them to be *closed* [31]). We leave it as our future work.

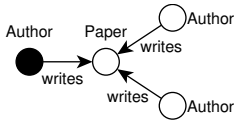
For all patterns presented in this paper, we use support ratios w.r.t. vertices with a specified label, instead of the absolute count mentioned in the problem statement. Such



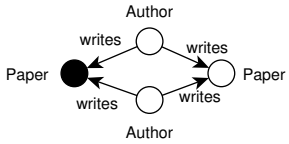
(a) 94.6% of authors have a co-author.



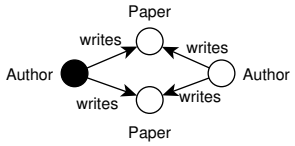
(b) 89.0% of conferences once accepted two papers from the same author.



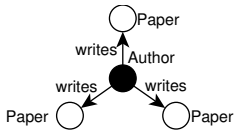
(c) 78.9% of authors have a paper with two co-authors.



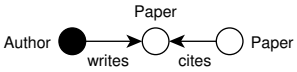
(d) For about 61.8% of papers, two of their authors have other cooperations.



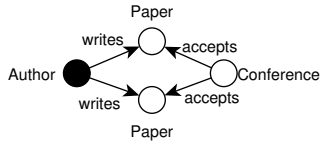
(e) 37.9% of authors have a co-author cooperating twice with him.



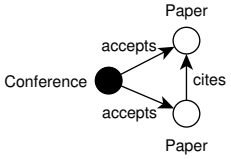
(f) 31.4% of authors have at least three papers.



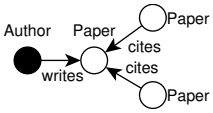
(g) 27.6% of authors have a cited paper.



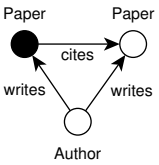
(h) 25.4% of authors have two papers accepted by one conference.



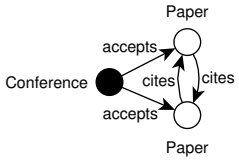
(i) 23.0% of conferences once accepted a paper and another paper this one cited.



(j) 19.0% of authors have a paper that was cited at least twice.



(k) About 9.8% of papers cite another paper of one of its authors.



(l) 6.3% of conferences once accepted two papers citing each other.

Figure 7: Selected interesting neighborhood patterns in the ArnetMiner dataset.

Vertex Type	Number	Time	Patterns	Cyclic
Conference	6,713	3,354	135	30
Author	916,979	9,469	163	24
Paper (sampled)	500,000	53,422	796	147

Table 2: Statistics of the runs on ArnetMiner

a variation enhances the interpretation of patterns, and it only requires a small modification to the algorithm to apply the variation. Suppose we want facts about all authors with a minimum support ratio of 1%. In Algorithms 1 and 2, when a scan on the entire $V(G)$ is required for support counting, we only scan those author vertices by accessing an index on the label names. Moreover, when calling the modified algorithms, τ should be assigned with 1% of the number of all author vertices. With this variation, though it is easy to mine a mixture of, say, paper and author vertices, it is insignificant, at least in this dataset. Because a pattern that simultaneously describes papers and authors seldom exists, mining such a heterogenous vertex set tends to result in the union of results of mining papers and authors separately. When mining patterns about authors, conferences, and papers, the support ratios were uniformly set to 1%. As the number of paper vertices is huge (over 1.5 million), the support calculation was performed on a subset sampled from the paper vertex set, which consists of 0.5 million papers. We didn't explore path patterns of size 4, or any pattern whose size exceeds 4. Readers may refer to Table 2 for more details.

5. RELATED WORK

5.1 Frequent Subgraph Mining

The frequent subgraph mining problem is well-investigated by the literature under the graph-transaction setting. Among them, the most influential methods include AGM[14], FSG[17], gSpan[30], FFSM[13], and Gaston[20]. The first and second adopt the apriori-based BFS scheme, and feature vertex-incremental and edge-incremental approaches, respectively. The last three fall under the pattern-growth-based DFS category. To the best of our knowledge, there exists no reasonable conversions between graph-transactional FSM and our FNM in either directions. However, the optimizations they utilized, such as canonical labeling, vertex invariants, and depth-first search, were inspiring and potentially employable in our method.

The literatures tackling the single-graph setting, however, are still at the stage of proposing various support measures, due to the reasons we mentioned above. The Maximum Independent Set support and corresponding mining algorithms were studied in [18, 29]. When calculating the MIS support of a pattern, one needs to solve a maximum independent set problem on the *instance-overlap graph* of that pattern,

which is NP-hard. The Harmful Overlap support[7, 8] and the MiNimum Image support[3] avoid solving the MIS problem. However, compared with our neighborhood patterns, which can be directly interpreted as "X% percent...", subgraph patterns mined under their supports cannot provide such intuitive interpretations. [21] discussed generalizations, optimizations, and approximations for the MNI-based methods. [2] proposed a network-flow based definition of single-graph support, and is more application-oriented.

It is worth noting that [29, 20] also use paths as building blocks to generate larger patterns. In their FSM problem setting, they start the mining procedure with paths instead of size-1 patterns to reduce the number of duplicate candidates and improve efficiency. However, in our problem setting, path is not an alternative, but a mandatory building block to ensure the completeness of results.

In recent works on heterogeneous information networks[23, 16], their key concept "meta path" resembles our path pattern described in Figure 3(a). As a chaining join of multiple binary relations, a meta path is essentially a binary relation between instances. However, a path pattern is roughly an unary relation, which describes all instances associated with such a path.

5.2 Frequent Tree Query Mining in Graphs

In [10, 12, 15], the authors attempted to mine tree patterns in graphs, whose support measure resembles ours in the way that distinct matches of some vertices are counted, while the match conditions on the others are only existential. They did not target patterns with cycles, which added a great deal to the users' understanding of the data. Moreover, because [10, 12] allowed multiple vertices to be counted (in other words, as our "pivots"), their problems were more complicated and thus did not completely follow and benefit from the well-solved apriori pattern mining scheme. We argue that, patterns with more than three pivots may explode in number, while bringing about some knowledge that is less explainable and utilizable. Finally, these papers both claimed that their mining algorithms supported constants in the patterns, e.g., "x% of the authors once cited a paper published in CIKM." Our problem setting supports multiple labels on a vertex. Therefore, we can achieve it by simply adding the name of each vertex to its label set. We can also modify our algorithm to implicitly perform such a data transformation.

5.3 ILP Related Works

In [4], Dehaspe et al. introduced an inductive logic programming system for mining frequent patterns in a datalog database. Their frequency measure for patterns are similar to ours, while the confirm of a match is slightly different. For our isomorphism-based match, two vertices in the source graph cannot be mapped to the same vertex of the target graph. However, they allow different variables in the pattern to be bound to the same constant, which resembles the graph homomorphism problem in some sense. We argue that, none of the two settings is contained by the other. Consider Figure 1(a), the two paper vertex are supposed to be matched to different paper vertices in the database, which exactly conveys the semantics of "has two papers". However, in their setting, it is trivially equivalent to a pattern where an author vertex points to a paper vertex with label "writes". Conversely, despite the fact that a father may become a friend to his child, when describing "a person with a father and a friend", our setting unnecessarily partitions this pattern into two patterns, each specifying whether the father and friend are or are not the same person. [6] also adopts the homomorphism setting, but only tree-shaped patterns are addressed. Methods learning horn clauses from knowledge bases such as [22, 19, 9] also belong to the Inductive Logic Programming category. These methods are characterized by a variety of metrics to evaluate the utility of a rule. Since noise and scalability issues in real data are their main concerns, they adopt stricter language biases and the rules mined are of more limited forms.

6. FUTURE WORK

Under the current problem setting and solution, encouraging results have been achieved in terms of performance and result utility. However, our method can still be further extended from the following aspects. First, the definition of *closed* neighborhood patterns may be introduced in a similar way as [31]. A pattern is closed if there exists no proper super-pattern with the same support. This definition is expected to significantly reduce the size of the results, while preserving the most meaningful ones. Second, the pivots may be allowed to be an edge to enable characterizing the "neighborhood" of an edge. This generalized pattern introduces new semantics, e.g., "x% of all citations are made between papers from the same institutes." Finally, we are also interested in exploring whether neighborhood patterns are effective in single-graph-based classification tasks, since it is reasonable to assume that vertices located in similar neighborhoods have similar properties[5]. We believe that neighborhood-based features or kernels are orthogonal to distance and connectivity-based techniques, and have potential applications in the social network research.

7. CONCLUSION

In this paper, we addressed mining single-graph databases and introduced the new neighborhood patterns as mining targets. They have clear semantics and are not limited to tree-like shapes. We formally defined the frequent neighborhood mining problem, and proved that it is as difficult as the frequent subgraph mining problem. We indicated that the major difference between FNM and FSM in terms of solution is that our patterns have non-trivial building blocks, which are clearly separated by us via a theorem and proof. After discussing possible optimizations, we conducted experiments on two real datasets to validate the efficiency and effectiveness of our method. The algorithm has proven to be feasible and shows a unique ability to provide users with especially interesting insights into the analyzed data.

8. ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their helpful comments. We also thank Changliang Wang and Chunbin Lin for their discussions and feedbacks.

9. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD Conference*, pages 207–216, 1993.

[2] P. Anchuri, M. J. Zaki, O. Barkol, R. Bergman, Y. Felder, S. Golan, and A. Sityon. Infrastructure pattern discovery in configuration management databases via large sparse graph mining. In *ICDM*, pages 11–20. IEEE, 2011.

[3] B. Bringmann and S. Nijssen. What is frequent in a single graph?. In *PAKDD*, pages 858–863, 2008.

[4] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and knowledge discovery*, 3(1):7–36, 1999.

[5] C. Desrosiers and G. Karypis. Within-network classification using local structure similarity. In *Machine Learning and Knowledge Discovery in Databases*, pages 260–275. Springer, 2009.

[6] A. Dries and S. Nijssen. Mining patterns in networks using homomorphism. In *Proceedings of the twelfth SIAM International Conference on Data Mining*, pages 260–271, 2012.

[7] M. Fiedler and C. Borgelt. Subgraph support in a single large graph. In *Data Mining Workshops, 2007. ICDM Workshops 2007*, pages 399–404. IEEE, 2007.

[8] M. Fiedler and C. Borgelt. Support computation for mining frequent subgraphs in a single graph. In *Proc. 5th Int. Workshop on Mining and Learning with Graphs (MLG 2007, Florence, Italy), Florence, Italy*, pages 25–30. Citeseer, 2007.

[9] L. Galarraga, C. Teflioudi, K. Hose, and F. M.Suchanek. Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22th international conference on World Wide Web*, WWW '13, 2013.

[10] B. Goethals, E. Hoekx, and J. V. den Bussche. Mining tree queries in a graph. In *KDD*, pages 61–69, 2005.

[11] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

[12] E. Hoekx and J. V. den Bussche. Mining for tree-query associations in a graph. In *ICDM*, pages 254–264, 2006.

[13] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*, pages 549–552. IEEE, 2003.

[14] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23, 2000.

[15] G. Jeh and J. Widom. Mining the space of graph properties. In *KDD*, pages 187–196, 2004.

[16] X. Kong, P. S. Yu, Y. Ding, and D. J. Wild. Meta path-based collective classification in heterogeneous information networks. In *CIKM*, pages 1567–1571. ACM, 2012.

[17] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *Knowledge and Data Engineering*, 16(9):1038–1051, 2004.

[18] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data mining and knowledge discovery*, 11(3):243–271, 2005.

[19] N. Lao, T. M. Mitchell, and W. W. Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, pages 529–539, 2011.

[20] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 647–652. ACM, 2004.

[21] M. E. Saeedy and P. Kalnis. Grami: generalized frequent pattern mining in a single large graph. 2011.

[22] S. Schoenmackers, J. Davis, O. Etzioni, and D. S. Weld. Learning first-order horn clauses from web text. In *EMNLP*, pages 1088–1098, 2010.

[23] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB'11*, 2011.

[24] J. Tang, L. Yao, D. Zhang, and J. Zhang. A combination approach to web user profiling. *ACM TKDD*, 5(1):1–44, 2010.

[25] J. Tang, D. Zhang, and L. Yao. Social network extraction of academic researchers. In *ICDM'07*, pages 292–301, 2007.

[26] J. Tang, J. Zhang, R. Jin, Z. Yang, K. Cai, L. Zhang, and Z. Su. Topic level expertise search over heterogeneous networks. *Machine Learning Journal*, 82(2):211–237, 2011.

[27] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008.

[28] V. Vacic, L. M. Iakoucheva, S. Lonardi, and P. Radivojac. Graphlet kernels for prediction of functional residues in protein structures. *Journal of Computational Biology*, 17(1):55–72, 2010.

[29] N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data. In *ICDM*, pages 458–465, 2002.

[30] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

[31] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *KDD*, pages 286–295, 2003.

APPENDIX

Proof of Theorem 1. We prove it by reducing it from the subgraph isomorphism problem. Labels are ignored because it is a generalization of, thus reducible from, the non-label case.

Given an instance $\langle G_1, G_2 \rangle$ of the subgraph isomorphism problem, we add a new vertex v_1 to G_1, and v_2 to G_2, respectively. They are marked as pivots and edges are created from them to all vertices of the same graph. Obviously, $G_1 \subseteq G_2$ iff. $\langle G_1, v_1 \rangle \subseteq_f \langle G_2, v_2 \rangle$. By solving the pivoted subgraph isomorphism problem $\langle \langle G_1, v_1 \rangle, \langle G_2, v_2 \rangle \rangle$ we are able to answer whether $G_1 \subseteq G_2$. So our problem is NP-hard. The solution of an instance of our problem is verified in polynomial time. Therefore, our problem is NP-complete.