Towards Proximity Pattern Mining in Large Graphs

Arijit Khan
Dept. of Computer Science
University of California at
Santa Barbara, CA
93106-5110, USA
arijitkhan@cs.ucsb.edu

Xifeng Yan
Dept. of Computer Science
University of California at
Santa Barbara, CA
93106-5110, USA
xyan@cs.ucsb.edu

Kun-Lung Wu
IBM T. J. Watson Research
Center
19 Skyline Drive, Hawthorne
NY 10532, USA
klwu@us.ibm.com

ABSTRACT

Mining graph patterns in large information networks is critical to a variety of applications such as malware detection and biological module discovery. However, frequent subgraphs are often ineffective to capture association existing in these applications, due to the complexity of isomorphism testing and the inelastic pattern definition.

In this paper, we introduce proximity pattern which is a significant departure from the traditional concept of frequent subgraphs. Defined as a set of labels that co-occur in neighborhoods, proximity pattern blurs the boundary between itemset and structure. It relaxes the rigid structure constraint of frequent subgraphs, while introducing connectivity to frequent itemsets. Therefore, it can benefit from both: efficient mining in itemsets and structure proximity from graphs. We developed two models to define proximity patterns. The second one, called *Normalized Probabilistic Association* (NmPA), is able to transform a complex graph mining problem to a simplified probabilistic itemset mining problem, which can be solved efficiently by a modified FP-tree algorithm, called pFP. NmPA and pFP are evaluated on real-life social and intrusion networks. Empirical results show that it not only finds interesting patterns that are ignored by the existing approaches, but also achieves high performance for finding proximity patterns in large-scale graphs.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: data mining; I.5.1 [**Pattern Recognition**]: Models—*statistical*

General Terms

Algorithms, Performance

Keywords

Graph, Association, Pattern, Mining

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

1. INTRODUCTION

Graph patterns are building blocks for several key graph applications, including graph indexing, graph search, graph classification and clustering [37, 13, 12, 40]. Existing graph pattern mining algorithms, like those developed in [19, 22, 23, 32, 9, 20, 28], achieved great success using strategies that efficiently traverse the pattern space. However, the definition of frequent subgraphs might not be appropriate for new application scenarios present in social and information networks. First, the definition is not elastic enough to capture fuzzy patterns existing in massive attributed graphs. Figure 1 shows one example, where each node is attached with a set of labels. These labels can be movies recommended by a user, functions carried by a gene, or intrusions initiated by a computer. As illustrated in Figure 1, a, b, c often occur together and formulate an association pattern, while d, c are not associated together. However, $\{a, b, c\}$ is neither a frequent subgraph, nor a frequent itemset if we treat each node as a transaction. Pattern $\{a, b, c\}$ has three characteristics: (1) Proximity, these three labels are tightly connected; (2) Frequency, they appear many times; (3) Flexibility, they are not always connected in the same way. Due to these characteristics, we can not apply the traditional frequent graph mining algorithms such as FSG [23] and gSpan [36] to find them. On the other hand, frequent itemset mining [4, 16] can not be used either, since $\{a, b, c\}$ do not appear in the same set of nodes.

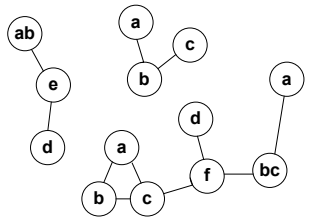


Figure 1: Proximity Pattern $\{a, b, c\}$

Secondly, for small graphs such as chemical structures, isomorphism checking is never a problem as demonstrated by the existing frequent graph mining algorithms. However, for large graphs like intrusion networks and social networks, there can be a huge set of isomorphic embeddings existing for frequent subgraphs. It becomes costly to generate all kinds of frequent subgraphs. To overcome the above two issues, we propose a new graph pattern concept, called *Proximity Pattern*. A proximity pattern is a subset of labels that repeatedly appear in multiple tightly connected subgraphs in

G. $\{a, b, c\}$ in Figure 1 is an example. Proximity pattern is an itemset. However, it has a connectivity requirement: the labels must be associated tightly and frequently in the graph. For example, in a social network, it can be a set of movies that are watched by multiple groups of users. That is, in order to find proximity patterns among movies, one should not only consider the collection of movies watched by each person (in this case, it is a traditional itemset mining problem); instead, one should also consider the movies watched by his or her friends and friends of friends. In this case, labels associated with two different nodes are related due to the connection between these two nodes. The same mining problem also exists in finding associations of intrusions on the Internet, where each node corresponds to an IP address and there is a directed edge between two IP addresses if an intrusion attack takes place between them. It is interesting to find the association of different attack types, which can be used to analyze intrusions.

In this paper, we first introduce an intuitive neighbor association model to define and allocate proximity patterns by identifying the embeddings of these patterns in a graph and then finding a weighted maximum independent set among these embeddings. Although this approach is intuitive, it is inefficient to find patterns in large graphs due to the complexity of embedding enumeration and maximum independent set finding. Therefore, we redefine proximity patterns from an influence point of view, using a probabilistic information propagation model. Based on this model, we propose novel techniques for finding proximity pattern within a large graph, which consider conditional probabilistic association of the labels at each vertex. In the end, a statistical test is developed to measure the significance of discovered proximity patterns.

Our Contributions. To the best of our knowledge, this is the first paper introducing the concept of proximity patterns in large graphs.

We model the problem of determining the proximity among labels in two distinct approaches, neighbor association and information propagation. While the neighbor association model is a direct approach of finding the association among labels based on their distance across the edges of the graph, we have shown that this method is not efficient for large scale graphs. In the information propagation model, we develop novel probabilistic techniques to determine the proximity among labels in a graph database, based on the Markov model [29]. We justify that they will be efficient as well as consistent under interpretations of "relation between transactions" and the "association of labels". The propagation model is able to transform a complex graph mining problem to a simplified probabilistic itemset mining problem, which can be solved efficiently by a modified FP-tree algorithm, called pFP(probabilistic FP-growth). Furthermore, for the discovered patterns, we define an objective function that will measure their interestingness using randomized test.

In summary, we propose a complete pipeline to define and mine proximity patterns in massive graphs in a scalable manner. As tested in real-life social networks and intrusion networks, proximity patterns turn to be interesting and are able to capture patterns missed by frequent itemsets and frequent subgraphs.

The rest of the paper is organized as follows. In Section 2, we define the abstract problem formulation and preliminaries. Our models are introduced in Sections 3 and 4. The probabilistic frequent itemset mining algorithm is described in Section 5. We analyze the experimental results in Section 6, and discuss related work in Section 7, followed by conclusions in Section 8.

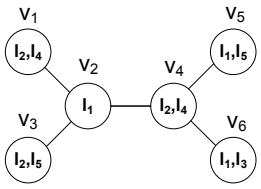


Figure 2: Frequent Itemset vs Proximity Pattern

2. PRELIMINARIES

An attributed graph $G = (V, E)$ has a label set L and each node is attached with a set of labels. The label set of a node u in G is $L(u)$. Let I be a subset of labels such that the labels in I tightly connect and appear repeatedly in G. I is named as *"Proximity Pattern"*. Proximity patterns are degenerated to frequent itemsets, if we drop all the edges in G. In this work, we focus on *bidirectional* and *unweighted* graphs. However, the proposed models and algorithms can be applied to *directed* graphs as well. Some modifications are required for *weighted* graphs, which we shall discuss later in Section 4.3.

Let $D = \{t_1, t_2, \ldots, t_m\}$ be a set of *independent transactions* (in the context of attributed graphs, the set of nodes). Each transaction contains a subset of items in L.

DEFINITION 1 (SUPPORT). *The support $sup(I)$ of an itemset $I \subseteq L$ is the number of transactions in the data set that contain I. Sometimes, we also use the percentage to represent support.*

An itemset is called *frequent* if its support is greater than a user-defined minimum threshold. Nearly all the classical frequent itemset mining algorithms apply the property of *Downward Closure* [4] to prune the pattern search space.

DEFINITION 2 (DOWNWARD CLOSURE). *For a frequent itemset, all of its subsets are frequent; and thus for an infrequent itemset, all of its supersets must be infrequent.*

Unfortunately, since frequent itemset mining does not consider the connections in an attributed graph, it might miss interesting patterns. Figure 2 shows an example. If we consider each node as an independent transaction, $\{l_1, l_2\}$ will not be reported as a frequent itemset. The two items do not occur together in any of the nodes. However, a careful examination of Figure 2 reveals that they always occur within one-hop distance of each other. $\{l_1, l_2\}$ is a proximity pattern: l_1 is associated in the proximity of l_2.

For a proximity pattern I, we need to identify locations of this pattern in G. Each of these locations shall contain all of labels in I.

DEFINITION 3 (EMBEDDING AND MAPPING). *Given a graph G and a subset of vertices π, $\pi \in V(G)$, Let $L(\pi)$ be the set of labels in π, i.e., $L(\pi) = \cup_{u \in \pi} L(u)$. Given a label subset I, π is called an embedding of I if $I \subseteq L(\pi)$. A mapping ϕ between I and the vertices in π is a function*

$\phi : I \rightarrow \pi$ *s.t.,* $\exists\ l,\ \phi(l) \in \pi$ *and* $l \in L(\phi(l))$. *A mapping is minimum if it is surjective, i.e.,* $\forall v \in \pi, \exists l$ *s.t.* $\phi(l) = v$.

In Figure 2, $\{v_1, v_2, v_3\}$ forms an embedding of $\{l_1, l_2, l_5\}$. There can be two possible mappings in this embedding: (1) ϕ_1 maps l_1 to v_2, l_2 to v_1, and l_5 to v_3, and (2) ϕ_2 maps l_1 to v_2, l_2 to v_3, and l_5 to v_3. In these two mappings, ϕ_1 is minimum, ϕ_2 is not. The vertices in π might not be connected. For example, $\{v1, v3\}$ is an embedding of $\{l_4, l_5\}$

Given an itemset I and a mapping ϕ, we need a function $f(\phi)$ to measure its association strength: how tightly the mapped labels in π are connected. For example, $f(\phi)$ could be the inverse of diameter of ϕ or the inverse of $\sum_{u,v \in V(\phi)} d(u, v)$, where $d(u, v)$ is the shortest distance between u and v. Since there could be multiple mappings in π, we always choose the mapping that has the highest value of $f(\phi)$. To simplify the presentation, we also denote the strength of an embedding as $f(\pi)$.

In the next section, we are going to investigate two models to define the support of proximity patterns.

3. NEIGHBOR ASSOCIATION MODEL

The complexity of proximity patterns rises from the interconnections of labels in a graph. One has to perform the following three steps to identify proximity patterns:

Step 1. Find all the embeddings, $\pi_1, \pi_2, \ldots, \pi_m$ of an itemset I in the graph,

Step 2. For each embedding π, measure its strength $f(\pi)$,

Step 3. Aggregate the strength of the embeddings, $F(I) = \sum_{i=1}^{m} f(\pi_i)$. Take $F(I)$ as the support of I.

In order to find the support of a proximity pattern, one has to first enumerate all the embeddings of the pattern. Unfortunately, due to graph connections, there could be an exponential number of redundant embeddings. First, the boundary between the embeddings of a pattern is not obvious. When two embeddings overlap, the overlapped part might be double counted. The support derived from multiple embeddings will violate the downward closure property (Definition 2). That is, the support of a pattern I might be less than a pattern I', even though $I \subseteq I'$, which makes it difficult to design fast mining algorithms. Secondly, any subset of vertices, π, could be an embedding of a pattern I as long as $I \subset L(\pi)$, though for those loosely connected embeddings, their strength might be negligible.

In order to solve the above two issues, we introduce two models in this paper, neighbor association model and information propagation model.

Let $\pi_1, \pi_2, \ldots, \pi_m$ be the embeddings of I in G. we build an overlapping graph: each node represents an embedding and an edge connects two embeddings if they share at least one common vertex. In the overlapping graph, each node has $f(\pi)$ as its weight. Figure 3 shows an example of a partial overlapping graph derived from Figure 2.

For frequent graph mining in a single graph, Kuramochi and Karypis [24] proposed using the maximum independent set as the support of subgraphs, which is proved to have the downward closure property [15]. An independent set in a graph is a subset of vertices with no edge connecting them. In Figure 3, embeddings π_1, π_4 form a maximum independent set. This concept can be extended to the overlapping

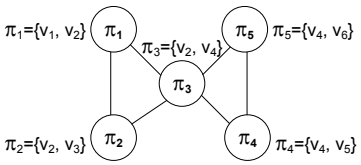


Figure 3: Overlapping Graph

graph with weights. For a label set I, the support of I could be the sum of vertex weights derived by the maximum weight independent set. It can be proved that the support defined using maximum weight independent set has the downward closure property too. We call this model *Neighbor Association Model*.

While the neighbor association model solves the pattern overlapping issue, it is NP-hard in general with respect to the number of embeddings for a given pattern [38]. Since the number could be huge, in practice, it is not feasible to generate all the embeddings of proximity patterns and then find their maximum weight independent set. Thus we resort to the second model, *Information Propagation Model*.

4. INFORMATION PROPAGATION MODEL

The neighbor association model examines the association from a graph structure perspective. For example, for two labels l_1, l_2, in a graph, how closely they are connected and how often they are connected. It is possible to examine the same problem from a network influence perspective. Take a movie recommendation social network as an example, where users could recommend movies to their friends. Assume G_0 is the initial graph. Based on the recommendations, users might watch more movies and generate a new graph G_1 with updated watched movie lists. This process iterates until it reaches a stable graph where the movie list for each user does not change any more.

$$G_0 \rightarrow G_1 \rightarrow \ldots \rightarrow G_n.$$

In an ideal situation, it is meaningful to mine frequent itemsets in G_n. However, in reality we only have an incomplete snapshot between G_0 and G_n. Proximity patterns in G_i could be interpreted as an approximation to frequent itemsets in G_n. With that being said, if we are able to simulate the influence process by generating $\widetilde{G}$ from G_i to approximate G_n, we can instead use frequent itemsets mined from $\widetilde{G}$ to represent proximity patterns in G_i. This is the main idea of the information propagation model.

We model the influence process using a first order Markov model. The given graph is considered as the present state, and the association among labels in the future state will be reached through an iterative stochastic process. Let $L(u)$ be the present state of u, denoted by the labels present in u, and l be a distinct label propagated by one of its neighbors and $l \notin L(u)$. Hence, the probability of observing $L(u)$ and l is written as

$$P(L \cup \{l\}) = P(L|l)P(l), \tag{1}$$

where $P(l)$ is the probability of l in u's neighbors and $P(L|l)$ is the probability that l is successfully propagated to u.

For multiple labels, $l_1, l_2, \ldots, l_m$, the joint probability of observing $L \cup \{l_1, \ldots, l_m\}$ can be written as, assuming each label is propagated independently,

$$P(L \cup \{l_1, \ldots, l_m\}) = P(L|l_1) * \ldots * P(L|l_m) * P(l_1) * \ldots P(l_m). \quad (2)$$

The propagation model captures an important characteristic in social graphs where nodes can influence each other. As the distance increases, the influence decreases [1], which is exactly what proximity patterns would like to capture. In the next two subsections, we introduce two distinct approaches to assign values to the aforementioned conditional probabilities, $P(L|l)$, along with the detailed algorithms. These two approaches handle the situation when the same label is propagated by multiple nodes, with different distances.

4.1 Nearest Probabilistic Association

According to the exponential decay model of transmissibility [34], the transmissibility decays as a power of the distance from the initial source. In the *Nearest Probabilistic Association* model (NPA), the conditional probability $P(L(u)|l)$ of Eq. 1, $A_u(l)$, is defined as follows.

DEFINITION 4 (NEAREST ASSOCIATION). *Let l be a label present in v which is the nearest one to u, where $l \notin L(u)$. $A_u(l) = P(L(u)|l) = e^{-\alpha \cdot d}$, where d is the distance from v to u, and α is the decay constant ($\alpha > 0$).*

$A_u(l)$ decays to zero as d approaches to ∞. For an unweighted graph, we assume $d = 1$ for each edge. The algorithm to find the stable propagated graph $\widetilde{G}$ is outlined in Algorithm 1. $\widetilde{G}$ is like a classical transaction database, where each node represents a transaction and each label represents an item. However, unlike transactions, in $\widetilde{G}$, items could have values 0, 1 or a fraction between them due to the probabilistic property of our model. Similar to classical transactions, 1 denotes full association and 0 no association; whereas a proper fraction indicates partial association of the labels at that vertex. The association value should decrease as the distance between a vertex and a label increases [25, 14]. Therefore, we have an input cut-off parameter ϵ in Algorithm 1. We do not propagate a label when the nearest association value for that label is less than ϵ.

Note that $A_u(l) = 1$ when u itself has the label l; $A_u(l) = 0$ when l is considerably away from u, or there is no path from u to any of the vertices having l. Since the association of a label at a vertex is determined by the nearest occurrence of the label, we call it "nearest association". Once the intermediate dataset is formed, following the joint distribution of Eq. 2, we shall define the support of a proximity pattern.

DEFINITION 5 (PROBABILISTIC SUPPORT). *Given an intermediate dataset $\widetilde{G}$ derived by the Nearest Probabilistic Association model, the support of $I = \{l_1, l_2, \ldots, _m\}$, $sup(I) = \frac{1}{|V|} \sum_{u \in V} A_u(l_1) \cdots A_u(l_m)$, where $A_u(l)$ represents the probability of observing l at u.*

The support definition in NPA has the downward closure property. That is, $sup(I) \geq sup(J)$ if $I \subseteq J$. This is due to the fact that $A_u(l) \leq 1$. Let $I = \{l_1, l_2, \ldots, _m\}$ and $J = \{l_1, l_2, \ldots, l_m, l_{m+1} \ldots, _n\}$. Since

$$\prod_{i=1}^{m} A_u(l_i) \geq \prod_{i=1}^{m} A_u(l_i) \prod_{i=m+1}^{n} A_u(l_i),$$

we have

$$sup(I) \geq sup(J).$$

Algorithm 1 Generate Intermediate Dataset $\widetilde{G}$

Input: Graph G, cut-off parameter ϵ.
Output: Intermediate Dataset $\widetilde{G}$.

1: $i = 0$ // iteration
2: **for all** vertex u of G **do**
3: Let $L_0(u)$ be the label set of u
4: $\forall l \in L_0(u)$, $A_u(l)$=1; otherwise $A_u(l)$=0
5: **end for**
6: **for all** vertex u of G **do**
7: **for all** label l in $L_i(v) \setminus L_i(u)$, v is u's neighbor **do**
8: update $A_u(l)$ using Definition 4 (choose the maximum one)
9: If less than ϵ, do not propagate l to u
10: **end for**
11: $L_{i+1}(u) = \{L_i(u) \cup \{l\} | A_u(l) > 0\}$
12: **end for**
13: **if** $L_{i+1} = L_i$ for all vertices in G **then**
14: Output A_u for all $u \in V(G)$
15: **else**
16: $i = i + 1$, goto step 2
17: **end if**

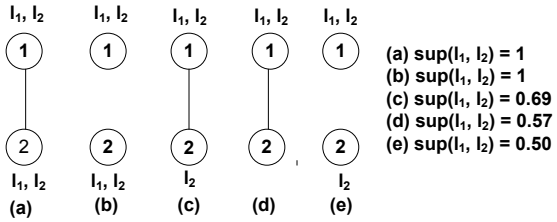


Figure 4: Consistency: NPA and Frequent Itemset

The definition is also consistent with the support definition of frequent itemsets, where $A_u(l)$ can only be 0 or 1. Figure 4 shows the connection, where the decay constant α is set at 1. NPA rightly assigns the highest support value (= 1) for $\{l_1, l_2\}$ in Figure 4(a) and 4(b), which is consistent with frequent itemsets. The support value gradually decreases in Figure 4(c), 4(d), and 4(e). The decreasing order of support reflects the association strength of l_1, l_2 in different structures. Figure 4(c) has a higher support for $\{l_1, l_2\}$ than Figure 4(d) since there are two l_2's close to l_1. Note that, we assume $\alpha = 1$ for all these examples.

The NPA support is both commutative and associative. It can also tell slight difference between structures. Table 1(a) and 1(b) show the intermediate dataset for two different substructures in Figure 5(a) and Figure 5(b) respectively. Rightly this approach assigns higher support for $\{l_1, l_2, l_3\}$ in Figure 5(a).

Complexity. Let $|V|$ be the total number of vertices in G, the average degree of each vertex be d, and the average

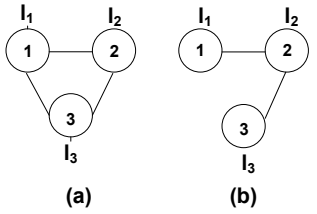


Figure 5: Support vs Structure Difference

Table 1(a)			
	l_1	l_2	l_3
node_1	1	0.37	0.37
node_2	0.37	1	0.37
node_3	0.37	0.37	1
$sup(l_1, l_2, l_3) = 0.14$			

Table 1(b)			
	l_1	l_2	l_3
node_1	1	0.37	0.14
node_2	0.37	1	0.37
node_3	0.14	0.37	1
$sup(l_1, l_2, l_3) = 0.08$			

Table 1: NPA Intermediate Dataset for Fig. 5

number of labels in each vertex be s. If there are total t iterations in Algorithm 1, the time complexity of generating the intermediate dataset $\widetilde{G}$ is $O(|V| \cdot d^t \cdot s)$. Since $t << |V|$, the complexity is almost linear in the number of vertices. The parameter t is a measure of the maximum depth where we may look for a label. The depth will be determined by the decay constant α and ϵ. In social networks, the mutual interaction and social influence usually decays quickly with distance t [18, 10, 27, 1]. The influence is negligible when $t > 3$. In NPA, the probabilistic association value of a label at a distance t is given by $e^{-\alpha \cdot t}$. Since we ignore the value less than ϵ, $t \leq \frac{1}{\alpha} \ln\left(\frac{1}{\epsilon}\right)$.

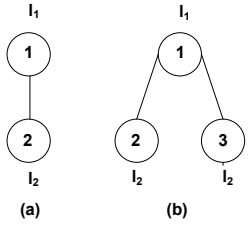


Figure 6: Problem with NPA

The NPA model is fast to calculate. However, there is a potential issue: For each vertex, it only considers the nearest neighbor of each label. Thus it cannot differentiate the situations when there are more than one nearest vertices with the same label. Figure 6 shows two graphs. In both cases, $sup(l_1, l_2) = 0.37$ according to NPA. In order to differentiate them, we propose the second model, Normalized Probabilistic Association, to take into account all the nearest occurrences of the same label.

4.2 Normalized Probabilistic Association

In the normalized probabilistic association model (NmPA), we try to normalize the association by the number of neighbors who have the same label.

Definition 6 (Normalized Association). *Given an attributed unweighted graph G and a node u, if the number of neighbors of u is n and there are m neighbors having the label l, the normalized probabilistic association of l at u is* $NA_u(l) = P(L(u)|l) = \frac{m}{n+1} e^{-\alpha}$.

The normalizing factor $Z = \left(\frac{m}{n+1}\right)$ will give more association strength for the labels that are contained by many neighbors. In order to differentiate the two cases in Figure 6, we choose $n+1$ rather than n as the denominator. Since $NA_v(l) \leq 1$, the downward closure property is maintained. For weighted graphs, the modified version of NmPA will be discussed in Section 4.3.

For an itemset I, the support of I under NmPA could be calculated similar to NPA (see Definition 5), by following the joint distribution in Eq. 2. The supports in NmPA shall be smaller than those in NPA.

NmPA has two advantages over NPA. We have the following lemmas.

Lemma 1. *Given two nodes u and u', assume u and u' have the same number of neighbors, label $l \notin L(u), l \notin L(u')$, we have $NA_u(l) > NA_{u'}(l)$ if more neighbors of u contain l.*

Lemma 2. *Given two nodes u and v, assume u has more neighbors than v, label $l \notin L(u), l \notin L(v)$, we have $NA_u(l) > NA_v(l)$ if the percentage of u's neighbors that contain l is no less than that of v's.*

Lemmas 1 and 2 show that the NmPA could break the tie situations when there is an equal number of neighbors or an equal number of neighbors having l. NmPA favors the case when more neighbors contain l. These are desirable properties over NPA.

For the two substructures shown in Figure 6. The normalized association of l_2 at vertex 1 is $\frac{0.37}{2} \approx 0.19$ for Figure 6(a) and $\frac{0.37 \times 2}{3} \approx 0.25$ for Figure 6(b). Therefore, NmPA assigns a higher support for $\{l_1, l_2\}$ in Figure 6(b) than that in Figure 6(a). It can be verified that the $sup(l_1, l_2)$ values will be $1.0, 1.0, 0.59, 0.52, 0.50$ for the substructures shown in Figure 4(a), (b), (c), (d) and (e) respectively. Therefore, similar to NPA, the NmPA support of $\{l_1, l_2\}$ decrease gradually from Structures (a) to (e) in Figure 4.

Next we apply Algorithm 1, as before, to find the intermediate dataset. The only change will be in Line 8 of Algorithm 1. We shall use the following equation to update the probability,

$$NA_u(l) = \frac{1}{n+1} \sum_{v \in N(u)} e^{-\alpha} * NA_v(l), \qquad (3)$$

where $NA_v(l)$ is the association strength of l at v and $N(u)$ is the neighbor set of u. Since l could be a label propagated from another vertex, $NA_v(l)$ could be less than 1.

Complexity. NmPA has the same complexity as NPA. However, in practice the propagation decays much faster since we normalize the probabilistic association with respect to the number of neighboring nodes at every iteration. Using NPA and NmPA, the set of all proximity patterns can be determined efficiently from the intermediate dataset, as they follow the downward closure property. In addition to NPA and NmPA, other influence models could be adapted here. As long as the probabilistic association is calculated by Definition 5, the same mining algorithm could be applied.

4.3 Modification for Weighted Graphs

It is easy to verify that NPA and NmPA are also applicable to weighted graphs. In NPA, we consider only the nearest vertex of any label among all the neighboring vertices. Suppose, the nearest occurrence of a label l is at distance d from vertex u. Then, the NPA probabilistic association of l at u is given by $N_u(l) = e^{-\alpha \cdot d}$. If there are total n neighbors from vertex u, and among them m neighbors, each at distance d_i from u, have the label l. The NmPA probabilistic association of l at u is given by $NA_u(l) = \sum_{i=1}^{m} e^{-\alpha \cdot d_i}/(n+1)$. The procedure of generating the intermediate dataset remains the same.

5. PROBABILISTIC ITEMSET MINING

Given an attributed graph G, the proposed information propagation models such as NPA and NmPA will generate a large set of probabilistic itemsets, whose number is equal to $|V(G)|$. Each itemset has tuples $\langle I, A_{id}(I)\rangle$, where id is the vertex id and $A_{id}(I)$ is the probabilistic association of label I to this vertex. To be consistent with the terminology used in frequent itemset mining, we also call vertex as transaction.

In the existing frequent itemset mining algorithms such as Apriori [5] and FP-growth [4, 16], the support of an itemset is the number of occurrences of all the items together in that itemset. Our problem setting is inherently different since it has to multiply the fractional support values of all the constituent items to determine the joint support of an itemset. In the following discussion, we will first describe an algorithm to mine all the proximity patterns from the intermediate dataset generated by the NPA or NmPA model described earlier. Next, we provide an approximate version that will improve efficiency and reduce memory consumption. Finally, in addition to the support definition, we introduce an objective function to measure the "interestingness" of a proximity pattern and make the algorithm more efficient and effective to generate only the top-k interesting patterns.

5.1 Exact Mining

Algorithm 2 describes an exact mining algorithm, called pFP(Probabilistic FP-Growth). pFP is derived from FP-Growth in [16]. It first removes the infrequent 1-itemsets and constructs the FP-tree, where transactions share the same upper path if their first few frequent items are the same. We briefly introduce FP-tree here. For details, readers are referred to [16]. FP-tree is a prefix tree. The root of an FP-tree is a NULL node, since each transaction can be prefixed by a NULL item. In the original FP-growth algorithm, each node v in the tree is labeled by an item I and also associated with a count, denoted by $count(v)$, representing the number of transactions that pass through the node. At the same time, a header table is built. For an entry $(I, H(I), ptr)$ in the header table, $H(I)$ denotes the count of nodes in FP-free containing the item I and ptr records the list of nodes containing the item I. This is also known as the *side-link* of I. Now, for each frequent length-1 pattern I present in the header table, the following technique is applied. The FP-growth algorithm starts from a frequent length-1 pattern, say I, and for each node u attached to the side-link of I, it follows the path till the root of the tree. These paths are called the conditional pattern base of I. Then, an FP-tree on this conditional pattern base (conditional FP-tree) is constructed, which acts as a transaction database with respect to I. Next, the algorithm recursively mines this resulting FP-tree to form all possible combinations of itemsets prefixed with I.

Algorithm 2 pFP: Probabilistic Itemset Mining

Input: Intermediate transaction dataset; and the minimum support threshold: minsup.
Output: frequent itemsets above the minsup.
Method: build the FP-tree; then call $mine\_tree(\emptyset, H)$
procedure $mine\_tree(X, H)$
1: **for all** entry I (top down order) in H **do**
2: **if** $[\frac{H(I)}{DBSIZE}] \geq minsup$ **then**
3: output $\{I\} \cup X$;
4: create a new header table H_I by calling $build\_subtable(I)$;
5: $mine\_tree(\{I\} \cup X, H_I)$;
6: **end if**
7: **end for**
procedure $build\_subtable(I)$
1: **for all** node v on the side-link of I **do**
2: walk up the path from v to the root once;
3: **if** encounter a node u with label J **then**
4: add/update the entry for J in H_I as below:
5: insert u as a side-link of J for that entry;
6: $H_I(J) = H_I(J) + \sum_{id \in \mathcal{B}(v) \cap \mathcal{B}(u)} \{A_{id}(J) \cdot A_{id}(I)\}$;
7: add $\{id, A_{id}(J) \cdot A_{id}(I)\}$ in $\mathcal{B}(vu)$ for all $id \in \mathcal{B}(v) \bigwedge \mathcal{B}(u)\}$;
8: **end if**
9: **end for**

In our problem setting, labels are probabilistic and we need to multiply these probabilistic association values to determine the joint association of multiple labels. To handle probabilistic itemsets, in our algorithm, each node v in the FP-tree is associated with a bucket $\mathcal{B}(v)$ consisting of the probabilistic association values of all single items contained in that node, which is a set of tuples $\langle id : A_{id}(I)\rangle$, where v is in the side-link of item I and id is the transaction id contained in v. The buckets can be stored in the disk and accessed when the corresponding nodes are processed. As we move up the tree, the buckets corresponding to the composite itemsets in the sub-header table can be formed recursively by the intersection of the buckets of its constituent items. For example, consider an intermediate dataset given in Table 2. Assume, the minimum support threshold is set at 0.06. Thus, the infrequent item l_5 having support=$0.15/3 = 0.05$ can be removed first. The remaining items are then arranged in a decreasing order of their frequency as l_2, l_1, l_3, l_4. The corresponding FP-tree is depicted in Figure 7. Note that, although l_1 has higher support than that of l_2, it is placed below l_2 in the FP-tree, since frequency of l_1 is lower than that of l_2 in the dataset.

transaction id	l_1	l_2	l_3	l_4	l_5
1	1	0.3	0	0	0.1
2	0.5	0.2	0.5	1	0
3	0	0.2	0.5	0	0.05

Table 2: Intermediate Dataset

The exact algorithm is given in Algorithm 2. Here, $H(I)$

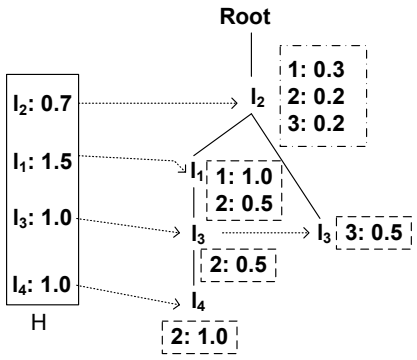


Figure 7: FP-Tree for Table 2

in the header table denotes the sum of the probabilistic association values for item I. For example, while processing l_3 from the original header table H, we start moving upwards from l_3 following two distinct paths $l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow root$ and $l_3 \rightarrow l_2 \rightarrow root$. The first node encountered is l_1, so it will be added in the sub-header table H_{l_3} of l_3 with $H_{l_3}(l_1) = 0.5 \times 0.5 = 0.25 > minsup \times DBSIZE$ (see Figure 8). So, $l_3 l_1$ is a frequent pattern. The corresponding bucket will contain only the entry $2 : 0.25$, which can be formed as an intersection of buckets of l_3 and l_1. The algorithm now recursively considers the sub-header table $H_{l_3 l_1}$ by moving upward from l_1 along the path $l_1 \rightarrow l_2 \rightarrow root$. It calculates $H_{l_3 l_1}(l_2) = 0.25 \times 0.2 = 0.05 < minsup \times DBSIZE$. So, $l_3 l_1 l_2$ is not a frequent pattern. Now, the control comes back to H_{l_3}, where the next entry is l_2, with $H_{l_3}(l_2) = 0.5 \times 0.2 + 0.5 \times 0.2 = 0.2 > minsup \times DBSIZE$. So, $l_3 l_2$ is also frequent. Its bucket will contain two entries $2 : 0.1$ and $3 : 0.1$; which can be determined by the intersection of buckets of l_3 and l_2. Note that, it cannot be extended further and this also finishes the processing of l_3 from the original header table H. So, the algorithm starts processing l_4, which is next to l_3 in H.

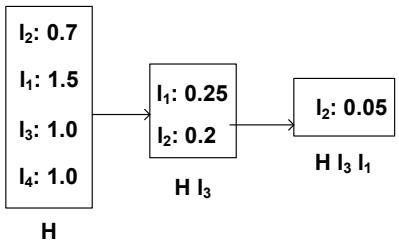


Figure 8: pFP applied on Table 2

The problem with the exact algorithm (Algorithm 2) is that, the running time increases compared to that of the original FP-growth algorithm [33], because we need to access the bucket whenever the corresponding node is processed. However, the arguments in support of this exact algorithm can be as follow.

1. Each bucket size is small compared to the original intermediate dataset size. Therefore, pFP is still efficient compared to apriori based approaches, where the whole dataset needs to be scanned every time.
2. During the execution of pFP, we need only two recent buckets in the main memory. Therefore, the buckets used before can be removed from the main memory. Since the buckets for the composite itemsets are formed by intersection of its constituent itemsets, the bucket of a large itemset usually gets smaller than that of its constituent itemsets.

5.2 Approximate Mining

The exact mining algorithm needs to maintain a bucket with a list of transaction ids, since we have to multiply the fractional association values to determine the joint support of multiple items. However, is it possible to compress buckets so that they can be accommodated in the main memory along with the FP-tree? The compact representation of buckets must be sufficient enough to generate an approximation to the joint association of multiple items. Here, we propose that, instead of maintaining a long bucket list of $\langle id, A_i d(l) \rangle$ for each node in the FP-tree, we can associate two variables, *sum* and *occurrence*, with each node.

Suppose l_x and l_y are two distinct labels appearing at nodes v_x and v_y respectively in the FP-tree, where v_x is the parent of v_y. Let, the buckets $\mathcal{B}(v_x)$ and $\mathcal{B}(v_y)$ in the exact algorithm have the association values $A_1(l_x) = x_1, A_2(l_x) = x_2, \ldots, A_n(l_x) = x_n$ and $A_1(l_y) = y_1, A_2(l_y) = y_2, \ldots, A_n(l_y) = y_n$ for transactions $1, 2, \ldots, n$ respectively. Note that some of y_i can be zero. We define *sum* for v_x and v_y as $sum(v_x) = \sum_{i=1}^{n} x_i$ and $sum(v_y) = \sum_{i=1}^{n} y_i$. The variable *occurrence* is defined as the number of all non-zero occurrences of that label in the corresponding node of the FP-tree. Clearly, $occurrence(v_x) = n$ and $occurrence(v_y) \leq n$. The approximate algorithm associates these two variables *sum* and *occurrence* with each node while forming the FP-tree. Now, we define the *approximate joint association* of l_x and l_y as given in Eq. 4.

$$\begin{aligned} \widetilde{A}(l_x, l_y) &= \frac{sum(v_x) \cdot sum(v_y)}{max\{occurrence(v_x), occurrence(v_y)\}} \\ &= \frac{1}{n} \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i \end{aligned} \tag{4}$$

The exact joint association of l_x and l_y is given by $A(l_x, l_y) = \sum_{i=1}^{n} \{x_i \cdot y_i\}$. Therefore, the absolute error E due to the approximation can be expressed as follow.

$$\begin{aligned} E &= A(l_x, l_y) - \widetilde{A}(l_x, l_y) \\ &= \sum_{i=1}^{n} [x_i \cdot (\frac{sum(v_y)}{n} - y_i)] \\ &= \sum_{i=1}^{n} [y_i \cdot (\frac{sum(v_x)}{n} - x_i)]. \end{aligned} \tag{5}$$

The error E is small compared to $A(l_x, l_y)$ when all the x_i's or all the y_i's are very close to each other. For example, if we consider the nodes of the FP-tree corresponding to l_2 and l_1 in Figure 7, the exact joint association $A = 0.40$, whereas the approximate joint association $\widetilde{A} =$

0.35, and the absolute error $E = 0.05$. Using this summarization technique, we develop aFP(approximate probabilistic FP-Growth), an approximation to pFP. Only the *build_subtable* procedure needs to be changed from the exact mining algorithm described earlier (see Algorithm 2). The new *build_subtable* procedure is given in Algorithm 3.

Algorithm 3 aFP, Approximate Itemset Mining

procedure *build_subtable*(I)

1: **for all** node v on the side-link of I **do**
2: walk up the path from v to root once;
3: **if** encounter a node u with label J **then**
4: add/update the entry for J in H_I as below:
5: insert u as a side-link of J for that entry;
6: calculate $\widetilde{A}(u, v)$, the approximate joint association of nodes u and v as mentioned in Section 5.2;
7: $H_I(J) = H_I(J) + \widetilde{A}(u, v)$;
8: $sum(vu) = \widetilde{A}(u, v)$;
9: $occurrence(vu) = occurrence(u)$;
10: **end if**
11: **end for**

5.3 Top-k Interesting Patterns

The support value defined by our probabilistic association model only tells the association strength of a proximity pattern in a given graph. In order to measure its real "interestingness", we need to compare the support value with the one generated by a randomization test.

Randomization Test. Given an attributed graph G, where each node has a set of labels, we conduct the following random permutation: Randomly select two nodes u, v and one of their labels, l_u, l_v, respectively, then swap these two labels so that u has l_v attached, and v has l_u attached. The permutation is repeated until all the labels are swapped. Let Q be the result graph.

Assume that p and q be the support value of an itemset I in G and Q respectively, using our probabilistic association model. If I is not found in the permutated graph Q, i.e., $q = 0$, we replace q with the product of support values of all its constituent labels. Now, we consider I as *interesting* if the difference between p and q is high. Note that, the higher difference between p and q indicates that the individual items in I truly formulate a pattern. If we only consider the p value of I, it might be high since some of its members occur very frequently, in which case, q value will also be high. Thus, by considering the difference between p and q, we can eliminate those uninteresting patterns from the result set. We propose to apply G-test score [31] as an objective function to measure the interestingness of a pattern.

$$p \cdot \ln \frac{p}{q} + (1 - p) \cdot \ln \frac{1 - p}{1 - q} \tag{6}$$

We developed a pruning method similar to the vertical pruning approach proposed by Yan et al. [35] and integrate it with pFP and aFP to mine interesting patterns using the probabilistic FP-tree built from G and Q.

Proximity Patterns vs Frequent Itemsets. It is also possible to compare the proximity patterns mined from a graph G with the frequent itemsets mined from the node label sets if one ignores the connection between nodes. One can run the above test by replacing q with the support of frequent itemsets. The result will tell the new patterns that are missed by the classic frequent itemset mining approaches. In the experiment section, we will demonstrate such patterns.

6. EXPERIMENTAL RESULTS

In this section, we present experimental results which illustrate the effectiveness of the information propagation model NmPA and the efficiency of our approximate itemset mining framework aFP on a number of real-life graph datasets. We are not going to experiment neighbor association model due to its time complexity. In order to evaluate the effectiveness, we report the top-k interesting patterns discovered by our approaches. We shall also analyze the effectiveness and efficiency of the approximate itemset mining algorithm (aFP) over the exact one (pFP). Finally, we provide a comparison of our result with that of frequent itemset and subgraph mining. The experiments are performed using a single core in a 32GB, 2.50GHz Xeon server.

6.1 Graph Datasets

Our models and mining algorithms are tested on a variety of real graph data sets including Last.fm, Intrusion network, and DBLP.

LAST.FM We crawled a local network consisting of $6,899$ users from www.last.fm. Last.fm is a music web site where users listen to their favorite tracks and communicate with each other based on their choice of music. For each user, we crawled the most recent communications among them. These communications recommend songs. We treat them as edges. There are total $58,179$ edges. For each user, we also crawled the name of 3 artists (or musical bands) of the most recently listened tracks by that user. There are total $6,340$ artists and musical bands crawled. We mined proximity patterns among these artists and musical bands by using the social network graph that we built.

Intrusion Alert Network This network contains the anonymous log data of intrusion alerts in a computer network. It has $200,858$ nodes and $703,020$ edges where each node is a computer and an edge means a possible attack such as Denial-of-Service and TCP Service Sweep. Each node has 25 labels (computer generated alerts in this case) on average. There are around $1,000$ types of alerts. We aim to find the association of alerts in this graph data, which could reveal multi-step intrusions.

DBLP Collaboration Graph The DBLP graph is downloaded from www.informatik.uni-trier.de/∼ley/db/. There are $684,911$ distinct authors and $7,764,604$ collaboration edges among them. We consider the keywords present in the paper titles as the labels corresponding to these authors. We select 130 important keywords to determine the association among them. Each node has around 9 labels on average in this graph.

6.2 Effectiveness

We present the top-5 interesting patterns for the **Last.fm** data set in Table 3. We applied the NmPA propagation model and the aFP mining algorithm. For the NmPA model, we set the decay constant $\alpha = 1$ and cut-off parameter $\epsilon =$

#	Proximity Patterns	Score
1	Tiësto, Armin van Buuren , ATB	0.62
2	Katy Perry, Lady Gaga, Britney Spears	0.58
3	Ferry Corsten, Tiësto, Paul van Dyk	0.55
4	Neaera, Caliban, Cannibal Corpse	0.52
5	Lacuna Coil, Nightwish, Within Temptation	0.47

Table 3: Top-5 Proximity Patterns (Last.fm)

0.12. These parameters ensure that we propagate a label at most two hops. A label is propagated to a node only when at least one third of its immediate neighbors contain that label. The patterns are ranked by the G-test score defined in Eq. 6. Also, we report only the top-5 patterns after eliminating their smaller sub-patterns.

These patterns are practically interesting, i.e., *ATB* and *Paul van Dyk* are popular German DJ; whereas *Tiësto, Ferry Corsten and Armin van Buuren* are Dutch trance producers and DJ. *Britney Spears, Lady Gaga, Katy Perry* are American female pop singers and entertainers. *Lacuna Coil, Nightwish and Within Temptation* are Gothic metal bands from Italy, Finland and Netherlands respectively. *Neaera* and *Caliban* are death metal bands from Germany; while *Cannibal Corpse* is an American death metal band.

#	Proximity Patterns	Score
1	Tiësto, Armin van Buuren , ATB	0.62
2	Katy Perry, Lady Gaga, Britney Spears	0.58
3	Ferry Corsten, Tiësto, Paul van Dyk	0.55
4	Neaera, Caliban, Cannibal Corpse	0.52
5	Lacuna Coil, Nightwish, Within Temptation	0.47

Table 4: Proximity Patterns minus Frequent Itemsets (Last.fm)

Table 4 illustrates the proximity patterns discovered by our algorithms but ranked low by the classic frequent itemset mining algorithm. It shows that the top-5 patterns in Table 3 and 4 are the same. That is, none of these top-5 'interesting' patterns are reported by the classical frequent itemset mining algorithm, since the items of these patterns do not co-occur frequently in individual nodes.

#	Interesting Patterns	Score
1	Ping_Sweep, Smurf_Attack	2.42
2	TFTP_Put, Audit_TFTP_Get_Filename, ICMP_Flood, Ping_Flood	2.32
3	TCP_Service_Sweep, Email_Error	1.21
4	HTML_Outlook_MailTo_Code_Execution, HTML_NullChar_Evasion	1.15
5	SQL_SSRP_Slammer_Worm, SQL_SSRP_StackBo	0.88

Table 5: Top-5 Proximity Patterns (Alerts)

The top-5 proximity patterns for the **Intrusion Network** data set are given in Table 5. The first one describes a Smurf denial of service attack. The ICMP echo request (Ping) packets addressed to an IP broadcast address cause a large number of responses, which might consume all available network bandwidth. The second one describes a TFTP (Trivial File Transfer Protocol) attack, which allows remote users to write files to the target system without any authentication. The fifth one is an attack to Microsoft SQL Server 2000 which is vulnerable to a stack-based buffer overflow in the SQL Server Resolution Service. The discovered proximity patterns show that multiple attacks are often coupled together to complete one intrusion.

#	Interesting Patterns	Score
1	ICMP_Flood, Ping_Flood	0.94
2	Email_Error, SMTP_Relay _Not_Allowed, HTML_Null Char_Evasion	0.94
3	Image_RIFF_Malformed, HTML_NullChar_Evasion	0.90
4	TFTP_Put, Ping_Flood, Audit_TFTP_Get_Filename	0.80
5	Email_Command_Overflow, Email_Virus_Double_Extension, Email_Error	0.75

Table 6: Proximity Patterns minus Frequent Itemsets (Alerts)

Table 6 illustrates the proximity patterns discovered by our algorithms but ranked low by the classic frequent itemset mining algorithm on the intrusion network dataset. The first one is related to ICMP DOS Attack. The second one could be triggered by spammers who use an open relay to send unsolicited email to a number of email accounts. The fifth one could indicate an attacker's attempt to overflow a buffer using a command that is longer than 512 characters.

#	Interesting Patterns	Score
1	Association, Rules, Mining	1.17
2	Distributed, Network, Architecture	0.84
3	Sensor, Video, Network	0.80
4	Channel, Allocation, Network	0.67
5	Vector, Machine	0.45

Table 7: Top-5 Proximity Patterns (DBLP)

Table 7 shows the top-5 interesting patterns mined from the **DBLP** data set. Itemsets 1 and 5 are related to Data Mining and Machine Learning. Itemsets 2 is from distributed systems. The remaining patterns are from sensor and network fields.

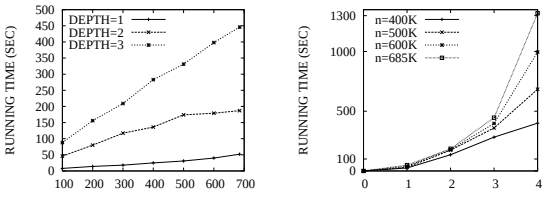
6.3 Efficiency and Scalability

Steps	**Last.fm**	**Intrusion**	**DBLP**
NmPA	2.0	5.0	187.0
FP-tree Formation	1.0	10.0	89.0
Top-k Pattern Mining	4.0	2.0	254.0

Table 8: Runtime (sec)

We present the running time for our algorithms on the three above mentioned data sets in Table 8. It can be observed that each component runs pretty fast. For example, the *DBLP* collaboration graph with about 0.7 million nodes requires less than 9 minutes to be processed.

Next, we analyze the influence of different parameters on the running time of information propagation, FP-tree building and Top-k pattern mining. We use the **DBLP** graph for these experiments. Figure 9(a) shows the variation of running time with respect to the number of nodes present in the graph. In order to vary the number of nodes, we randomly delete some nodes and the corresponding edges from the graph. We set the decay constant $\alpha = 1$ and vary the depth of propagation from 1 to 3 for this experiment. The cut-off parameter is set at $\epsilon = 0.36, 0.12$ and 0.04 respectively. Figure 9(a) shows that the $NmPA$ running time increases linearly with the increasing number of nodes. However, the slope of the lines increases as we increase the depth of propagation.



(a) # of Nodes (b) Propagation Depth

Figure 9: NmPA Time (DBLP)

Figure 9(b) shows the variation of running time with respect to the propagation depth using $NmPA$ method. We set $\alpha = 1$. In order to achieve a propagation depth at 1, 2, 3 and 4, we set the cut-off parameter $\epsilon = 0.36$, 0.12, 0.04 and 0.01 respectively. Experiments are performed for different values of n. Figure 9(b) shows that the NmPA running time increases exponentially with the increasing depth of propagation; which can be explained as the number of h-hop neighbors increases exponentially as we increase the depth h.

Next, we show the variation of running time with respect to the total number of labels in Figure 10. We use the complete **DBLP** graph and labels are selected randomly for this experiment. Similar to the previous case, we set the decay constant $\alpha = 1$ and vary the depth of propagation from 1 to 3. It can be observed that the running time of NmPA on the **DBLP** data set increases linearly with the increasing number of labels.

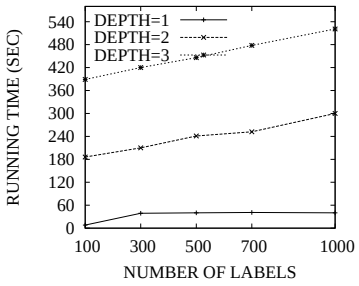


Figure 10: NmPA Time vs. # of Labels (DBLP)

In Figure 11, we analyze the running time of FP-tree formation and top-k pattern mining using aFP with respect to the number of nodes. The propagation is done using NmPA with $\alpha = 1$ and $\epsilon = 0.12$. Note that, as we increase the number of nodes, the running time for the FP-tree formation increases almost linearly. However, the running time for mining levels off after a certain value of n. This can be explained as follow. If there are s labels, each transaction requires at most s scans to form the complete FP-tree. So, the time complexity of building the FP-tree is almost linear in the number of nodes present in the graph. However, once the FP-tree is built, the mining depends on the size of the FP-tree and not on the actual size of the database. Hence, the running time for mining levels off after a certain value of n.

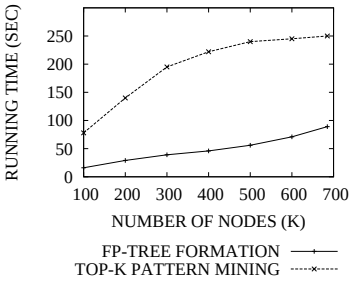


Figure 11: Mining Time vs. # of Nodes (DBLP)

In Figure 12, we plot the running time of FP-tree formation and top-k pattern mining using aFP with respect to the number of labels. Note that, the latter increases at a higher rate compared to the former as we increase the number of labels.

6.4 Exact vs. Approximate Mining

We compare the effectiveness of our two mining algorithms, i.e. pFP (exact) and aFP (approximate) on **Last.fm** data set. Table 9 reports the top-5 proximity patterns minus frequent itemsets reported by the pFP mining algorithm. If we compare these patterns with those reported by the aFP in Table 4, the top-5 patterns remain the same. Only the score values differ slightly and therefore, the rank is a little bit different for some patterns. However, if we consider the running times given in Table 10, it is easy to conclude that the aFP is very efficient compared to the pFP . Moreover, this difference in running time grows very fast as the size of the database increases. For the **DBLP** graph data, the pFP mining algorithm requires about **8 hours**, whereas aFP reports the top-k patterns in less than **9 minutes**.

#	Proximity Patterns	Score
1	Katy Perry, Lady Gaga, Britney Spears	0.58
2	Ferry Corsten, Tiësto, Paul van Dyk	0.55
3	Tiësto, Armin van Buuren, ATB	0.55
4	Neaera, Caliban, Cannibal Corpse	0.51
5	Lacuna Coil, Nightwish, Within Temptation	0.46

Table 9: Proximity Patterns minus Frequent Itemsets using Exact Mining Algorithm (Last.fm)

6.5 Frequent Subgraph Mining

Subgraph patterns are a subset of proximity patterns, if we collapse their structure. For Last.Fm, the top-5 significant patterns discovered by LEAP search [35], are given

Steps	aFP(approximate)	pFP(exact)
FP-tree Formation	1.0	3.0
Top-k Pattern Mining	4.0	21.0

Table 10: Runtime Comparison (sec) (Last.fm)

in Table 11. These top-5 patterns are also discovered by our probabilistic association method. If the support threshold is set at 1%, there are 67 frequent subgraphs, while our approach discovers $5,444$ proximity patterns. If we raise the support threshold further, our approach could still find interesting patterns, while the existing subgraph mining algorithm cannot. For Last.Fm($\approx$ 6K nodes), the running time of subgraph mining is comparable with ours. But for the Intrusion Alert Network ($\approx$ 200K nodes), it needs about 4 hours, while our algorithm terminates within 17 seconds. Our approach avoids subgraph isomorphism testing.

#	LEAP Patterns
1	Nirvana, Arctic Monkeys, Muse
2	Radiohead, Arctic Monkeys, Muse
3	Red Hot Chili Peppers, Arctic Monkeys, Metalica
4	Radiohead, Placebo, Depeche Mode
5	Radiohead, Cold Play, Arctic Monkeys

Table 11: Significant Patterns via LEAP (Last.fm)

7. RELATED WORK

Finding graph patterns is an active research topic in data mining. In the area of mining a set of graphs, efficient frequent subgraph mining algorithms have been proposed, including AGM [22],FSG [23], gSpan [36], followed by PathJoin, MoFa, FFSM, GASTON, etc. Recently, techniques were developed to mine maximal graph patterns [21] and significant graph patterns [17]. These methods adopt subgraph isomorphism testing as a way to count the support of graph patterns in multiple graphs.

In the area of mining single massive graphs, [24, 11, 15] developed techniques to calculate the support of graph patterns, i.e., how many times we should count a subgraph in one graph, when there are overlapping embeddings. Kuramochi and Karypis [24] proposed using the maximum independent set as the support of subgraphs, which is proved to have the downward closure property by [15]. [7] proposed a support measure that is computationally less expensive and often closer to intuition than other measures. Since subgraph isomorphism is still used in these methods, they cannot handle the proximity patterns discussed in this work, where strict isomorphism is not desired.

Discovering rules from transactions has been extensively studied. The concept of association rules was first introduced in [3, 5], where the authors proposed an Apriori based approach to determine all frequent itemsets. [30] describes a hash-based algorithm which is an improvement over the Apriori approach. In [39], Zaki proposed a depth-first search algorithm using set intersection. FP-growth was introduced by Han et al. in [16], which uses an extended prefix-tree (FP-tree) structure to store the database in a compressed form. In [6], Au and Chan introduced fuzzy association rules based on the fuzzy set theory. Here, each item is assigned a non-binary weight according to its significance with

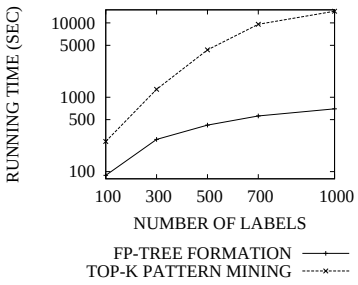


Figure 12: Mining Time vs. # of Labels (DBLP)

respect to a user defined criterion. In [26], Mangalampalli and Pudi have shown how the existing algorithms like Apriori and FP-growth can be modified to mine data in a fuzzy environment. Very recently, Bernecker et al. [8] and Charu et al. [2] proposed techniques for mining frequent itemsets from uncertain databases. Their techniques could also be applied. However, to the best of our knowledge, no previous work targets the problem of finding proximity patterns in the context of massive graphs.

8. CONCLUSIONS

We introduced a new pattern concept in graphs - proximity pattern, which is a significant departure from the traditional concept of frequent subgraphs and frequent itemsets. Proximity pattern blurs the boundary between itemset and structure. It relaxes the rigid structure constraint of frequent subgraphs, while introducing structure association to frequent itemsets. We discussed the weakness of a neighbor association model and proposed an information propagation model that is able to transform a complex mining problem to a simplified weighted itemset mining problem, which was solved efficiently by a modified FP-tree algorithm. Furthermore, for the discovered patterns, we defined an objective function that could measure their interestingness using randomization test. In summary, we proposed a complete pipeline to define and mine novel proximity patterns in massive graphs in a scalable manner. This pipeline was evaluated on real-life social and intrusion networks. Empirical results show that it not only finds interesting patterns that are ignored by the existing approaches, but also achieves high performance for finding proximity patterns in large-scale graphs.

9. ACKNOWLEDGEMENTS

Research was sponsored in part by the U.S. National Science Foundation under grant IIS-0847925, and by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 (NS-CTA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

10. REFERENCES

[1] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

[2] C. C. Aggarwal, Y. Li, J. Wang, and J. Wang. Frequent pattern mining with uncertain data. In *KDD*, pages 29–38, 2009.

[3] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. Knowledge and Data Engineering*, 5:914–925, 1993.

[4] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 69–84, 1993.

[5] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.

[6] W. Au and K. C. C. Chan. Farm: A data mining system for discovering fuzzy association rules. In *FUZZY-IEEE*, pages 1217–1222, 1999.

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

[8] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Zuefle. Probabilistic frequent itemset mining in uncertain databases. In *KDD*, pages 119–128, 2009.

[9] C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *ICDM*, pages 211–218, 2002.

[10] N.R. Buchan, E.J. Johnson, and R.T.A. Croson. Lets get personal: An international examination of the influence of communication, culture and social distance on other regarding preferences. *Journal of Economic Behavior and Organization*, 60(3):373–398, 2006.

[11] J. Chen, W. Hsu, M.-L. Lee, and S.-K. Ng. NeMoFinder: Dissecting genome-wide protein-protein interactions with meso-scale network motifs. In *KDD*, pages 106–115, 2006.

[12] J. Cheng, Y. Ke, W. Ng, and A. Lu. FG-Index: Towards verification-free query processing on graph databases. In *SIGMOD*, 2007.

[13] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. on Knowledge and Data Engineering*, 17:1036–1050, 2005.

[14] S. Feld. The focused organization of social ties. *American Journal of Sociology*, 86:1015–1035, 1981.

[15] M. Fiedler and C. Borgelt. Support computation for mining frequent subgraphs in a single graph. In *MLG*, pages 25–30, 2007.

[16] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, pages 1–12, 2000.

[17] H. He and A. Singh. Efficient algorithms for mining significant substructures in graphs with quality guarantees. In *ICDM*, page 163Ű172, 2007.

[18] E. Hoffman, K. McCabe, K. Shachat, and V. Smith. Social distance and other-regarding behavior. *American Economic Review*, 86:653–666, 1996.

[19] L. B. Holder, D. J. Cook, and S. Djoko. Substructure discovery in the subdue system. In *KDD*, pages 169–180, 1994.

[20] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining spatial motifs from protein structure graphs. In *RECOMB*, pages 308–315, 2004.

[21] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: Mining maximal frequent subgraphs from graph databases. In *KDD*, pages 581–586, 2004.

[22] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23, 1998.

[23] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.

[24] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. In *SDM*, 2004.

[25] P. Lazarsfeld and R. Merton. Friendship as a social process: A substantive and methodological analysis. In *Freedom and Control in Modern Society*, pages 18–66. Van Nostrand, 1954.

[26] A. Mangalampalli and V. Pudi. Fuzzy logic-based pre-processing for fuzzy association rule mining. Technical report, International Institute of Information Technology Hyderabad, India, 2008.

[27] S. Milgram. The small world problem. *Psychology Today*, 6:62–67, 1967.

[28] S. Nijssen and J. Kok. A quick start in frequent structure mining can make a difference. In *KDD*, pages 647–652, 2004.

[29] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. NY: McGraw Hill, 1991.

[30] J. S. Park, M. S. Chen, and P. S. Yu. Mining association rules with adjustable accuracy. In *IBM Research Report*, 1995.

[31] R. Sokal and F. Rohlf. *Biometry: the principles and practice of statistics in biological research*. W. H. Freeman and Co., 1994.

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

[33] K. Wang, L. Tang, J. Han, and J. Liu. Top down fp-growth for association rule mining. In *PAKDD*, pages 334–340, 2002.

[34] F. Wu, B. A. Huberman, L. A. Adamic, and J. R. Tyler. Information flow in social groups. *Physica A*, 337:327–335, 2004.

[35] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by leap search. In *SIGMOD*, pages 433–444, 2008.

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

[37] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004.

[38] Guizhen Yang. Computational aspects of mining maximal frequent patterns. *Theor. Comput. Sci.*, 362(1):63–85, 2006.

[39] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowledge and Data Engineering*, 12:372–390, 2000.

[40] P. Zhao, J. Yu, and P. Yu. Graph indexing: tree + delta >= graph. In *VLDB*, pages 938–949, 2007.