RAM: Randomized Approximate Graph Mining

Shijie Zhang and Jiong Yang

EECS Department
Case Western Reserve Univ.,
10900 Euclid Ave, Cleveland OH 44106, USA
shijie.zhang@cwru.edu, jiong.yang@cwru.edu

Abstract. We propose a definition for frequent approximate patterns in order to model important subgraphs in a graph database with incomplete or inaccurate information. By our definition, frequent approximate patterns possess three main properties: possible absence of exact match, maximal representation, and the Apriori Property. Since approximation increases the number of frequent patterns, we present a novel randomized algorithm (called RAM) using feature retrieval. A large number of real and synthetic data sets are used to demonstrate the effectiveness and efficiency of the frequent approximate graph pattern model and the RAM algorithm.

1 Introduction

A large number of algorithms have been developed for mining exact graph patterns [7] [8] [9] [11] [18] [3], [12], [17], [5], [16]. However, in many important applications, the relationships modeled by edges may be inaccurate or incomplete. This is especially true in bioinformatics and social network analysis. Many types of biological data are known to be inaccurate, e.g., gene expression profiles, protein interaction networks, metabolic pathways. Besides, when describing networks of human social interactions, some relationship can be dual, e.g., two people can be both friends and enemies. Last but not least, approximate graph mining can be applied in procedure dependency graphs to discover neglected conditions, e.g., missing paths, conditions, and cases in the field of software engineering [2].

One example of such data is protein interaction networks. A challenging technical problem described in a recent review [1] is the prevalence of spurious interactions due to self-activators, abundant protein contaminants and weak, nonspecific integrations. Analysis based on the agreement of the interaction and expression data shows that less than half of these interactions are biologically relevant. Therefore one might observe an interaction among two proteins that are not functionally related in the cell at all. Due to the fact that various experiments use different confidence thresholds, there also exist a large number of false negative interactions.

Another important application of approximate graph mining lies in the study of cross-market customer segmentations [14]. In a specific market, the similarity among customers in market behavior can be modeled as a similarity graph. Each customer is a vertex in the graph, and two customers are connected by an edge if their behaviors in the market are similar. While we can identify customer types easily, the similarity level between customers in different markets is more difficult to evaluate. Thus, there may exist both false positive and false negative edges.

Due to edge distortion, exact graph mining algorithms may not find important patterns even when the support threshold is low. Therefore, in this paper, we first present a formal definition of frequent approximate patterns, which can be applied in graph databases when the edge relationships are less reliable. Then, we introduce our approximate graph mining algorithm.

One contribution of the paper is the formal definition of frequent approximate patterns. We propose two constraints on such a pattern: (1) the pattern should approximately be embedded in at least a certain number of graphs in the database; (2) the occurrence of each edge in the pattern should be higher than a minimum threshold. The second constraint gives users the ability to keep unrelated edges from affecting the frequency of patterns due to approximation.

Compared to exact graph pattern mining, approximate patterns tend to be larger, i.e., with more vertices and more edges. This adds two additional challenges for mining approximate graphs on top of the exact graph mining problems. First, with larger patterns, the total number of patterns increases. This can greatly impact the efficiency of any pattern mining algorithm. Secondly, canonical forms are commonly used for graph isomorphism tests. When the graph patterns grow large, it becomes extremely inefficient to compute the canonical forms and storing all the canonical forms as strings may lead the program to run out of memory.

With these two challenges in mind, we introduce a randomized algorithm, called RAM, based on feature retrieval. RAM mines frequent graph patterns in a depth first fashion. Instead of using canonical forms for isomorphism tests, we construct a vector of hash values for features. It greatly improves the efficiency and scalability of the mining process. However, it may produce false negatives, missing some patterns. To overcome this problem, we adopt a randomized algorithm. During each run, the insertion of edges and addition of vertices are ordered randomly. As a result, if during a run a pattern could be discovered with x probability, then by p runs, the probability to find that given pattern would be $1-(1-x)^p$. For example, even if 20% of the patterns may be missed during one run, then less than 1% of patterns would be missed after 3 runs. (Looking ahead, the accuracy of one run is in fact higher than 80% for the real data sets in our experiments.) Therefore, the RAM algorithm can achieve a high degree of accuracy with relatively short execution time.

The remainder of the paper is organized as follows. Section 2 is the related work. Section 3 defines the preliminary concepts. Some important properties of frequent approximate patterns are explained in Section 4. Section 5 presents a basic algorithm for the problem based on depth first search. Then, in section 6 we provide an optimized algorithm Monkey. Section 7 presents experiment results. In Section 8, some improvements are discussed, and Section 9 concludes our study.

2 Related Work

In recent years, a large number of algorithms have been designed to find exact frequent patterns. Inokuchi et al. proposed an Apriori-based algorithm [9], called AGM, to discover all frequent substructures. Kuramochi and Karypis further developed FSG [11], using a more efficient graph representation structure and edge-growth instead of vertex-growth. Yan et al. developed gSpan and designed DFS lexicographic order to support the mining algorithm. Huan et al. proposed FFSM [7] with a graph canonical form, called CAM, and a set of CAM tree operations. While AGM and FSG took advantage of apriori-like level-wise approaches, gSpan and FFSM adopted depth first search, which was more efficient for graph mining problems. Meanwhile, Huan et al. developed SPIN [8] and Nijssen et al. presented Gaston [13], both of which mined frequent patterns by first mining frequent trees or more basic patterns.

One important application of the graph mining algorithms is to find frequent patterns, motifs, and modules in biological networks. Koyuturk et al. [10] introduced an efficient algorithm specially designed for biological data. However, many edges in the dataset were unreliable, which indicated efficient approximate pattern mining algorithms should be designed for biological datasets with edge distortions.

In [19], Yan et al. presented Grafil to perform approximate graph indexing with edge relaxation. However, there are very few approximate graph mining algorithms in the current state of graph-based data mining. Holder et al. proposed SUBDUE [6] to discover approximate substructure pattern based on the minimum description length principle and optional background knowledge. Since SUBDUE used a computationally-constrained beam search, it cannot discover the complete set of frequent patterns. Furthermore, SUBDUE is not designed to find patterns in datasets with edge distortions. In [20], approximate pattern mining in the itemset setting has been proposed.

To the best of our knowledge, hashing has not been used in graph mining, but it is used in frequent itemset mining. In [15], the authors proposed a hashing method to find frequent 2-itemsets. Multiple 2-itemsets were hashed into a single entry. If any of these 2-itemset occurred in a transaction, the count of the hashing entry incremented. This can be used to efficiently discover frequent 2-itemsets. On the other hand, in [4], randomized algorithm has been first used in pattern mining.

3 Preliminaries

In this section, some terminologies are introduced, and then the problem statement is given.

Definition 1. *A* **labeled graph** *G is a five element tuple $G = \{V, E, \Sigma_V, \Sigma_E, L_G\}$ where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. Σ_V and Σ_E are the sets of vertex and edge labels respectively. The labeling function L_G defines the mappings $V \to \Sigma_V$ and $E \to \Sigma_E$. If there is no edge between vertex x and y in G, we assume a virtual edge e_{xy} between x and y and define $L_G(e_{xy}) = \varnothing$. $V(G)$ and $E(G)$ represent the vertex and edge set of G.*

Definition 2. *To model the frequent approximate patterns with edge distortion, we introduce three parameters,* **support** *γ,* **variability** *β, and* **tolerance** *α. We will illustrate the details of these parameters in the following definitions.*

1. Support γ confines the size of support set.
2. Variability β indicates the maximum number of missing edges from an embedding.
3. Tolerance α regulates the maximum number of missing edge from the support set.

Definition 3. *We define a labeled graph p is β edge isomorphic to q, denoted by $p \approx_\beta q$ iff*

1. *$\exists$ bijection $f : V(p) \leftrightarrow V(q)$, $L_p(x) = L_q(f(x))$, $x \in V(p)$, $f(x) \in V(q)$*
2. *At most β different pairs of vertices (x, y) in $V(p)$, such that $L_p(e_{xy}) \neq L_q(e_{f(x)f(y)})$. Moreover, if $L_p(e_{xy}) \neq L_q(e_{f(x)f(y)})$, $L_p(e_{xy}) = \varnothing$.*

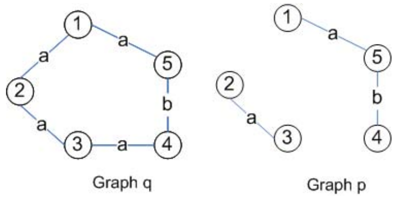


Fig. 1. β edge isomorphism ($\beta \geq 2$)

Figure 1 shows an example of β edge isomorphism. In the example, graph p is at least 2 edge isomorphic to q; however, graph q is not any edge isomorphic to p. The definition of β edge isomorphic is not symmetrical, i.e., $p \approx_\beta q$ cannot always lead to $q \approx_\beta p$, and p is not necessarily connected.

Graph G is β **edge subgraph isomorphic** to graph G', denoted by $G \subseteq_\beta G'$, iff there exists a subgraph G'' of G' such that $G'' \approx_\beta G$. G'' is defined as an **embedding** of G in G'. In figure 1, when q is a given pattern, then p can be considered as an embedding of q.

Definition 4. *Given a graph database $D = \{g_1, g_2, ..., g_n\}$, the β edge different support set of subgraph g, denoted by $sup(D, g, \beta)$, is defined as the subset of D to which g is β edge subgraph isomorphic.*

$$sup(D, g, \beta) = \{g_i | g \subseteq_\beta g_i, g_i \in D\}$$

When $\beta = 0$, $sup(D, g, 0)$ is the subset of D to which g is exactly subgraph isomorphic.

Parameter **support** γ is used to indicate the minimum frequency threshold. However, if graph g contains some edge which rarely appears in the database, it is not beneficial to include g in the results. Therefore, we introduce another parameter **tolerance** α. If any edge of graph g appears less than $\gamma - \alpha$ times in $sup(D, g, \beta)$, we will not define g as frequent.

Definition 5. *Let $|E_g(e)|$ be the number of edges in graph g with the same edge and vertex label, e.g., in figure 2, $|E_g(e)| = 4$.*

Graph g is β **edge** γ **frequent with tolerance** α *(i.e.,* **frequent approximate pattern***) iff*

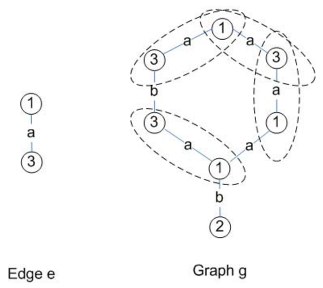


Fig. 2. Definition of $|E_g(e)|$

1. $|sup(D, g, \beta)| \geq \gamma$.
2. *For each edge* e *in* g, $|S| \geq \gamma - \alpha$, *where* S *is a subset of* $sup(D, g, \beta)$ *and each database graph* g_i *in* S *satisfies the following condition:* g_i *contains at least one embedding* s, *s.t.* $s \approx_\beta g$, $|E_s(e)| = |E_g(e)|$.

In figure 2, if graph g is a frequent approximate pattern, it indicates that there are at least $\gamma - \alpha$ embeddings of g in different graphs. Each of these embedding contains all the 4 marked edges. A further example will be presented in section 4.2.

Problem Statement: Given a graph database D and parameters support γ, variability β and tolerance α, we are to find all the connected graph g which is β edge γ frequent with tolerance α of edge missing in the support set.

4 Properties

In this section, we illustrate some important properties of approximate patterns.

4.1 Possible Absence of Exact Match

In the traditional graph mining models, a frequent pattern is exactly embedded in a minimum number of database graphs. However, after adding approximation, it is possible for a pattern to be a frequent approximate pattern even though it is not exactly embedded in the database. For example, assume that the parameters are $\alpha = 2$, $\beta = 1$, and $\gamma = 3$, the graph pattern g (as in figure 3) does not have any exact embedding in figure 4. However, g is β edge subgraph isomorphic to all the graphs in the database. Also, every type of edges occurs more than 1 $(= 3 - 2)$ times. Therefore, g is a frequent approximate pattern in the database.

In our model, having exact embeddings is not a necessity to be defined as frequent. In many chemical and biological applications, important patterns may have several variations. As a result, traditional exact graph mining methods may miss these patterns when all the corresponding mutations are below the threshold. However, approximate graph mining may find the patterns, given the range of edge distortion.

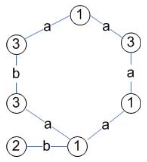


Fig. 3. Frequent Approximate Pattern without Exact Match

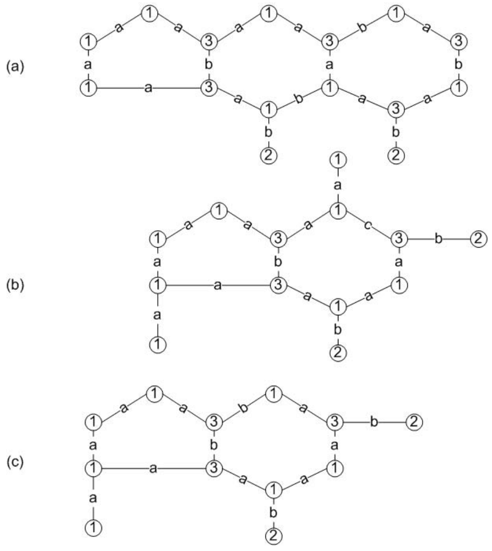


Fig. 4. Graph Database

4.2 Maximal Representation

There are two kinds of constraints. The first one is on the whole approximate frequent pattern, the other is on each type of embedding edges.

The second constraint guaranteesthe maximal representation property of approximate patterns. Any subgraph of pattern g, except g itself, cannot represent all the embeddings of g, i.e., for any non-trivial subgraph t of g, there exist at least $\gamma - \alpha$ embeddings of g which are not β edge isomorphic to t.

For example, assume a pattern p occurs in γ graphs, without the second property, i.e., the tolerance parameter, we can add any β edges into p, the resulting is still a frequent approximate pattern. Thus, we employ the tolerance parameter to prevent this.

Figure 5 further explains the property. Suppose the graph database and the parameters are the same as those in figure 4. For any non-trivial subgraph t of g, there exists at least one edge e (as marked) embedded in g but not in t. We also marked all the edges of the same type as e by ellipses. Suppose g is a frequent approximate graph pattern; then there are at least $\gamma - \alpha$ embeddings of g, which contain all the edges we have marked. Therefore, any of these embeddings contains e. According to definition 2, these

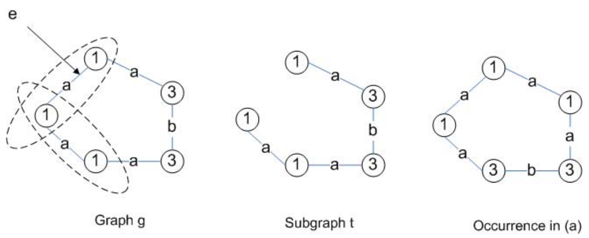


Fig. 5. Maximal Representation

embeddings are not β edge isomorphic to t. In this example, one embedding in graph (a) is not 1-edge isomorphic to t, as shown in figure 5.

However, if we do not include the second constraint in the definition, the maximal representation property can no longer hold. Many redundant patterns may be generated, whose embeddings can be represented by one of its subgraphs.

4.3 Apriori Property

The Apriori property claims that if a pattern satisfies the minimum support threshold, then any sub-pattern of it also satisfies the minimum support threshold. The property is very useful in the field of data mining.

By our definition of frequent approximate patterns, we have the following theorem.

Theorem 1. For a frequent approximate graph g, any subgraph of g is also a frequent approximate pattern.

Proof: For any subgraph s of g, there is a sequence of subgraphs of g: $a_0 = s$, $a_1 = s + e_1$, $a_2 = s + e_1 + e_2$, ..., $a_n = g$. a_i be a subgraph of a_{i+1} with one edge missing. By definition, when $k = n$, $a_n = g$ is a frequent approximate pattern. Suppose when $k = i$, a_i is a frequent approximate pattern; we therefore assume $a_i - a_{i-1} = e$. We want to prove that the approximate pattern a_{i-1} is frequent.

For every embedding t of a_i, if e is embedded in t as it is in a_i, we define $t' = t - e$. Otherwise, $t' = t$. From our definition, t' is also β edge isomorphic to a_{i-1}. Thus, $sup(D, a_{i-1}, \beta) \geq sup(D, a_i, \beta) \geq \gamma$. Also, for any edge e' in a_{i-1}, if $|E_t(e')| = |E_{a_i}(e')|$, we have $|E_{t'}(e')| = |E_{a_{i-1}}(e')|$. Thus, every type of edge exists no less than $\gamma - \alpha$ times in the graph database. Therefore, a_{i-1} is a frequent approximate pattern. By induction, we have $a_0 = s$ is also a frequent approximate pattern.

Theorem 1 guarantees that if graph h is not a frequent approximate pattern, we need not examine any supergraphs of h. This theorem is the foundation of our mining algorithm.

5 Basic Algorithm

In this subsection, we present a basic approximate graph mining algorithm based on depth first search, which is very similar to former exact graph mining algorithms.

Given parameters α, β, and γ, we first find frequent approximate edges by enumeration, and remove non-frequent edges in the database. Starting from these approximate frequent edges as a first set of patterns, we recursively grow edges in a depth first fashion. If all possible candidate patterns grown from pattern g have been checked, we backtrack to the pattern from which g is grown, and grow another edge to that pattern.

We first discuss the case when we grow an edge with a new vertex to an existing pattern g. Let e_{uv} be the growing edge, in which u exists in g but v is the new vertex to be introduced. Here we assume the label of vertex u is l_u and the label of v is l_v. If the pattern of $g + e_{uv}$ has not been reached before by the mining algorithm. We are to decide whether $g + e_{uv}$ could be a frequent approximate pattern. To achieve this, all the embeddings of g are examined. We test,

1. whether there are no less than γ database graphs which contain at least one embedding of g and a vertex v' with label l_v. These database graphs constitute the support set. Moreover, if each embedding in a database graph has exactly β edge difference from pattern g, and none of them are connected to v' with the same label as the growing edge e_{uv}, this database graph cannot be added to the support database graph set.
2. whether there are no less than $\gamma - \alpha$ database graphs which contain at least one embedding of g and vertex v'. Besides, if none of the embeddings are connected to v' with the same label as the growing edge e_{uv} and exactly contain all the edges in pattern g of the same type of e_{uv}, the database graph cannot be counted effectively.

If these two requirements are satisfied, the new pattern of $g + e_{uv}$ is also a frequent approximate pattern.

The procedure of adding a new edge to an existing pattern g without introducing any new vertex is similar. The search is also done in a depth first fashion. We do not need to identify the new vertex any more since the two endpoints of the new edge are already known. After discovering all frequent approximate patterns in which edge s is embedded, we remove s from the graph database to shrink the searching space.

In this basic algorithm, we adopt canonical adjacent matrix to determine whether the ongoing candidate pattern has been reached or not. The basic algorithm is a natural extension of exact graph mining algorithms to approximate graph mining. However, the calculations of canonical forms of graphs, especially for larger graphs, is extremely time-consuming. Additionally, frequent approximate graph models tend to generate many more patterns. We need to deal with these two difficulties in approximate graph mining. In the next section, we introduce a randomized algorithm, which is more efficient and flexible.

6 Algorithm RAM

A maximal frequent graph is a frequent graph all of whose supergraphs are infrequent. In approximate graph mining, taking edge relaxation into account, the average size of maximal frequent approximate pattern grows. Consequently, the number of non-maximal frequent approximate graphs is even larger. (This is due to the exponential growth of subgraphs, e.g., there are approximately $2^{\binom{n}{2}}/n$ subgraphs of a complete graph with n

nodes.) In the basic algorithm, we calculate the canonical form for every pattern. Here we present a Randomized Approximate Graph Mining algorithm (RAM) for better efficiency. RAM works in a depth first fashion just like the basic algorithm. Instead of using canonical forms, we retrieve a set of features to identify possible patterns. The set of features are carefully selected, and can be obtained from pattern graphs in polynomial time. To achieve better space management, we hash those feature sets into hash vectors. In the mining process, if a pattern with the same hash vector has been examined before, we will stop the searching from the pattern. The algorithm can also be applied to exactly mining large graphs. Next we will introduce the details of the algorithm.

6.1 Feature Retrieval

In the last section, we use canonical forms to distinguish graph patterns. However, the calculation of canonical forms of arbitrary graph patterns can be very expensive. Therefore, in this section, we use feature sets to identify patterns. The intuition is that more often than not a feature set is strong enough to distinguish graph patterns as long as it is well selected. Advantages of the substitution include: (1) calculating feature set is asymptotically easier than canonical forms; and (2) the space consumption of feature sets after hashing is very flexible.

The design of feature set is of crucial importance to the randomized algorithm. Although the design may vary for the graphs in different databases, there are some goals in common, which include: (1) to assure any two isomorphic graphs have the same feature set, which suggests that any kind of automorphism form of the same pattern should result in the same feature values; (2) to minimize the number of patterns which may share the same feature values with another pattern. When collisions are inevitable, patterns sharing the same feature values should at least have great topological similarity; (3) to minimize calculation - any feature selected can be calculated in polynomial time. In this section, we provide one possible design of a set of feature set, which takes advantage of a variety of topological information.

We map each vertex and edge label to a positive prime number. Suppose the total weight of the minimum spanning tree of graph g is $W(MST(g))$; the distance between two vertices v_1, v_2 is $dis(v_1, v_2)$; the degree of vertex v is $deg(v)$; the set of edges incident with vertex v is $e(g, v)$. Then, for a graph g, we can select the following set of six features $f_1, ..., f_6$.

1. f_1 relates to how big g is.
   $f_1(g) = |V||E|$.
2. f_2 relates to the type of edges.
   $f_2(g) = \Sigma_{e_{xy} \in E(g), Lg(e_{xy}) \neq \varnothing} L_g(x)L_g(y)L_g(e_{xy})$.
3. f_3 relates to the minimum spanning tree of g.
   $f_3(g) = W(MST(g))$.
4. f_4 relates to the connectivity of g.
   $f_4(g) = \Sigma_{v_1, v_2 \in V(g), v_1 \neq v_2} dis(v_1, v_2)$.
5. f_5 relates to the degree sequence of g and vertex labels.
   $f_5(g) = \Sigma_{v \in V(g)} deg^2(v)L_g(v)$.

6. f_6 relates to the degree sequence of g and edge labels.
   $f_6(g) = \Sigma_{v \in V(g)} deg^2(v)(\Sigma_{e \in e(g,v)} L_g(e))$.

All the features can be obtained in polynomial time. In fact, since we adopt a depth first search as framework, all the feature values can be calculated in linear time by using the information of the predecessor. It is obvious that for a pair of graphs g_1, g_2, if $g_1 \approx g_2$, then $f_i(g_1) = f_i(g_2)$. For the benefit of space efficiency, we hash these features into a hash vector, i.e., $H_i = f_i \bmod P_i$, while P_i is a prime number. The selection of the prime numbers is based on the size of available space, i.e., P_i is proportional to the number of possible value of f_i and $\prod_i P_i$ is about the size of available space for the mining program.

6.2 A Randomized Algorithm for Approximate Graph Mining

In this subsection, we introduce the randomized algorithm RAM for approximate graph mining. The main differences between RAM and the basic algorithm are (1) RAM grows edges in a random order, and (2) RAM adopts hash vectors of feature sets instead of using canonical forms.

We use basic algorithm to mine the frequent approximate patterns with less than L edges first. Those small patterns tend to be missed by the randomized algorithm but can be mined efficiently. Expanding from those relatively small patterns, we reach candidate graph patterns by growing edges in a random order. Moreover, for any candidate graph g, instead of calculating the canonical form, we use the hash vector of the selected feature set of g. If a pattern with the same hash vector has already been found, graph g is considered to be reached (even if it is not). Otherwise, if g is frequent, we continue to grow edges from g.

After discovering all frequent approximate patterns in which edge s is embedded, we remove s from the graph database to shrink the search space. Also, we reset all sets of hash values. Because none of the patterns discovered afterwards contain s, it is safe to reset used hash vectors to avoid collisions. Algorithm RAM is shown in algorithm 1.

As we did not adopt canonical forms for efficiency, the algorithm may lose some patterns in a single run, i.e., if pattern A and B have exactly the same hash vector value, we may lose B if A has been reached first. However, since we grow edges in a random order, low miss rate can be achieved by multiple runs; i.e., in another run, B is recognized as frequent if it appears earlier than A. Furthermore, missing most of the non-maximal patterns is harmless to if their supergraph patterns can be discovered by the algorithm, i.e., if B and A are subgraphs of pattern C, we may find C by growing edges from A even if we failed to find B. Thus we recover B as a subgraph of C. Further analysis is presented in the following subsection.

6.3 Algorithm Analysis

As shown in the previous subsection, the randomized algorithm can miss patterns. We miss a pattern g when (1) a different graph with the same hash vector has been discovered already, and (2) all the connected subgraphs of g with one edge absent are missing.

When the feature set is well selected, we assume that their corresponding hash vectors are uniformly distributed. Then, the first factor is determined by the number of possible values of hash vectors, i.e. $\prod_{i=1}^{6} P_i$ in this paper for the features and hash functions we selected. Assume there are at most M approximate patterns with a given graph database and parameters which contain a same edge, the average number of graphs which share a same hash vector is $M/\prod_{i=1}^{6} P_i$. We define $P = min\{1, \prod_{i=1}^{6} P_i/M\}$, P quantifies the average probability of pattern being found without considering the second factor.

A connected subgraph with one edge of g absent is either a connected spanning subgraph, when e is not a bridge edge in g; or a non-trivial component of graph $g - e$ with $|V(g) - 1|$ vertices, when e is a bridge edge in g. The second factor is largely affected by the number of connected subgraphs with one edge absent, which is reversely related to the number of bridge edges of the pattern graph. The number of connected subgraphs with one edge absent varies a lot with the pattern graph, e.g., it is 2 for a line graph of size n while it is m for a cycle graph of size m. To simplify the problem, we assume that the number of connected subgraphs of a graph of size n with one edge absent is $n/2$.

Algorithm 1. *RAM*

```
   Input: Graph database $D$, parameter $\beta$, $\gamma$, $\alpha$
   Output: FAG (frequent approximate graph) set $G(D)$
 1: Find one edge FAG set $G^1(D)$
 2: Find small FAG set (below $L$ edges) $G^s(D)$
 3: $G(D) \leftarrow G^s(D)$
 4: for each graph $s \in G^s(D)$ do
 5:     $G(D) \leftarrow G(D) + RandomGrow(s, G^1(D), D)$
 6:     $G^s(D) \leftarrow G^s(D) - s$
 7:     if $\exists$ edge $e \in s, \forall s' \in G^s(D), e \in s'$ then
 8:         $G^1(D) \leftarrow G^1(D) - e$
 9:         Delete saved hash vectors
10:     end if
11: end for
   Subprocedure name: RandomGrow
   Input: FAG $a$, edge set $S$, Graph database $D$
   Output: frequent supergraphs of $a$
 1: $S' \leftarrow S$
 2: while $|S'| > 0$ do
 3:     Randomly select edge $s$ from $S'$
 4:     for each candidate pattern $p \in a + s$ do
 5:         if $p$ is frequent and $H(p)$ has not been saved then
 6:             output $p$ and save $H(p)$
 7:             $RandomGrow(p, S, D)$
 8:         end if
 9:     end for
10:     $S' \leftarrow S' - s$
11: end while
```

Assume the average probability of a pattern graph of size n being found is P_n; combining above two factors, $P_n = P - P(1 - P_{n-1})^{(n/2)}$. Since we start the randomized algorithm only from larger patterns, the second part of the formula is negligible. Thus, we have $P_n \approx P$.

Supposing we would find approximately P of the maximal patterns in a single run, we would find approximately $1 - (1 - P)^q$ portion of the maximal patterns if we run RAM q times.

The hash vector of any pattern occupies only one bit in the multidimensional hash table. Therefore, P can be quite large. If the set of features is well designed, we can find most of the patterns at a much faster speed, as we avoid the tedious calculation of canonical forms. Moreover, if the amount of main memory is limited, we can adopt a hash function with small hash table, at the cost of losing more patterns in a single run. However, we can improve the accuracy by employing more runs.

7 Experiments Results

In this section, we report on our experiments, which validate the efficiency and effectiveness of the approximate graph mining algorithms. Due to Property 1, some patterns may not have any exact embedding. As a result, existing exact subgraph mining methods cannot be directly applied because they aim to find exact patterns. The performance of algorithm RAM is compared with that of the basic algorithm which can be considered a modification of existing depth first graph mining methods, e.g., FFSM [7], gSpan [18], etc. We do not compare RAM with any breadth first search algorithm because breadth first search is too space-consuming for approximate graph mining problems.

We use two kinds of datasets in our experiments: one real dataset and several synthetic datasets. The real dataset consists of 394 graphs. Each graph corresponds to a metabolic network of glycan. The network is generated from KEGG [22] using the second-level categories defined in [23]. In each graph, vertices represent enzymes, labeled with protein family ID. If there is an edge between enzymes (vertices) A and B, it indicates that A and B interact with each other. The synthetic data was generated using a method similar to that in [11]. The generator allows the user to specify the number of graphs, their average size, the number of seed graphs, the average size of seed graphs, the number of distinct labels, and the approximate level (the random probability that we change the edge label) of each edge.

In algorithm RAM, we adopt the set of hash functions for the real dataset. We set $P_1 = 53$, $P_2 = 53$, $P_3 = 43$, $P_4 = 43$, $P_5 = 13$, and $P_6 = 13$. All our experiments are performed on a 2.8GHZ, 2GB memory, Intel PC. Both algorithm RAM and the basic algorithm are compiled with VS2005.

7.1 Metabolic Pathways Dataset

We employ a categorized metabolic pathway dataset for the experiments. In the pathways graph databases, there are 394 graphs in total. The randomized algorithm starts when the size of the pattern graph is bigger than the size threshold 5. In each experiment, we only run RAM once.

Table 1. Results of Pathways Dataset (1)

(γ, β, α)	Exe.Time of Algorithm RAM	Basic Algorithm
(25,1,2)	9 s	25 s
(25,1,4)	12 s	33 s
(35,1,2)	8 s	17 s
(35,1,4)	10 s	26 s

(γ, β, α)	No. of Max. Patterns of RAM	Basic Algorithm
(25,1,2)	123	123
(25,1,4)	131	131
(35,1,2)	107	107
(35,1,4)	112	112

Table 2. Results of Pathways Dataset (2)

(γ, β, α)	Exe.Time of Algorithm RAM	Basic Algorithm
(25,1,2)	9 s	25 s
(25,2,2)	54 s	126 s
(35,1,2)	8 s	17 s
(35,2,2)	45 s	104 s

(γ, β, α)	No. of Max. Patterns of RAM	Basic Algorithm
(25,1,2)	123	123
(25,2,2)	461	467
(35,1,2)	107	107
(35,2,2)	431	431

First we show the runtime of both algorithms and the number of maximal approximate patterns. We set parameters (γ, β, α) to range from $(25, 1, 2)$ to $(35, 2, 4)$. As there are three different parameters, we show the results of both algorithms in the following tables. In table 1, in each subgroup, we keep β and γ unchanged and vary α from 2 to 4. In table 2, we vary β from 1 to 2. Lastly, in table 3, we vary γ from 25 to 35.

We can observe from the tables that overall, algorithm RAM is about 60% more efficient than the basic algorithm. On average, RAM can find no less than 99% of the total maximal patterns at a single run. When parameter γ decreases, the optimization of algorithm RAM tends to be more effective, due to the increase of frequent approximate patterns; the basic algorithm must calculate the canonical forms of every graph. The number of maximal frequent patterns grows quickly with parameter β, and it is consistent with the runtime of the algorithms.

When $\alpha = 2$, $\beta = 1$, and $\gamma = 25$, RAM found 123 maximal patterns, and the largest approximate pattern contained 14 edges and 14 vertices. The exact graph mining method only found 24 maximal patterns, and the maximal edge and vertex count of the exact patterns were 10 and 9, respectively. This indicates that approximate graph mining methods can find more and larger frequent patterns.

Last, we tested if the approximate patterns are useful in real applications. We divided the graph database into two parts. There were 300 database graphs in the first part and 94 in the second. We generated the exact and approximate patterns from the first set of patterns. Graphs in the second part were mixed with 100 arbitrary graphs as a test set.

Table 3. Results of Pathways Dataset (3)

(γ, β, α)	Exe.Time of Algorithm RAM	Basic Algorithm
(25,2,2)	54 s	126 s
(35,2,2)	45 s	104 s
(25,2,4)	63 s	149 s
(35,2,4)	50 s	125 s

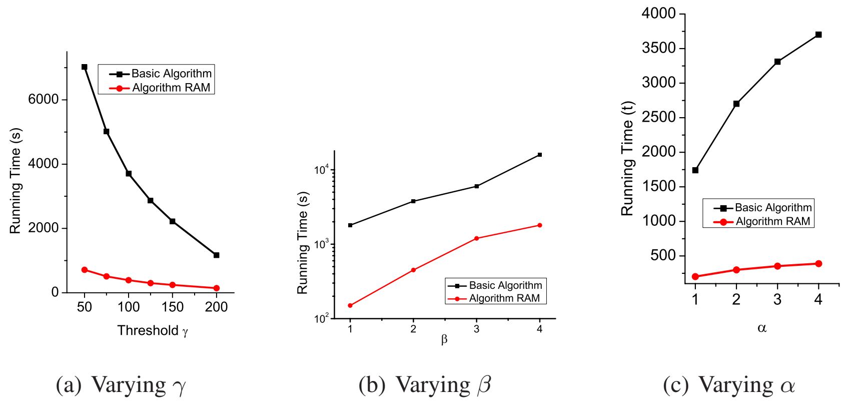
(γ, β, α)	No. of Max. Patterns of RAM	Basic Algorithm
(25,2,2)	461	467
(35,2,2)	431	431
(25,2,4)	497	505
(35,2,4)	448	450

The exact and approximate patterns from the first part were used to predict whether a graph in the mixed set belongs to the original family. We found that approximate patterns yield about 15% higher accuracy than exact patterns on average, which indicates that approximate patterns better capture characteristics of the original data set.

7.2 Synthetic Dataset

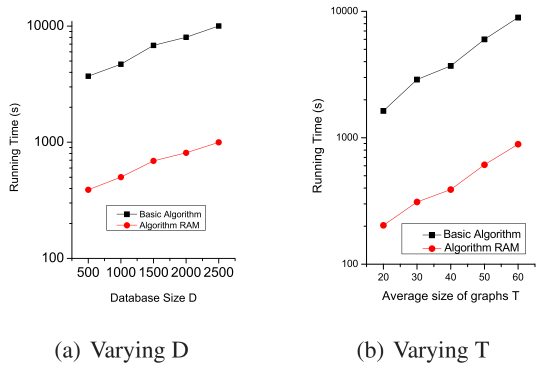
In this section, we analyze the performance of the RAM algorithm on synthetic datasets. The synthetic graph dataset is generated as follows. First, a set of seed fragments is generated randomly, whose size is determined by a Poisson distribution with mean I. The size of each graph is a Poisson random variable with mean T. Seed fragments are then randomly selected and inserted into a graph one by one until the graph reaches its assigned size. We add an approximate level parameter P, which is the random probability that we change the edge label. More details about the synthetic data generator are available in [11]. A typical dataset may have the following setting: it has 500 graphs and uses 300 seed fragments with 30 distinct labels. On average, each graph has 40 edges and each seed fragment has 20 edges, the probability of change for any edge label is 0.05. This dataset is denoted as D500I20T40S300L30A0.05.

We first compared the performance of both algorithms with respect to the user input parameters β, γ, and α on the synthetic dataset D500I20T40S300L30A0.05. In test 1, we set α to 4 and β to 2, and vary γ from 50 to 200. In test 2, we set γ to 100 and β to 2, and vary α from 1 to 4. In test 3, we set γ to 100 and α to 2, and vary β from 1 to 4. Figure 6 shows the running time of both algorithms in tests 1, 2 and 3, respectively. Compared with the real dataset, RAM has a similar performance gain as in the synthetic datasets. However, it can be observed that the runtime increases significantly as β grows. We then tested the performance of the approximate graph mining algorithms with respect to different types of input datasets. Two characteristics of the datasets were selected for analysis: (1) the average size of graphs in the dataset (T), and (2) the number of graphs in the database (D). During the experiments, we set $\alpha = 4$, $\beta = 2$, and $\gamma = D/5$. Other non-varying parameters were the same as in the last experiments. In these tests, RAM can discover almost all maximal frequent approximate graphs that the basic algorithm can find, since we set P_i to be fairly large. To test the



(a) Varying γ (b) Varying β (c) Varying α

Fig. 6. Performance of Varying α, β, γ

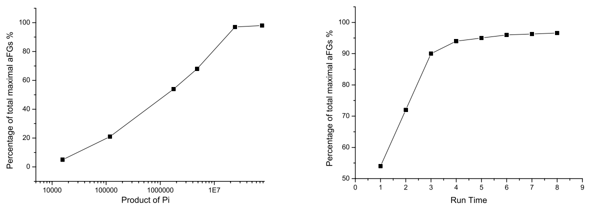


(a) Varying D (b) Varying T

Fig. 7. Performance of Varying D, T

performance of RAM under a small space requirement, we set P_i to smaller values. We set $\alpha = 4$, $\beta = 2$, and $\gamma = 100$. Then, we varied the product of P_i, i.e., $\prod_{i=1}^{6} P_i$ from 10^4 to 10^8. Figure 8 shows the percentage of total maximal patterns algorithm RAM can find with different products of P_i. As we can see from the figure, the percentage of the total maximal patterns can be discovered increases greatly when products of P_i increases. Next, we tested the effects when we ran RAM multiple times with a small product of P_i. We set the product of P_i to 10^6, leaving the other parameters the same as those in the last experiment. Figure 8 indicates that the percentage of discovered maximal patterns rises significantly after we run RAM two or three times. However, when we ran it more times, increased amount of discovered patterns diminished. We will discuss the reason for this in a later section.

The empirical study clearly shows that RAM is much more time efficient than the basic algorithm under different sets of parameters , though it tends to lose a very small



(a) Varying Product (b) Runtime

Fig. 8. Percentage of total maximal FAGs varying product of P_i, and run time

fraction of maximal pattern with a small product of P_i and run times. However, when we execute RAM multiple times with a larger product of P_i, most frequent subgraphs can be discovered.

8 Discussion

In the experimental results section, we found that when there are more frequent approximate patterns than the available memory allows for, some patterns are more difficult to reach than others. To discover these patterns, we may have to run algorithm RAM many times. In this section, we discuss the reason and a possible solution.

If the hash functions are well designed, we can make the distribution of hash vectors close to uniform. However, the second factor, i.e., the number of connected subgraphs, is determined by the nature of the graph database. Even if we restrict the randomized search to larger patterns, some patterns may still be more vulnerable to not being discovered. To compensate the loss of patterns which have fewer subgraphs, one possible solutions is to add a probability parameter ϕ for those patterns, indicating the likelihood for continuing to search from those patterns when there is a hit on their hash value.

9 Conclusion

Graphs play an important role in modeling complex structural data, e.g., protein interaction networks, social networks, etc. In many applications, the graphs in the database may contain a number of unreliable edges. However, none of the current graph mining algorithms are designed to find the complete set of frequent patterns in graph databases with edge distortions.

We first introduced a formal definition of frequent approximate patterns in a graph database with edge distortion. Then we proposed a basic algorithm based on depth first search. As there are much more frequent patterns and embeddings, we presented the hashing-based randomized algorithm RAM. As shown in the experimental results, the approximate graph mining algorithm can find more and larger frequent patterns. More significantly, by approximate pattern mining, we may find important patterns that cannot be discovered by exact mining algorithms.

References

1. Bader, J., Chaudhuri, A., Rothberg, J., Chant, J.: Gaining confidence in high-throughput protein interaction networks. Nature Biotechnology 22(1), 78–85 (2004)
2. Chang, R., Podgurski, A., Yang, J.: Finding What's not there: a new approach to revealing neglected conditions in software. In: Proc. of ISSTA (2007)
3. Cong, G., Yi, L., Liu, B., Wang, K.: Discovering frequent substructures from hierarchical semi-structured data. In: Proc of SDM (2002)
4. Gunopulos, D., Mannila, H., Saluja, S.: Discovering All Most Specific Sentences by Randomized Algorithms Source. LNCS 1997(1997)
5. Hasan, M., Chaoji, V., Salem, S., Besson, J., Zaki, M.: ORIGAMI: Mining Representative Orthogonal Graph Patterns. In: Perner, P. (ed.) ICDM 2007. LNCS (LNAI), vol. 4597. Springer, Heidelberg (2007)
6. Holder, L., Cook, D., Djoko, S.: Substructure discovery in the subdue system. In: Proc. AAAI (1994)
7. Huan, J., Wang, W., Prins, J.: Efficient mining of frequent subgraphs in the presence of isomorphism. In: Proc. of ICDM (2003)
8. Huan, J., Wang, W., Prins, J., Yang, J.: SPIN: mining maximal frequent subgraphs from graph databases. In: Proc. of KDD (2004)
9. Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. In: Proceedings of PDKK (2000)
10. Koyuturk, M., Grama, A., Szpankowski, W.: An efficient algorithm for detecting frequent subgraphs in bioloical networks. Bionformatics 20, 200–207 (2004)
11. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Proc. of ICDE (2001)
12. Kuramochi, M., Karypis, G.: Finding frequent patterns in a large sparse graph. Data Min. Knowl. Discov. (2005)
13. Nijssen, S., Kok, J.: A quickstart in frequent structure mining can make a difference. In: Proc of KDD (2004)
14. Pei, J., Jiang, D., Zhang, A.: On Mining Cross-Graph Quasi-Cliques. In: Proc. of KDD (2005)
15. Park, J., Chen, M., Yu, P.: An effective hash based algorithm for mining association rules. In: Proc. SIGMOD, pp. 175–186 (1995)
16. Thomas, L., Valluri, S., Karlapalem, K.: MARGIN:Maximal Frequent Subgraph Mining. In: Proc. of ICDM (2006)
17. Yan, X., Han, J.: CloseGraph: Mining closed frequent graph patterns. In: Proc. of SIGKDD (2003)
18. Yan, X., Han, J.: gSpan: graph-based substructure pattern mining. In: Proc. of ICDM (2002)
19. Yan, X., Yu, P., Han, J.: Substructure similarity search in graph databases. In: Proc. of SIGMOD (2005)
20. Liu, J., Paulsen, S., Xu, X., Wang, W., Nobel, A., Prins, J.: Mining approximate frequent itemset from noisy data. In: ICDM (2005)
21. Zaki, M.: Efficiently mining frequent trees in a forest: algorithms and applications. In: IEEE TKDE (2005)
22. Kyoto Encyclopedia of Genes and Genomes, `http://www.genome.jp/kegg/`
23. Metabolic pathway categories in KEGG, `http://www.kegg.com/kegg/pathway/map/map01100.html`