Knowl Inf Syst (2011) 28:423–447
DOI 10.1007/s10115-010-0376-y

REGULAR PAPER

An efficient graph-mining method for complicated and noisy data with real-world applications

Yi Jia · Jintao Zhang · Jun Huan

Received: 29 January 2010 / Revised: 10 November 2010 / Accepted: 20 November 2010 /
Published online: 2 February 2011

Abstract In this paper, we present a novel graph database-mining method called APGM (APproximate Graph Mining) to mine useful patterns from noisy graph database. In our method, we designed a general framework for modeling noisy distribution using a probability matrix and devised an efficient algorithm to identify approximate matched frequent subgraphs. We have used APGM to both synthetic data set and real-world data sets on protein structure pattern identification and structure classification. Our experimental study demonstrates the efficiency and efficacy of the proposed method.

Keywords Graph mining · Approximate subgraph isomorphism

1 Introduction

Frequent subgraph mining is an active research topic in the data-mining community. The graph-mining techniques have been extensively applied in a wide range of applications domains, such as bioinformatics [9,13], chemoinformatics [8,24], social network analysis [20,27], and many others.

Many current frequent subgraph-mining algorithms share a common strategy in determining the support value of a subgraph pattern and hence deciding whether the subgraph is frequent. In this strategy, in matching a subgraph pattern to a graph, we require that node

Y. Jia · J. Huan (✉)
Department of Electrical Engineering & Computer Science,
University of Kansas, Lawrence, KS 66045, USA
e-mail: jhuan@ittc.ku.edu

Y. Jia
e-mail: jiayi@ittc.ku.edu

J. Zhang
Center for Bioinformatics, Department of Molecular Biosciences,
The University of Kansas, Lawrence, KS 66046, USA
e-mail: jtzhang@ku.edu

labels, edge relationships, and edge labels should be the same between the subgraph pattern and the matching graph [10]. We call this strategy it exact matching.[1]

Although exact matching is widely used, in applying frequent subgraph mining to real-world applications, we observe that exact matching may not always produce the optimal results in all applications. The situation becomes worse in those graph databases that have a large volume of noises (in terms of node or edge label changes) and distortions (in terms of edge relationship changes). For example, in the application of protein structure comparison and structure motif identification, which we are specifically interested in within this paper, graphs corresponding to protein structures often contain a large volume of noises and distortions. In this application, noise and distortion come from a multidimensional source: amino acid changes in proteins (which are called it mutations in biology), slightly different geometric shape of similar proteins, and imperfect experimental measurements, just to name a few examples. As a consequence, using exact matching posts an unrealistical constraint in algorithm design and may miss a lot of important patterns in practice.

The goal of our research is to devise frequent subgraph-mining algorithms that are capable of identifying salient patterns in large graph database that are otherwise overlooked by using exact matching due to the presence of noises and distortions in the graph databases. We call this new strategy it approximate graph mining.

Although approximate graph-mining offers provisions for noise and distortion handling, designing an approximate graph-mining algorithm certainly presents a non-trivial effort. There are two major challenges for the new algorithm design:

- We want to incorporate noise and distortion handling capability in a general framework such that we could adapt the algorithm to different practical problems. This follows the philosophy of "build once, use many times". The challenge is how to provide a general model to cover many application domains.
- It is known that frequent subgraphs mining may suffer from intensive computation due to two reasons: (1) subgraph matching is known as an NP-complete problem, and hence it is unlikely that we will have a polynomial running time solution in general context with the exception of planar graphs and (2) the total number of frequent patterns may grow exponentially in the number of graphs in a database and in the average size of the graphs in the database. Approximate subgraph mining certainly will face the same two problems and the challenge is how to design a practical algorithm that scales to large graphs and large graph databases.

Here, we present a new approximate subgraph-mining method called APGM (APproximate Graph Mining) to address these two challenges. To handle the first challenge, we have developed a general framework that uses a probability matrix to score label mismatches in matching a subgraph pattern to a graph. The advantage of the strategy is that it holds a solid probabilistic ground for a whole class of applications. Utilizing this scoring scheme, we have renewed important definitions, such as isomorphism, subgraph isomorphism, and redesigned the conventional support measures in this new context.

To target at the second challenge, we have designed a depth-first search strategy with a set of pruning strategies. We have applied our algorithm to both synthetic and real data sets. The experimental results demonstrate that our algorithm identifies important subgraphs that cannot be identified by exact matching algorithms with a pattern discovery speed (number of

[1] Technically, we should use *subgraph isomorphism* to define exact matching. The definition of subgraph isomorphism is deliberately delayed to a later section. An intuitive description is provided here to avoid excessive details in the introduction.

patterns divided by the running time) close to, and sometime better than, conventional exact matching algorithms.

In addition, our probability-based method provides a very important property called evolvement, that is, the observed subgraphs in a set of noisy and distorted graphs could evolve from the underlying true patterns. In the biological domain, it is called the evolutionary process.

The rest of the paper is organized in the following way. In Sect. 2, we present an overview of related work on subgraph mining. In Sect. 3, we present our model for approximate subgraph mining. In Sect. 4, we present the details of our algorithm, including its extension and its optimization. In Sect. 5, we present our empirical study of the proposed algorithm using both synthetic and real data sets. Finally, in Sect. 6, we conclude our paper with a short discussion of our work. In the appendix, we show two important extensions of APGM.

2 Related work

Graph database mining is an active research field in data-mining research. The goal of graph database mining is to locate useful and interpretable patterns in a large volume of graph data. Current exact matching graph-mining algorithms can be roughly divided into three categories. The first category uses a level-wise search strategy including AGM (Apriori-based Graph Mining) [16] and FSG (Frequent Subgraphs) [18]. The second category takes a depth-first search strategy including gSpan (Graph-based Substructure PAtterN mining) [28] and FFSM (Fast Frequent Subgraph Mining) [11]. The third category works by mining frequent trees, in which SPIN (SPanning tree-based maximal graph mINing) [15] and GASTON (GrAph/Sequence/Tree extractiON) [21] are the representative. Recently, researchers extend the graph-mining problem from static networks into temporal dynamic networks [19] or involving networks [4]. We refer to [7] for a recent survey.

Frequent subgraph mining with approximate matching capability has also been investigated. Chen et al. proposed a method called gapprox [5], which discovers approximate matched patterns in a single large network. Yan et al. designed a graph query algorithm Grafil (Graph Similarity Filtering) for approximate structure data search [29]. The algorithm SUBDUE [8] considers the situation of inexact matching and includes a distortion cost function as a solution. Zhang et al. provided a method called Monkey [31] to identify maximal approximately frequent trees. Further, the same group introduced a randomized algorithm called RAM to find approximate subsequent subgraphs by using feature retrieval to avoid canonical form calculation [30]. Zou et al. proposed an approximation algorithm MUSE (Mining Uncertain Subgraph pattErns) focusing on uncertain graph database [32]. This method calculated the expected support values of patterns by considering both the occurrences in the uncertain graph databases and the probabilities of the uncertain graph databases.

The differences between existing algorithms and our proposed one are below. Yan's work focuses on proximity measures between graphs and Chen's work concentrates on pattern discovery in a single large graph, which are out of the scope of our current paper. SUBDUE did not provide a complete general frame to address the approximate match issue. It is only applied to small databases and generates an incomplete set of characteristic subgraphs. By using a feature set instead of the canonical form to distinguish patterns, RAM may not provide a complete pattern set. Hence, the algorithm's efficacy highly depends on the quality of user-defined feature set. Different from our method and other methods, instead of the deterministic data, MUSE addressed the uncertain data with inherent statistical properties in nature [1,2,26]. It only handles the uncertain edges and quantifies the uncertainty with the probability distributions.

Different from these existing works, we use a parametric model to determine the probability that a pattern belongs to a graph. We developed a general framework to fully utilize a probability matrix for approximate matching, which we can apply to a number of different applications. And our theoretic framework promises the completeness of the pattern set. Finally, we offered a practical implementation, applied it on both synthetic and real data sets, and evaluated our method rigorously.

3 Theoretic framework

We demonstrate our method called **APGM** (**AP**proximate **G**raph **M**ining) with two steps. In this section, we introduce the theoretic model. In the next section, we show our algorithm in details.

Definition 1 A **labeled graph** G is a 5-tuple $G = \{V, E, \Sigma_V, \Sigma_E, \lambda\}$ where V is the set of vertices of G and $E \subseteq V \times V$ is the set of undirected edges of $G.\Sigma_V$ and Σ_E are (disjoint) sets of labels and labeling function $\lambda : V \rightarrow \Sigma_V \cup E \rightarrow \Sigma_E$ maps vertices and edges in G to their labels. A **graph database** D is a set of graphs.

We also use $V[G]$ to denote the node set of a graph G and $E[G]$ to denote the edge set of G. We also use $\Sigma_{V[G]}$ to denote the node labels, $\Sigma_{E[G]}$ to denote edge labels, and λ_G to denote the labeling function for a graph G. Before, we introduce approximate matching, we define the exact subgraph isomorphic and the compatibility matrix.

Definition 2 A graph G is **subgraph isomorphic** to another graph G', denoted by $G \subseteq G'$ if there exists an injection $f : V \rightarrow V'$, such that

- $\forall\, u \in V, \lambda(u) = \lambda'(f(u))$
- $\forall\, u, v \in V, (u, v) \in E \Rightarrow (f(u), f(v)) \in E'$, and
- $\forall\, (u, v) \in E, \lambda(u, v) = \lambda(f(u), f(v))$

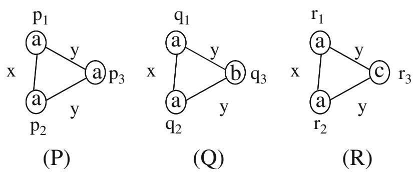
Definition 3 A **compatibility matrix** $M = (m_{i,j})$ is an $n \times n$ matrix indexed by symbols from a label set Σ ($n = |\Sigma|$). An entry $m_{i,j}$ ($0 \leq m_{i,j} \leq 1, \sum_j m_{i,j} = 1$) in M is the probability that the label i is replaced by the label j.

The compatibility matrix offers a probability framework for approximate subgraph mining. A compatibility matrix M is it stable if the diagonal entry is the largest one in the row (i.e. $M_{i,i} > M_{i,j}$, for all $j \neq i$). In a stable compatibility matrix, for any label i, it is likely to be replaced by itself rather than by any other symbols.

Most compatibility matrices in real-world applications are stable or almost-stable ones, and hence for the rest of this section, we only deal with the stable compatibility matrices.

Example 1 We show a graph database D with three labeled graphs P, Q, R on the left side of Fig. 1. In this database, the node label set is $\{a, b, c\}$ and the edge label set is $\{x, y\}$. On the right part of Fig. 1, we show a compatibility matrix M, which is a 2D matrix indexed by the set of node labels in D. The probability that the vertex label a is substituted by b is $m_{a,b} = 0.3$. In M, we use probability 0 to simplify the matrix. In reality, these probabilities are never 0.

Definition 4 A labeled graph $G = \{V, E, \Sigma_V, \Sigma_E, \lambda\}$ is **approximately subgraph isomorphic** to another graph $G' = \{V', E', \Sigma'_V, \Sigma'_E, \lambda'\}$ if there exists an injection $f : V \rightarrow V'$ such that



	a	b	c
a	0.5	0.3	0.2
b	0	1	0
c	0	0	1

M

Fig. 1 Example of a graph database D and a compatibility matrix M

- $\prod_{u\in V} \frac{M_{\lambda(u),\lambda'(f(u))}}{M_{\lambda(u),\lambda(u)}} \geq \tau$, and
- $\prod_{(u,v)\in E} \frac{M'_{\lambda(u,v),\lambda'(f(u),f(v))}}{M'_{\lambda(u,v),\lambda(u,v)}} \geq \tau'$

The injection f is an it approximate subgraph isomorphism between G and $G'.M$ is a compatibility matrix for node label sets $\Sigma_V \cup \Sigma'_V$. M' is a compatibility matrix for edge label sets $\Sigma_E \cup \Sigma'_E$. In an edge compatibility matrix, Σ_E and Σ'_E both contain a special label called empty edge. In this way, we handle both topology distortion (missing edges) and edge label mismatches in the same unified way through an edge compatibility matrix. τ $(0 < \tau \leq 1)$ is the threshold for node mismatch and τ' $(0 < \tau' \leq 1)$ is the threshold for edge mismatch.

For simplicity in the following discussion, we restrict our algorithmic study to cases that we only need to handle node label mismatches (i.e. corresponding edge relations and corresponding edge labels should exactly match each other in matching two graphs). One of such cases is protein structure pattern identification where edges encode geometric information.

With the assumption, the new definition of approximate subgraph isomorphism is:

Definition 5 A graph G is **approximate subgraph isomorphic** to another graph G', denoted by $G \subseteq_a G'$ if there exists an injection $f : V \rightarrow V'$, such that

- $\prod_{u\in V} \frac{M_{\lambda(u),\lambda'(f(u))}}{M_{\lambda(u),\lambda(u)}} \geq \tau$,
- $\forall\, u, v \in V, (u, v) \in E \Rightarrow (f(u), f(v)) \in E'$, and
- $\forall\, (u, v) \in E, \lambda(u, v) = \lambda(f(u), f(v))$

Given a node injection f from graph G to G', the co-domain of f is an it embedding of G in G'. The it approximate subgraph isomorphism score of f, denoted by $S_f(G, G')$, is the product of normalized probabilities: $S_f(G, G') = \prod \frac{M_{\lambda(u),\lambda'(f(u))}}{M_{\lambda(u),\lambda(u)}}$. For a pair of graphs, there may be many different ways of mapping nodes from one graph to another and hence may have different approximate isomorphism scores. The it approximate matching score (score for simplicity) between two graphs, denoted by $S(G, G')$, is the largest approximate subgraph isomorphism score, or

$$S(G, G') = \max_f\{S_f(G, G')\}.$$

Example 2 In Fig. 1, we show a graph database $D = \{P, Q, R\}$ and a compatibility matrix M. We set isomorphism threshold $\tau = 0.4$ and with this threshold, graph P is approximate subgraph isomorphic to graph Q with the approximate subgraph isomorphic score is $\frac{0.5\times0.3\times0.5}{0.5\times0.5\times0.5} = 0.6$. To see this, there are a total of 6 different ways to map nodes of P to those of Q. The only two that satisfy edge label constraints are $f_1 = p_1 \rightarrow q_1 p_2 \rightarrow q_2 p_3 \rightarrow q_3$ and $f_2 = p_1 \rightarrow q_2 p_2 \rightarrow q_1 p_3 \rightarrow q_3$. The approximate subgraph isomorphism score of f_1 equals that of f_2.

Definition 6 Given a graph database D, an isomorphism threshold τ, a support threshold $\sigma(0 < \sigma \leq 1)$, the **support value** of a graph G, denoted by $\sup_G$, is the average score of the graph to graphs in the database, which G is approximately subgraph isomorphic to:

$$\sup_G = \sum_{G' \in D, G \subseteq_a G'} S(G, G')/|D| \tag{1}$$

G is a it frequent approximate subgraph if its support value is at least σ. With this definition, we only use those graphs that a subgraph G is approximate subgraph isomorphic to (controlled by the parameter τ) to compute the support value of G. We do this to filter out low-quality (but potentially many) graph matchings in counting the support value of a subgraph. For a moderate sized graph database (100–1,000), according our experience, the number of frequent subgraphs identified is usually not sensitive to the isomorphism threshold, which makes sense since low-quality graph matching has low "weight" in the support computation nevertheless.

With the above definition, we have the support Apriori property as claimed by the following Theorem 1.

Theorem 1 *Given a graph database D and two graphs* $G \subseteq G'$*, we have* $\sup(G) \geq \sup(G')$.

Proof In order to prove the theorem, it is sufficient to show that for all graphs P in a graph database, we have $S(G, P) \geq S(G', P)$ for all graphs $G \subseteq G'$. This is true if the compatibility matrix is stable ($m_{i,i} > m_{i,j}$ for all $j \neq i$). The rest of the proof are trivial and are left to interested readers.

Problem Statement. Given a graph database D, an isomorphism threshold τ, a compatibility matrix M, and a support threshold σ, the it approximate subgraph-mining problem is to find all the frequent approximate subgraphs in D.

It is easy to adapt the frequent approximate subgraph-mining algorithm to the approximate clique subgraph mining by adding the full-connection constraint. In order to keep the consistency with our real-world applications, The subgraphs shown in all the examples below are clique subgraphs instead of subgraphs.

In Fig. 2, we show all the frequent approximate subgraphs in the graph database D shown in Fig. 1. By comparison with the frequent subgraphs acquired by exact graph mining, the approximate mining method identifies meaningful patterns that cannot be identified by exact graph-mining methods. Since the support value of approximate subgraph mining and that of

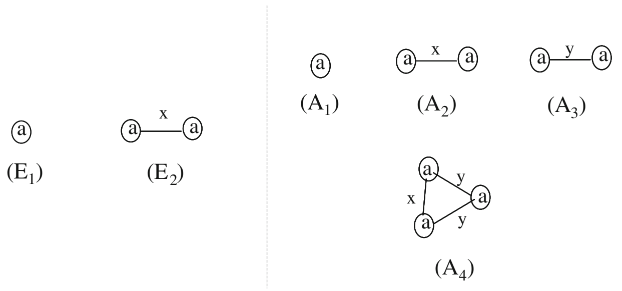


Fig. 2 Example of frequent subgraphs and approximate frequent subgraphs. Given the graph database D in Fig. 1 and the support threshold $\sigma = 2/3$, the *left side* shows the frequent subgraphs mined by the general exact graph mining. Given the compatibility matrix M in Fig. 1, isomorphism threshold $\tau = 0.4$, and support threshold $\sigma = 2/3$. *The right side* presents the frequent approximate subgraphs in D

frequent subgraph mining have different meaning, it is generally hard to do a comparison of approximate subgraph mining and that of frequent subgraph mining. Fortunately, with the assumption of stable compatibility matrix, we show that frequent subgraph mining is a special case of approximate subgraph mining with the following theorem.

Example 3 Given a graph database D, a compatibility matrix M in Fig. 1, the support threshold $\sigma = 2/3$ and isomorphism threshold $\tau = 0.4$, we show how to calculate the isomorphism score and support value for the approximate frequent patterns in Fig. 2.

$$S(A_1, P) = 1, S(A_1, Q) = 1, S(A_1, R) = 1, \mathrm{Sup}(A_1) = 3/3;$$
$$S(A_2, P) = 1, S(A_2, Q) = 1, S(A_2, R) = 1, \mathrm{Sup}(A_2) = 3/3;$$
$$S(A_3, P) = 1, S(A_3, Q) = 0.6, S(A_3, R) = 0.4, \mathrm{Sup}(A_3) = 2/3;$$
$$S(A_4, P) = 1, S(A_4, Q) = 0.6, S(A_4, R) = 0.4, \mathrm{Sup}(A_4) = 2/3.$$

Theorem 2 *Given a graph database D, an isomorphism threshold $\tau = 1$, a compatibility matrix M, and a support threshold σ, the set of approximate frequent subgraph patterns P_a is exactly the set of frequent subgraph patterns P_f or $P_a = P_f$. If $\tau < 1$, $P_f \subseteq P_a$.*

Proof This is the direct consequence of the support value definition 6.

4 Algorithm design

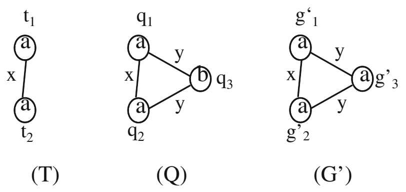
Here, we demonstrate a new algorithm APGM for approximate subgraph mining. APGM starts with frequent single-node subgraphs. At a subsequent step, it adds a node to an existing pattern to create new subgraph patterns and identify their support value. If none of the resulting subgraphs are frequent, APGM backtracks. APGM stops when no more patterns need to be searched. Before we proceed to the algorithmic details, we introduce the following definitions to facilitate the demonstration of the APGM algorithm.

Definition 7 Given a graph T, one of the embeddings $e = v_1, v_2, \ldots, v_k$ of T in a graph G, a node v is a **neighbor** of e if $\exists u \in e, (u, v) \in E[G]$.

In other words, a neighbor node of a embedding e is any node that connects to at least one node in e. The it neighbor set of an embedding e, denoted by $N(e)$, is the set of e's neighbors.

Definition 8 Given a graph T, one of the embeddings $e = (v_1, v_2, \ldots, v_k)$ of T in a graph G, an injection f from T to e, an isomorphism threshold τ, a compatibility matrix M, a node $v \in N(e)$, and a node label l, the **approximate subgraph pattern candidate**, denoted by $G|_{T,e,v,l}$, is a graph $(V', E', \Sigma_{V'}, \Sigma_{E'}, \lambda')$ such that

- $\lambda'(v) = l$
- $V' = \{v_1, v_2, \ldots, v_k\} \cup v$
- $E' \subseteq V' \times V' \cap E[G]$
- $\Sigma_{V'} = \Sigma_{V[T]}$
- $\Sigma_{E'} = \Sigma_{E[T]}$
- $\forall\, u, v \in V' : \lambda'((u, v)) = \lambda_G(f(u, v))$
- $\prod_{u \in V'} \frac{M_{\lambda'(u), \lambda_G(f(u))}}{M_{\lambda'(u), \lambda'(u)}} \geq \tau$



	a	b	c
a	0.5	0.3	0.2
b	0	1	0
c	0	0	1

M

Fig. 3 A pattern T, a graph Q and an approximate subgraph candidate G'

Example 4 In Fig. 3, we show a pattern T and one of its embedding $e = (q_1, q_2)$ in a graph Q. Node q_3 is a it neighbor node of e since it connects to at least one node of e (in fact both). Given a node label l="a", we obtain an approximate subgraph $G' = Q|_{T,e,v,l}$ of Q shown in the same Figure. The G' has an embedding $e' = (q_1, q_2, q_3)$ in Q and the score of the embedding is $\frac{M(a,a)}{M(a,a)} \frac{M(a,a)}{M(a,a)} \frac{M(a,b)}{M(a,a)} = \frac{M(a,b)}{M(a,a)} = 0.6$ (Recall the score of an embedding is the multiplication of the probability of the observed node label replacement, normalized by the probability of the node label self-replacement).

With the definitions, we present the pseudocode of APGM below.

```
APGM_MAIN($D$, $M$, $\tau$, $\sigma$)
1: Begin
2: $C \leftarrow$ { frequent single node }
3: $F \leftarrow C$
4: for each $T \in C$ do
5:   $APGM\_SEARCH(T, \tau, \sigma, F)$
6: end for
7: return $F$
8: End
```

```
APGM_SEARCH($T$, $\tau$, $\sigma$, $F$)
1: Begin
2: $C \leftarrow \emptyset$
3: for each $(e, v)$, $e$ is an embedding of $T$ in a graph $G$, $v \in N(e)$ do
4:   $CL \leftarrow$ approximateLabelSet($T$, $G$, $e$, $v$)
5:   for each $l \in CL$ do
6:     $X \leftarrow G|_{T,e,v,l}$
7:     $C \leftarrow C \cup \{X\}$
8:     $\mathcal{H}(X) = \mathcal{H}(X) \cup (e, v)$
9:   end for
10: end for
11: remove infrequent $T$ from $C$
12: $F \leftarrow F \cup C$
13: for each $T \in C$ do
14:   $APGM\_SEARCH(T, \tau, \sigma, F)$
15: end for
16: End
```

$\mathcal{H}$ is a hash function to store candidate subgraphs and their embeddings. The hash key of the function in our implementation is a canonical code of the subgraph X, which is a unique string representation of a graph. We use the Canonical Adjacency Matrix (CAM) and the Canonical Adjacency Matrix code, developed in [10], to compute the canonical code of a graph.

```
approximateLabelSet($T$, $G$, $e$, $v$)
1: Begin
2: $R \leftarrow \emptyset$
3: $l_0 \leftarrow \lambda_G(v)$
4: for each $l \in \Sigma_{V[G]}$ do
5:     if $S(e, T) \times \frac{M(l_0,l)}{M(l_0,l_0)} \geq \tau$ then
6:         $R \leftarrow R \cup l$
7:     end if
8: end for
9: return $R$
10: End
```

APGM enumerates the subgraph candidates from the new proposed embeddings. The procedure to find new embeddings are described in Definition 8. The information of neighbors are collected at the beginning of Algorithm APGM_MAIN. When all the new embedding are enumerated based on the embeddings of an existing subgraph, APGM has the new subgraph candidates and each candidate has all its embeddings. The support value of a new subgraph candidate is calculated by following Definition 4, 5 and 6. We gave one example below to show the enumeration procedure of patterns and their embeddings.

Example 5 Given a graph database with two graphs T and Q in Fig. 3, $\tau = 0.6$, and $\sigma = 1.5$, the enumeration procedure of subgraph candidates and embeddings are showed as the following. APGM first finds single-node pattern candidates: the frequent pattern a (Sup$=$2) with its embeddings (t_1) $(S = 1)$ and (t_2) $(S = 1)$ in graph T , and (q_1) $(S = 1)$, (q_2) $(S = 1)$, and (q_3) $(S = 0.6)$ in graph T; the infrequent pattern b (Sup$=$1) with its embeddings (q_3) $(S = 1)$ in graph Q. By adding one neighbor in the embeddings of frequent single-node pattern a, APGM enumerates a single edge pattern candidates: the frequent pattern $a - a$ with the embeddings (t_1, t_2) $(S = 1)$, (t_2, t_1) $(S = 1)$ in T and (q_1, q_2) $(S = 1)$, (q_2, q_1) $(S = 1)$in Q; the infrequent pattern $a - b$ (Sup$=$1) with the embeddings (q_1, q_3) $(S = 0.6)$ and (q_2, q_3) $(S = 0.6)$. APGM stops here since there is no more candidate patterns to explore.

4.1 Extensions

It is easy to adapt the above algorithm for mining other types of approximate subgraph such as approximate cliques, approximate quasi-cliques, approximate paths, and approximate trees. See “Appendix” for details about two extensions: that of approximate cliques and that of approximate quasi-cliques.

4.2 Optimization techniques

The key operations in the APGM algorithm are the operations at lines 4, 6, and 7 in Algorithm APGM_SEARCH. The function approximateLabelSet returns a set of labels at line 4, candidate subgraphs are created at line 6, and the embeddings of the candidates are stored at line 7. The performance of the APGM algorithm improves as the number of returned labels decreases, as the number of created candidate decreases, and as the number of stored embeddings decreases. This observation helps us devise the following optimizations. Our experimental results show that both optimization techniques do not affect the output approximate subgraph pattern sets.

4.2.1 Optimization 1. (focusing on line 4)

In order to reduce the symbols returned by the function approximateLabelSet, we designed the following rules regarding optimization in addition to that already implemented in the function approximateLabelSet:

- No infrequent labels may be returned. Infrequent labels are labels of single node whose support values are less than σ.
- Since our depth-searching paths start with single nodes that are sorted in canonical form, the new paths never include single nodes' label searched in previous paths.

We also set a rule that if a candidate pattern X is not in its canonical form, as defined in [10], we prune the candidate.

Theorem 3 *With the pruning technique in Optimization 1, APGM does not miss the frequent patterns.*

Proof (1) Prune the infrequent labels.

Suppose we have an infrequent single-node pattern X with the label l_1 and an existing frequent subgraph pattern Y. We propose a new pattern candidate Z by adding an extra node with the label l_1 to Y and have $X \subseteq_a Z$. With Theorem 1, we have $sup(X) \geq sup(Z)$. With the infrequent X, Z is infrequent pattern. Hence, to prune l_1 label does not change the frequent pattern output.

(2) Sort node labels in canonical form and exclude the labels in previous paths.

Suppose we have an ordered node label set $(l_1, \ldots, l_n)$. Starting with the frequent single-node graph with the label l_1, APGM finds all the patterns containing the label l_1 when it reaches its depth search limit of depth-first search (DFS) Tree. Since all the patterns containing the label l_1 are found, there is no need to include l_1 in the new pattern candidates. In the further search, with the new ordered label set $(l_2, \ldots, l_n)$, APGM applies the same rule. Hence, to exclude the previous searched node label from a sorted node label set in the further search, APGM does not miss frequent patterns.

As evaluated by our experimental study, Optimization 1 helps us significantly. We list the optimization here since it helps us to explain the next optimization.

4.2.2 Optimization 2. (focusing on lines 6 and 7)

Given a candidate graph X constructed from a pattern T, if we could estimate an upper-bound of the support value for X and the estimated upper-bound is less than the required support threshold σ, we can prune X and stop accumulating its embeddings. This strategy, which we call it early termination, certainly improves the computational efficiency of the APGM algorithm, given that we have a (tight) upper-bound estimation of a candidate graph pattern.

Toward that end, we have devised an efficient way to provide an upper-bound estimation of a candidate pattern. Our strategy is based on the following observation: given two graphs $T \subseteq X$ and a graph G, we have $S(T, G) \geq S(X, G)$ where $S(X, Y)$ is the matching score for a graph X to Y. In other words, the score of a supergraph X to a graph G is always less than or equal to that of its subgraph T with G (This observation is discussed in proving the support monotonicity in Theorem 1).

Within the context of depth-first search, when a candidate "grows" ($T \subseteq X$), the score value of the candidate to a fixed graph (and hence to a graph database) never increase ($S(X, G) \leq S(T, G)$).

With the observation in enumerating the embeddings of a graph T at line 3 of Algorithm APGM_SEARCH, we divide graphs in the graph database into two groups. The first group are those graphs that have been scanned and hence the (partial) support value of a candidate graph X in those graphs is known. The second group are those graphs that have not been scanned yet, and we estimate the (partial) support value of X in those graphs by the support value of T in those graphs (which is a known value). The overall support of X in a graph database is upper-bounded by the sum of the two partial support values. If the sum is below the support threshold, we know the candidate X cannot be frequent and can stop storing its embedding values.

Formally, given a candidate X that is proposed in processing the embeddings of a pattern T, we assume that $G_{j_1}, G_{j_2}, \ldots, G_{j_m}$ are the graphs that have been processed and $G'_{i_1}, G'_{i_2}, \ldots, G'_{i_n}$ are the remaining graphs. We estimate the support value of X by the following formula:

$$\tilde{\sup}(X) = \sum_k S(X, G_k) + \sum_k S(T, G'_k) \tag{2}$$

We claim that $\tilde{\sup}(X)$ is a upper-bound estimation of the real support value of X.

Theorem 4 *With the pruning technique in Optimization 2, APGM does not miss the frequent patterns.*

Proof With Theorem 1, we have
$\sum_k S(X, G_k) + \sum_k S(X, G'_k) \le \sum_k S(X, G_k) + \sum_k S(T, G'_k)$.
If $\sum_k S(X, G_k) + \sum_k S(T, G'_k) < \sigma$,
then $\sum_k S(X, G_k) + \sum_k S(X, G'_k) < \sigma$.
Hence, to prune the pattern candidates by using the upper boundary does not change the frequent pattern output.

Practically in implementing this early termination strategy, for a pattern T, we sort graphs in the related graph database according to the score values of T to the graphs from high to low. We then estimate the support value of any candidate X according to Eq. (2) and perform early termination if X is not frequent.

A tighter estimation can be obtained if we have information about other subgraph(s) of X. Intuitively, if we have score values of another subgraph $T' \subseteq X$ in the unprocessed graphs $G'_{i1}, G'_{i2}, \ldots, G'_{in}$, we have a tighter upper-bound estimation as the following:

$$\hat{\sup}(X) = \sum_k S(X, G_k) + \sum_k \min\{S(T, G'_k), S(T', G'_k)\}$$

We notice that with the depth-first search algorithm APGM_SEARCH, it is not always possible that we have score information for two subgraphs of a given candidate. For example, if the candidate X is a path with length 2, due to depth-first search, we only have score information for one of its two subgraphs. However, for many candidates X we do have score information for two of subgraphs of X. Whether we have two subgraphs of a candidate graph X has been extensively discussed in the Fast Frequent Subgraph-Mining algorithm [10].

5 Results

5.1 Experimental setup

We performed all the experiments on a cluster with 256 Intel Xeon 3.2 Ghz EM64T processors with 4 GB memory each. The approximate graph-mining algorithm was implemented in the C++ language and compiled by using the g++ compiler in Linux environment with -O3 optimization.

5.2 Data sets

We generated synthetic data set by the same synthetic graph generator as [18]. We downloaded all protein structures from Protein Data Bank (PDB). We followed [3] to use the same software as [14] to calculate Almost-Delaunay (AD) for graph representation of protein geometry. We took BLOSUM62 [22] as the compatibility matrix and back-calculated the conditional probability matrix by following the procedure described in [6]. In the BLOSUM62 substitution matrix, there is only one violation of the criterion of stable matrices—the row for methionine (MET). We normalized the row of MET with the maximum entry inside it and other rows in the matrix according to *Definition* 5.

5.2.1 Synthetic data sets

The synthetic graph generator takes the following set of parameters: D is the total number of graphs; T is the average size of graph; I is the average size of potentially frequent subgraphs; L is the number of potentially frequent subgraphs; V is the number of vertex labels; E is the number of edge labels. The default parameter values that we use are $D = 10000,\ T = 30,\ I = 11,\ L = 200,\ E = 20, and\ V = 20$.

5.2.2 Real data sets

We investigated two immunologically relevant protein domain families: the Immunoglobulin V set and the Immunoglobulin C1 set. Immunoglobulin domains are among the most valuable to give insight into host-defense mechanisms, and insight that can help guide development of therapies and vaccines against refractory organisms [17]. We collected proteins from SCOP release 1.69. For each family we created a culled set of proteins with maximal pairwise sequence identity percentage below 30% by using PISCES server [25]. The PDB ID of Individual proteins for two sets are shown in Table 1. The graph properties of two protein families are listed in Table 2. We denote Immunoglobulin domain proteins as positive sample and random proteins as negative.

5.3 Computational performance

5.3.1 Performance measurement

Since the support value of approximate subgraph mining and that of frequent subgraph mining have different meaning, it is generally hard to compare the performance of approximate subgraph mining and that of frequent subgraph mining. As indicated in Theorem 2, if we set the isomorphism threshold τ to 1, the set of frequent approximate subgraphs will be equal to the set of frequent subgraphs. If τ is less than 1, approximate subgraph mining will obtain

Table 1 Immunoevasins protein lists for research

	PDB ID of proteins in Immunoglobulin C1 set
Proteins for feature extraction (10):	$1fp5a$ $1onqa$ $1ogad$ $1pqza$ $1t7va$
	$1l6xa$ $1je6a$ $1mjul$ $1uvqb$ $1dn0b$
Proteins for leave-one-out testing (11):	$1nfda$ $1uvqa$ $1q0xl$ $1mjuh$ $1a6za$
	$1k5na$ $1hdma$ $3frua$ $1ogae$ $1hdmb$ $1k5nb$
	PDB ID of proteins in Immunoglobulin V set
Proteins for feature extraction (10):	$1pkoa$ $1ogad$ $1npua$ $1cdca$ $1jmaa$
	$1fo0b$ $1nkoa$ $1mjuh$ $1nfdb$ $1qfoa$
Proteins for leave-one-out testing (9):	$1zcza$ $1f97a$ $1eaja$ $1mjul$ $1cida$
	$1neua$ $1cdya$ $1hkfa$ $1nezg$

Table 2 Graph properties of immunoevasins proteins

	Immunoglobulin C1 set	Immunoglobulin V set
Avg. node size	220	158
Avg. edge size	3,107	2,263
Max. node size	276	159
Min. node size	100	96
Max. edge size	4,000	4,030
Min. edge size	1,350	713
Avg. density	14	14
Node label size	20	20
Edge label size	27	30

a superset than that of frequent subgraph mining. Compared with the exact matching, the approximate matching finds more patterns by enumerating more pattern candidates and takes more computational cost. Hence, the measurement of the time cost may not be a fair metric to evaluate the computational performance. We decided to use it pattern discovery rate (rate for simplicity) discussed in [12], which is computed as the number of discovered patterns N divided by the running time t. We use the rate rather than the running time as the criteria to compare computational efficiencies of different algorithms.

5.3.2 Performance testing with synthetic data sets

We created a synthetic data using the default parameters indicated in Sect. 5.2 for the graph generator. We compared the computational efficiency of APGM with an exact induced mining method MGM and showed the results in Fig. 4. For APGM, we have implemented O1 and O2 defined in Sect. 4.2. The experimental results in Fig. 4 show that $O1$ gave the most performance improvement. We notice that although $O2$ separately didn't show significant effect on APGM, when combined with $O1$, $O2$ improved the performance significantly. Our explanation is that $O2$ could not help prune repetitively enumerated frequent patterns since those patterns are frequent ones. When combined with $O1$, $O2$ does show a significant improvement. We see that APGM has similar performance with MGM if we use exact

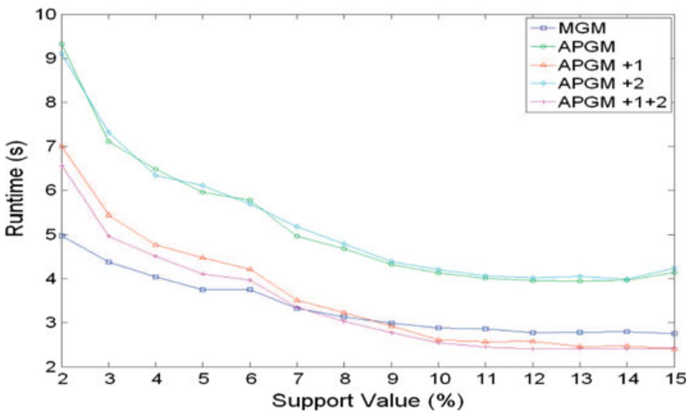


Fig. 4 Computational performance comparison between MGM and APGM with different optimization policies on synthetic data ($D10000$ $E20$ $I11$ $L200$ $T30$ $V20$). APGM used isomorphism threshold $\tau = 1.0$. Here APGM + 1 means APGM with $O1$

matching ($\tau = 1$). For subsequent experiments, we used APGM with both optimization O1 and O2.

We further compared the performance rate between APGM and MGM, with different isomorphism threshold values (and hence introduce different level of approximate matching) on the same synthetic data. From the upper part of Fig. 5, we find that with the change of isomorphism threshold, the performance rate change of APGM differ narrowly. Even if APGM takes approximate matching, its performance is very similar with MGM. In some range of support threshold, APGM with low isomorphism threshold ($\sigma = 0.6$) even has much higher rates.

Besides scalability test with varying support value, the scalability of APGM was also tested on the synthetic data sets with varying database size (D) or average size of graphs in a database (T), while other parameters keep default value as indicated in the experimental setup. In the middle of Fig. 5, we show the trend by increasing the value of D. We notice that even if the performance rate of APGM decreased rapidly at the beginning, it slowed down flat finally, which means APGM at least promised a minimum performance gain. And in the lower of Fig. 5, we also show the trend by increasing the value of T. The performance rate decreased approximately linearly. Our explanation is that the number of potential patterns in the graph database was a constant because the parameter (L) was not changed. With the increasing of the average size of graphs, the search space was expanded, which meant more running time. As the ratio of the number of patterns and the running time, the rate shows the trend of decreasing. But APGM still scales linearly with average size of graphs T. Both figures demonstrate the good scalability for APGM. In addition, compared with MGM, APGM showed the similar trend and isomorphism threshold τ did not affect the scalability.

5.4 Domain relevance

During this experimental research, we mined frequent clique subgraphs [12] in order to enforce biological constraints on the patterns. We compared APGM with the exact graph mining methods MGM. We chose MGM as the counterpart for the comparison because it is an available clique pattern-mining algorithm (any exact matching method with clique constraint should provide the same number of patterns from a graph database).

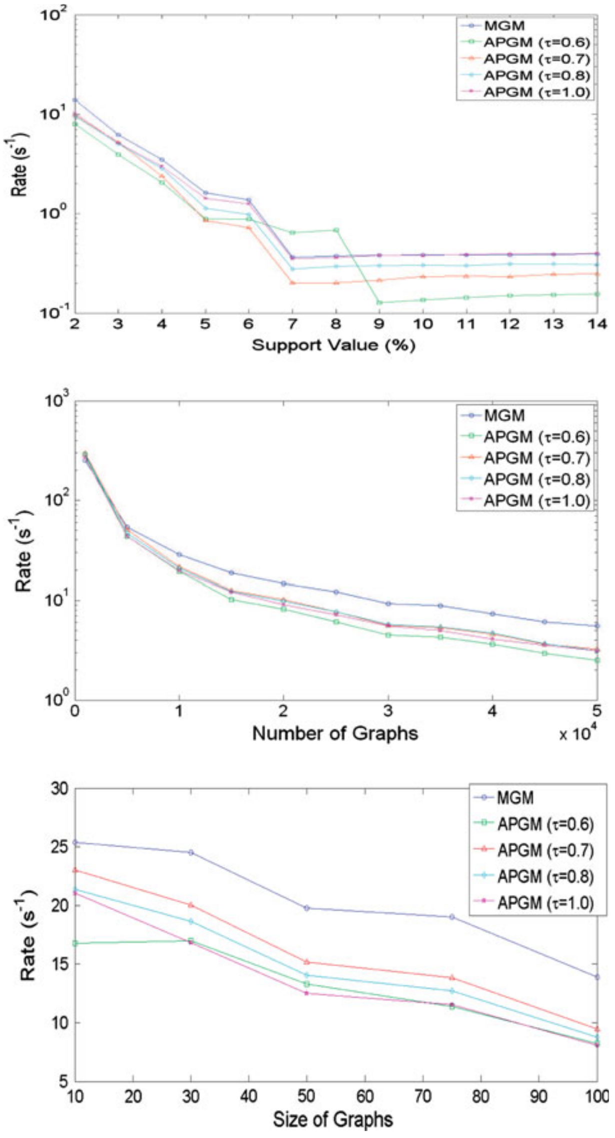


Fig. 5 Computational performance comparison between MGM and APGM using synthetic data sets. APGM used isomorphism threshold $\tau = 1.0, 0.8, 0.7, 0.6$. Given the patterns' number N and running time t (s), rate $= N/t$. *Left* The trend of performance rate with the increasing support threshold σ. *Middle* The trend of performance rate with the increasing size of graph database (D). APGM used support threshold $\sigma = 1\%$. *Right* The trend of performance rate with the increasing average size of graphs in database (T). APGM used support threshold $\sigma = 1\%$

5.4.1 Experimental protocol

We created our experimental protocol as the following:

- We randomly chose 10 proteins from each family as group I to serve as sources for feature extraction.

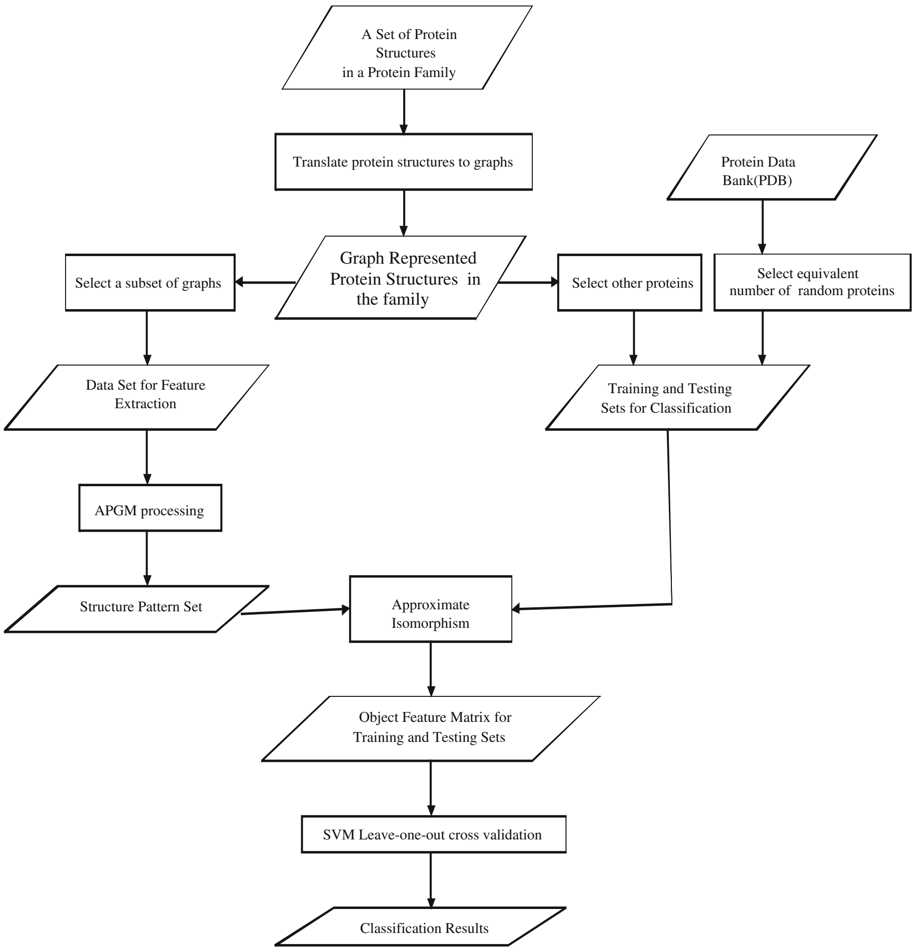

A Set of Protein Structures in a Protein Family
Translate protein structures to graphs
Protein Data Bank(PDB)
Select a subset of graphs
Graph Represented Protein Structures in the family
Select other proteins
Select equivalent number of random proteins
Data Set for Feature Extraction
Training and Testing Sets for Classification
APGM processing
Structure Pattern Set
Approximate Isomorphism
Object Feature Matrix for Training and Testing Sets
SVM Leave-one-out cross validation
Classification Results

Fig. 6 The procedure of experimental research

- We used the remainder (positive sample) as group II for training and testing in “leave-one-out” cross validation.
- A negative sample set of the same size as the positive samples in group II was randomly chosen from PDB. The negative sample was used along with the positive sample in training and testing.

The complete flowchart of our experiment procedure is shown in Fig. 6.

- In order to eliminate the effect of randomness on our classification results, we chose the optimal parameters to repeat the procedure shown in Fig. 6, 100 times for each data set.

5.4.2 Number of patterns identified

We identified frequent approximate subgraph patterns from 10 positive proteins in each family. There are two parameters that may have significant influence on the set of mined patterns. The first is the support threshold (σ), and the second is the isomorphism threshold (τ).

Table 3 Number of patterns for Immunoglobulin C1 set acquired by APGM

	$\tau = 3.5$	$\tau = 4.5$	$\tau = 5.5$
$\sigma = 4$	811	774	750
$\sigma = 5$	141	140	136
$\sigma = 6$	17	17	17

Table 4 Number of patterns by APGM ($\tau = 0.35$) and MGM on Immunoglobulin C1

	Support threshold (σ)				
	6	5.5	5	4.5	4
APGM ($\tau = 0.35$)	17	24	141	202	841
MGM	16	16	126	126	660

Table 5 Number of patterns by APGM ($\tau = 0.75$) and MGM on Immunoglobulin V

	Support threshold (σ)				
	6	5.5	5	4.5	4
APGM ($\tau = 0.75$)	0	0	0	160	14,686
MGM	0	0	0	0	13,911

For simplicity, in the following experiments in this section we use the new support threshold $\sigma' = \sigma \times |D|$, where $|D|$ is the size of the graph database, and applied the same change in support value. In Table 3, we run APGM with different combinations of τ and σ and collect the total number of identified patterns. Our results show that the total number of patterns is not sensitive to the isomorphism threshold, and depends on the support threshold heavily. Such fact eases the worry that the parameter τ may be too strong for deciding the number of patterns.

For the purpose of comparison, the patterns mined by two mining methods are shown in Tables 4 and 5, and the patterns acquired by APGM from Immunoglobulin C1 proteins are also shown in Table 3. In our experiment, we treat a pattern set with the number more than 10,000 as a meaningless one because our sample space is comparatively small and the isomorphism check is computationally expensive. From Table 5, we see that exact matching fails to provide useful patterns on the Immunoglobulin V proteins, which is the typical data set with very noisy background. In comparison, APGM does find some pattern set with a reasonable size in such situation (We only use rough parameter combination grids to do the pattern search. If we increase the precision of τ and σ , more patterns will be found). In order to evaluate the quality of these patterns, we use the identified frequent subgraphs in classification tests as discussed below.

5.4.3 Classification performance

In this experimental section, we used *libsvm* SVM package (http://www.csie.ntu.edu.tw/thicksimcjlin/libsvm) for protein structure classification. We treat each mined pattern as a feature and a protein is represented as a feature vector $V = (v_i)$ where $1 \leq i \leq n$ and n is the total number of identified features. v_i is 1, if the related feature occurs in the protein and otherwise v_i is 0. We used the linear kernel and default parameters for SVM leave-one-out cross validation, where Accuracy $= (TN+TP)/(TN+TP+FN+FP)$ (TP, true positive; FP, false positive; TN, true negative; FN, false negative).

Table 6 Classification accuracy of APGM ($\tau = 0.35$) and MGM on Immunoglobulin C1 set

	Support threshold (σ)				
	6	5.5	5	4.5	4
APGM	68.18%	77.27%	86.36%	90.91%	81.82%
MGM	72.73%	72.73%	72.73%	72.73%	72.73%

Table 7 Classification accuracy of APGM ($\tau = 0.75$) and MGM on Immunoglobulin V Set

	Support threshold (σ)			
	6	5.5	5	4.5
APGM	–	–	–	77.78%
MGM	–	–	–	–

TP true positive, *FP* false positive, *TN* true negative, *FN* false negative
Accuracy = (TN + TP)/(TN + TP + FN + FP)
− means accuracies are unavailable

We followed the procedure in Fig. 6 to create one data set for feature extraction and another for training and testing on both Immunoglobulin C1 and V proteins. The classification results are summarized in Tables 6 and 7. For some parameter combinations, there are no accuracies—an event that happens under two circumstances. First, there are no patterns found. Second, the pattern set is too big to be useful. From the tables, we see that the classification with APGM-based feature highly outperforms those based on exact matching. For Immunoglobulin C1 set, the classification based on feature identified by MGM only reaches 73%, while APGM is between 69 and 91%. For Immunoglobulin V set, since the exact matching method cannot mine any meaningful patterns, it fails in classification, while by using APGM, we have the accuracy about 78%. It shows that our APGM has more capability to mine useful structure information from very noisy background than general exact matching graph-mining algorithms.

We repeated the experimental procedure 100 times for both protein families. We showed the results of average Accuracy and its variance in Figs. 7 and 8, and the results of average Precision and Recall and their variance in Tables 6 and 7. In all of three classification measures, APGM outperformed the exact matching method MGM, which demonstrates our previous finding in the previous single experiment (Tables 8, 9).

5.4.4 Significance of patterns

In order to further demonstrate the quality of the patterns mined by APGM, we chose the parameter combination with the best accuracy for the Immunoglobulin C1 proteins and the Immunoglobulin V proteins to check the distribution and significance of patterns. Figure 9 shows the number of the patterns that the 11 Immunoglobulin C1 proteins contain and the significance scores. Figure 10 shows those for the 9 Immunoglobulin V proteins. Proteins in Figs. 9 and 10 are numbered according to their appearance order in Table 1. For example, protein "10" in Fig. 9 is protein 1nfa (chain A). The proteins in Figs. 9 and 10 are sorted according to the number of patterns contained in the proteins. The significance score P of a pattern is defined as follows.

$$P = \log \frac{f^+/N^+}{f^-/N^-}, \quad \text{if } f^- \neq 0 \quad f^+ \neq 0 \tag{3}$$

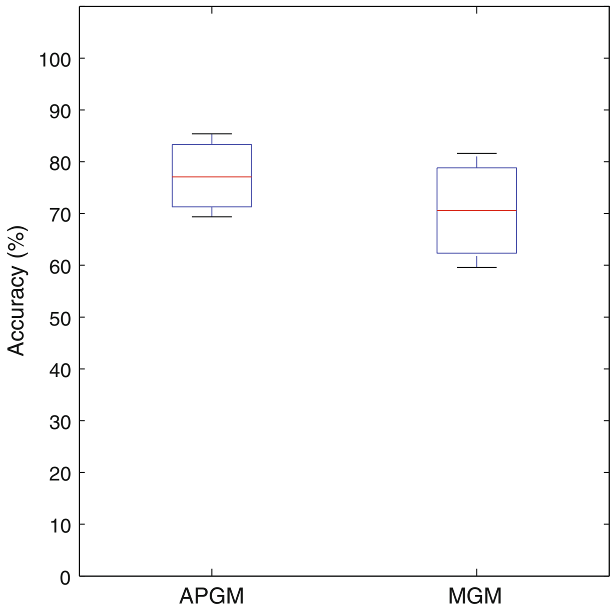


Fig. 7 The accuracy comparison between APGM and MGM on Immunoglobulin C1 set

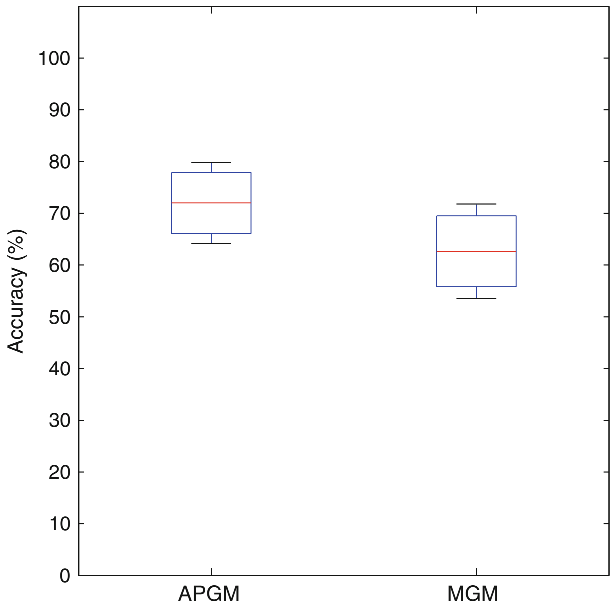


Fig. 8 The accuracy comparison between APGM and MGM on Immunoglobulin V set

N^- is the number of negative samples; N^+ is the positive samples; f^- is the number of negative samples that contain the pattern; f^+ is the number of positive samples that contain the pattern. There are three special cases of P's value. If $f^- = 0$ and $f^+ \neq 0$, we set

Table 8 Prediction comparison between APGM ($\tau = 0.35$) and MGM on Immunoglobulin C1

	Precision (avg. ± variance)%	Recall (avg. ± variance)%
APGM	87.87 ± 7.96	62.50 ± 12.40
MGM	86.79 ± 13.35	48.21 ± 16.05

Table 9 Prediction comparison between APGM ($\tau = 0.75$) and MGM on Immunoglobulin V

	Precision (avg. ± variance)%	Recall (avg. ± variance)%
APGM	92.90 ± 11.63	47.57 ± 13.24
MGM	86.26 ± 17.72	30.53 ± 13.67

Precision = TP/(TP + FP)
Recall = TP/(TP + FN)
For the C1 set, APGM chose two optimal parameter combinations ($\tau = 0.35, \sigma = 4.5$) and ($\tau = 0.35, \sigma = 5$), and MGM chose two optimal parameters $\sigma = 5, 6$. In 200 mining times, APGM found 200 non-empty pattern sets and MGM found 185. For the V set, APGM chose two optimal parameter combinations ($\tau = 0.75, \sigma = 4.5$) and ($\tau = 0.75, \sigma = 5$), and MGM chose two optimal parameters $\sigma = 5, 6$. In 200 mining times, APGM found 192 non-empty pattern sets and MGM found 135

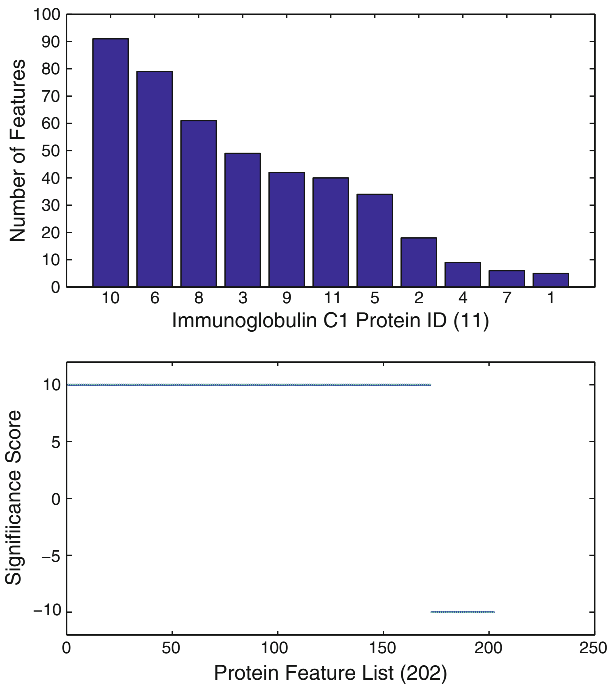


Fig. 9 *Upper* Distribution of frequent subgraph features among Immunoglobulin C1 proteins. *Lower* Significance of frequent subgraph features among Immunoglobulin C1 proteins. Both figures are constructed for the set for classification. There are 202 patterns that are mined with the support threshold $\sigma = 4.5$ and the isomorphism threshold $\tau = 0.35$

$P = 10$; if $f^- \neq 0$ and $f^+ = 0$, we set $P = -10$; and if $f^- = 0$ and $f^+ = 0$, we set $P = 0$.

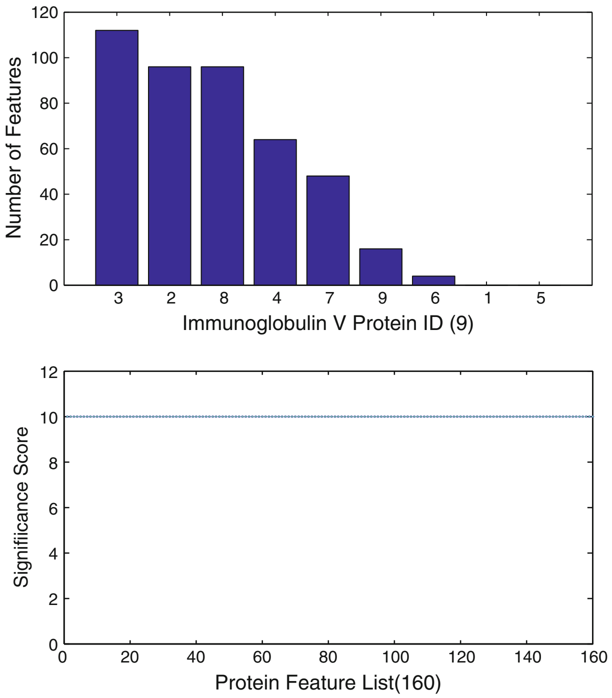


Fig. 10 *Upper* Distribution of frequent subgraph features among Immunoglobulin V proteins. *Lower* Significance of frequent subgraph features among Immunoglobulin V proteins. Both figures are constructed for the set for classification. There are 160 patterns that are mined with the support threshold $\sigma = 4.5$ and the isomorphism threshold $\tau = 0.75$

Although the patterns do not distribute uniformly among Immunoglobulin C1 proteins, they cover all the positive proteins. The significance score of these patterns shows strong bias toward the Immunoglobulin C1 proteins, and among 202 only 30 noise features ($P = -10$) exist. For Immunoglobulin V proteins, the features miss two positive proteins, but these features are highly correlated with positive samples with all P equaling 10.

6 Conclusion and future work

In this paper, we present a novel data-mining method APGM (APproximate Graph Mining). Instead of using exact matching for graph comparison, we developed a graph-mining algorithm with approximate matching policy. We took advantage of known substitution matrices as the prior domain knowledge and incorporated them into a general framework to evaluate qualified frequent induced graph patterns. We tested this method on the field of structure motif identification in diverse proteins. In that application, we encoded structural motifs as subgraphs of geometric graph of proteins and utilized biological mutation matrices as our domain knowledge base. We also enforced the biological constraints on our structure patterns. Through our experimental research, we found, compared with general graph-mining algorithms, APGM not only offers more qualified patterns that achieves higher classification accuracy, but also shows a reasonable pattern discovery rates.

In biological research, the compatibility matrices on amino acid mutation are widely used and publicly available, such as PAM 30, PAM70, BLOSUM45, and BLOSUM62. Without loss of generality, choice of appropriate compatibility matrices allows our method to be employed in any domain where subgraph labels have some uncertainty. For example, in the domain of social networks, networks of personal contacts “mutate” as people die or change employment. Compatibility matrices assigning probabilities of ‘label substitution’ within families or organizations may allow the essential natures of personal contact subgraphs to be preserved nevertheless. Furthermore, without loss of generality, choice of appropriate compatibility matrices allows our method to be employed in any domain where subgraph labels have some uncertainty. We believe that our method and framework can be easily scalable into different domain applications.

Acknowledgments We thank Dr. Carayannopoulos from the School of Medicine in the University of Washington for his help in our research. This work is supported by NSF IIS award 0845951.

Appendix: Extensions

Here, we show two extensions of APGM: that of approximate cliques and that of approximate quasi-cliques below.

Approximate cliques

A pattern is a it frequent approximate clique if it is a frequent approximate subgraph and it is a clique (i.e. fully connected graph). In order to adapt the above algorithm to identify approximate cliques, we only need to modify the definition of neighbor node of an embedding by requiring the neighbor node connects to each and every node in an embedding, or:

Definition 9 Given a clique C, one of the embeddings $e = v_1, v_2, \ldots, v_k$ of C, a node v is a **clique neighbor** of e if $\forall u \in e, (u, v) \in E[G]$.

The it clique neighbor set of an embedding e, denoted by $N_C(e)$, is the set of e’s clique neighbors. If we replace the neighbor set $N(e)$ used at line 3 of Algorithm APGM_SEARCH with that of clique neighbor set, we obtain frequent approximate cliques.

Approximate quasi-cliques

A pattern P is it quasi-clique if for all nodes $u \in V[P]$ we have $d(v) \geq k(|P|-1)$ $(0 < k \leq 1)$ [23]. We use $d(v)$ to denote the degree of a node v (the number of node that connects to v directly). As studied in [23], there is a connection between a quasi-clique and the diameter of a graph. For example, for quasi-clique with $k = 0.5$ (every node is connected to at least half of the rest of the nodes in a graph), the diameter of the graph is at most 2.

Definition 10 Given a clique C, one of the embeddings $e = v_1, v_2, \ldots, v_k$ of C, a node v is a **quasi-clique neighbor** of e with distance x if $\forall u \in e, \mathcal{X}(u, v) \in E[G]$ and $\exists u \in e, (u, v) \in E[G]$.

where $\mathcal{X}(u, v)$ is the shortest distance between two nodes u and v. The it quasi-clique neighbor set of distance x of an embedding e, denoted by $N_{Cx}(e)$, is the set of e’s quasi-clique neighbors with distance x.

If we replace the neighbor set $N(e)$ used at line 3 of Algorithm APGM_SEARCH with that of quasi-clique neighbor set, we obtain frequent approximate quasi-cliques.

References

1. Aggarwal CC (2009) Managing and mining uncertain data. Springer, Berlin
2. Aggarwal CC, Li Y, Wang J, Wang J (2009) Frequent pattern mining with uncertain data. In: Proceedings of the 2009 ACM SIGKDD international conference on Knowledge discovery and data mining (SIGKDD'09), pp 29–37
3. Bandyopadhyay D, Snoeyink J (2004) Almost-Delaunay simplices: nearest neighbor relations for imprecise points. In: ACM-SIAM symposium on distributed algorithms, pp 403–412
4. Chan J, Bailey J, Leckie C (2008) Discovering correlated spatio-temporal changes in evolving graphs. Knowl Inf Syst 16(1):53–96
5. Chen C, Yan X, Zhu F, Han J (2007) Gapprox: mining frequent approximate patterns from a massive network. In: Proceedings of the 2007 international conference on data mining (ICDM'07)
6. Eddy SR (2004) Where did the blosum62 alignment score matrix come from. Nat Biotechnol 22:1035–1036
7. Han J, Cheng H, Xin D, Yan X (2007) Frequent pattern mining: current status and future directions. Data Min Knowl Discov 14
8. Holder LB, Cook DJ, Djoko S (1994) Substructures discovery in the subdue system. In: Proceedings of AAAI'94 workshop knowledge discovery in databases, pp 169–180
9. Hu H, Yan X, Huang Y, Han J, Zhou XJ (2005) Mining coherent dense subgraphs across massive biological networks for functional discovery. In: Proceedings of the 2005 international conference on intelligent systems for molecular biology (ISMB'05)
10. Huan J, Wang W, Prins J (2003) Efficient mining of frequent subgraph in the presence of isomorphism. In: Proceedings of the 2003 IEEE international conference on data mining (ICDM'03), pp 549–552
11. Huan J, Wang W, Prins J (2003) Efficient mining of frequent subgraphs in the presence of isomorphism. In: Proceedings of the 2003 international conference on data mining (ICDM'03)
12. Huan J, Bandyopadhyay D, Snoeyink J, Prins J, Tropsha A, Wang W (2006) Distance-based identification of spatial motifs in proteins using constrained frequent subgraph mining. In: Proceedings of the IEEE computational systems bioinformatics
13. Huan J, Prins J, Wang W, Carter C, Dokholyan NV (2006) Coordinated evolution of protein sequences and structures with structure entropy. In: Computer Science Department Technical Report
14. Huan J, Wang W, Bandyopadhyay D, Snoeyink J, Prins J, Tropsha A (2004) Mining family specific residue packing patterns from protein structure graphs. In: Proceedings of the 8th annual international conference on research in computational molecular biology (RECOMB), pp 308–315
15. Huan J, Wang W, Prins J, Yang J (2004) Spin: mining maximal frequent subgraphs from graph databases. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining, pp 581–586
16. Inokuchi A, Washio T, Motoda H (2000) An apriori-based algorithm for mining frequent substructures from graph data. In: Proceeding of 2000 practice of knowledge discovery in databases conference (PKDD'00), pp 13–23
17. Judson KA, Lubinski JM, Jiang M, Chang Y, Eisenberg RJ, Cohen GH, Friedman HM (2003) Blocking immune evasion as a novel approach for prevention and treatment of herpes simplex virus infection. J Virol 77:12639–12645
18. Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: Proceedings of the 2001 international conference on data mining (ICDM'01), pp 313–320
19. Lahiri M, Berger-Wolf TY (2009) Periodic subgraph mining in dynamic networks. Knowl Inf Syst (online first 09/2009)
20. Lahiri M, Berger-Wolf TY (2007) Structure prediction in temporal networks using frequent subgraphs. Computat Intell Data Min, pp. 35–42
21. Nijssen S, Kok JN (2004) A quickstart in frequent structure mining can make a difference. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining, pp 647–652
22. Orengo CA, Michie AD, Jones S, Jones DT, Swindells MB, Thornton JM (1997) CATH—a hierarchic classification of protein domain structures. Structure 5(8):1093–1108
23. Pei J, Jiang D, Zhang A (2005) Mining cross-graph quasi-cliques in gene expression and protein interaction data. ICDE, pp 353–354
24. De Raedt L, Kramer S (2001) The levelwise version space algorithm and its application to molecular fragment finding. In: IJCAI'01: seventeenth international joint conference on artificial intelligence, vol 2, pp 853–859
25. Wang G, Dunbrack RL Jr (2003) PISCES: a protein sequence culling server. Bioinformatics 19:1589–1591

26. Weng C-H, Chen Y-L (2010) Mining fuzzy association rules from uncertain data. Knowl Inf Syst 23(2):129–152
27. Yada K, Motoda H, Washio T, Miyawaki A (2004) Consumer behavior analysis by graph mining technique. Lecture Notes in Computer Science, pp 800–806
28. Yan X, Han J (2002) gspan: graph-based substructure pattern mining. In: Procceeding of international conference on data mining (ICDM'02), pp 721–724
29. Yan X, Zhu F, Yu PS, Han J (2006) Feature-based substructure similarity search. ACM Trans Database Syst 31(4):1418–1453
30. Zhang S, Yang J (2008) Ram: randomized approximate graph mining export. Scientific and Statistical Database Management
31. Zhang S, Yang J, Cheedella V (2007) Monkey: approximate graph mining based on spanning trees. In: Proceeding of IEEE 23rd international conference data engineering (ICDE'07), pp 1247–1249
32. Zou Z, Li J, Gao H, Zhang S (2009) Frequent subgraph pattern mining on uncertain graph data. In: Proceedings of the 2009 conference on information and knowledge management (CIKM'09), pp 583–592

Author Biographies

Yi Jia received a B.S. in Mechanical Engineering from Shanghai Jiao Tong University, Shanghai, China, in 1999, a M.S. degree in Computer Science from Shanghai Jiao Tong University, Shanghai, China, in 2003, and a M.S. degree in Information Science and Systems Engineering from Ritsumeikan University, Kusatsu, Japan, in 2005. He is currently working toward the Ph.D. degree in Computer Science at the University of Kansas, Kansas, US. His research interests include graph mining, probabilistic graphical models and their applications in protein functional annotation and gene regulatory network structure inference.

Jintao Zhang has been a Ph.D. student in the Center for Bioinformatics at University of Kansas since Fall 2006, and joined Dr. Huan's research group in summer 2007 as a teaching/research assistant, with research interest on data mining in bioinformatics and chemical biology. Mr. Jintao Zhang received his Bachelor degree in Chemical Physics from the University of Science & Technology of China in 2001, and a Master degree in Chemistry from University of California, Riverside in 2005.

Jun Huan has been an assistant professor in the Electrical Engineering and Computer Science department at the University of Kansas since 2006. He is an affiliated member of the Information and Telecommunication Technology Center (ITTC), Bioinformatics Center, Bioengineering Program, and the Center for Biostatistics and Advanced Informatics–all KU research organizations. Dr. Huan received his Ph.D. in Computer Science from the University of North Carolina at Chapel Hill in 2006. Before joining KU, he worked at the Argonne National Laboratory (with Ross Overbeek) and the GlaxoSmithKline plc (with Nicolas Guex). Dr. Huan was a recipient of the NSF Faculty Early Career Development (CAREER) Award in 2009. He serves on the program committees of leading international conferences including ACM SIGKDD, IEEE ICDE, ACM CIKM, IEEE ICDM.