REAFUM: Representative Approximate Frequent Subgraph Mining

Ruirui Li[*] Wei Wang[*]

Abstract

Noisy graph data and pattern variations are two thorny problems faced by mining frequent subgraphs. Traditional exact-matching based methods, however, only generate patterns that have enough perfect matches in the graph database. As a result, a pattern may either remain undetected or be reported as multiple (almost identical) patterns if it manifests slightly different instances in different graphs. In this paper, we investigate the problem of approximate frequent pattern mining, with a focus on finding non-redundant representative frequent patterns that summarize the frequent patterns allowing approximate matches in a graph database. To achieve this goal, we propose the REAFUM framework which (1) first extracts a list of diverse representative graphs from the database, which may contain most approximate frequent patterns exhibited in the entire graph database; (2) then uses distinct patterns in the representative graphs as seed patterns to retrieve approximate matches in the entire graph database; (3) finally employs a consensus refinement model to derive representative approximate frequent patterns. Through a comprehensive evaluation of REAFUM on both synthetic and real datasets, we show that REAFUM is effective and efficient to find representative approximate frequent patterns and REAFUM is able to find patterns that much better resemble the ground truth in the presence of noise and errors, and are less redundant than that from any exact-matching based methods.

1 Introduction

Mining frequent patterns in a set of graphs has attracted much research interest due to its wide applications in, for example, bioinformatics [13], cheminformatics [7, 24], social network analysis [17], and hardware design [4, 5]. Given a set S of labeled graphs, the support of a graph g is the fraction of all graphs in S of which g is a subgraph. Graph g is frequent iff the support of g meets a certain support threshold. The problem of frequent subgraph mining is to find all connected subgraphs that are frequent in a graph database. Most existing frequent subgraphs mining approaches are based on exact matching [14, 26], which require the topology and labels be identical between a subgraph and its instance in another graph. Exact-matching based pattern mining algorithms have serious limitations. First, some important subgraph patterns may exhibit some slight differences in different graphs. For example, a protein may have accumulated a few mutations through evolution which may lead to small structural variations but do not alter the protein's function [1]. Graph mining approaches based on exact matching will miss these patterns when all the corresponding variations are below the support threshold. Second, almost all data contain noise. For example, in image processing, the two instances of the same object may exhibit slight differences due to detection errors [1]. These noise and errors in data may lower the observed frequency of some patterns and prevent them from being discovered.

Figure 1 shows a small graph database of 6 graphs in which six subgraph patterns are embedded. Each subgraph

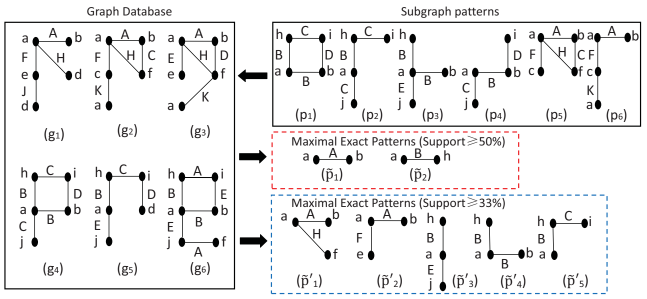


Figure 1: Exact patterns mining

pattern has two or more variations embedded in these graphs. Subgraphs within the red dashed box are the maximal patterns found by exact-matching based pattern mining algorithms with support threshold 50%. It is not surprising to see that the two patterns $\tilde{p}_1$ and $\tilde{p}_2$ are substantially different from the embedded subgraphs. Even if we relax the support threshold to 33%, as shown in the lower right corner of Figure 1, all the embedded subgraph patterns are still missing.

This example demonstrates the need to go beyond exact pattern matching. We need to tolerate certain level of structural and/or label differences in two graphs, as long as such differences are within a clearly defined threshold. Such approximate pattern mining framework is expected to find patterns that are missed by exact pattern mining algorithms. The need of approximate matching is also pronounced in document clustering tasks [12], chemical data processing [2], and molecule data processing [19].

Approximate-matching based pattern mining is far more challenging than exact-matching based pattern mining. First, given a pattern candidate, checking whether one database graph contains its approximate instances or not is at least as hard as the subgraph isomorphism challenge, which is a NP-complete problem [6]. Second, since variations of the same pattern may likely be approximate matches of each other, many such variations may exceed the support threshold. A straightforward model that outputs every pattern whose approximate matches exceed the support threshold may thus produce redundant patterns. An effective model that summarizes these redundant patterns is needed. Third, allowing approximate matches may lead to larger patterns than the exact match patterns (as demonstrated in Figure 1). Consequently, the number of non-maximal patterns grows too. This may lead to longer computation time and larger memory usage.

To address the challenges, we propose a representative approximate frequent subgraph mining algorithm REAFUM. Figure 2 shows the framework. In our model, we first select a list of m diverse graphs sequentially as representative graphs. To ensure diversity, each time we select a graph that is most distant from any selected representative graphs yet most similar to the remaining graphs. Intuitively, each representative graph corresponds to a "mode" of a set of similar graphs in the graph space. For example, given the graph database D of 6 graphs in Figure 1, to pick 2

[*]Dept of Computer Science, University of California Los Angeles, CA, USA. {rrli,weiwang}@cs.ucla.edu

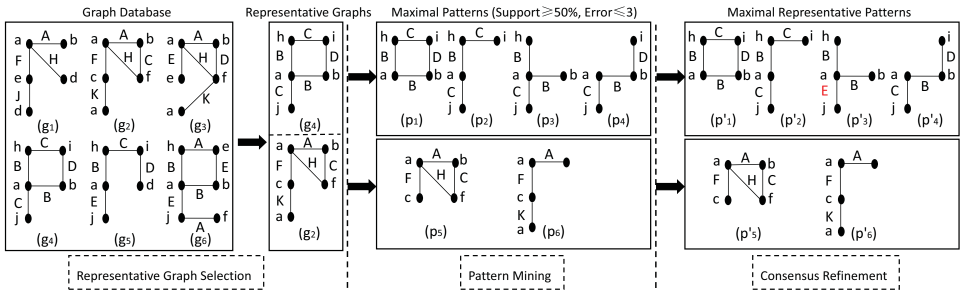

Graph Database
Representative Graphs
Maximal Patterns (Support≥50%, Error≤3)
Maximal Representative Patterns
Representative Graph Selection
Pattern Mining
Consensus Refinement

Figure 2: The pattern mining framework of REAFUM

representative graphs, g_4 and g_2 are sequentially selected. g_4 is the mode among graphs in $\{g_1, g_2, g_3, g_4, g_5, g_6\}$ and g_2 is the mode among the remaining graphs that cannot be well represented by g_4.

These representative graphs collectively provide a comprehensive yet non-redundant coverage of approximate patterns in the graph database. Each representative graph well represents a subset of similar graphs and likely contains an instance of any frequent subgraph pattern in this set of similar graphs. After representative graph selection, pattern candidates are enumerated and examined in each representative graph in a topological order. Only qualified pattern candidates in the representative graphs will be used to seed the approximate match exploration in the entire graph database. Variations of the same patterns will be examined all together when retrieving approximate matches in the graph database and will be summarized in the refinement step into one consensus pattern that optimally represents the set of variants.

Here we summarize the major contributions of this paper:

1. REAFUM extends the previous graph mapping distance metric to handle graphs with edge labels.
2. REAFUM is the first to allow both structural and label approximations in mining frequent subgraphs.
3. We propose an efficient algorithm to mine approximate frequent patterns in a graph database.
4. We present a comprehensive empirical evaluation of REAFUM with exact pattern mining methods using both synthetic and real datasets. The results show that REAFUM is highly effective and efficient in detecting patterns based on approximate-matching.

2 Related Work

Frequent subgraph mining is an active research topic. In this section we give a brief account of some representative works on exact pattern mining (Section 2.1). Since our approach aims to find approximate patterns, we highlight some related work on pattern mining approaches which allow approximate matches (Section 2.2). To distinguish our work from existing representative pattern mining approaches, we further discuss some works on pattern summarization (Section 2.3).

2.1 Exact Pattern Mining Exact pattern mining approaches strictly require the pattern and its instances be exactly the same in terms of their labels and structures. Most existing exact pattern mining approaches can be roughly divided into two categories based on their pattern enumeration orders, namely breadth-first order [15, 21] and depth-first order [14, 23, 26]. For the methods with breadth-first order, AGM [15] generates pattern candidates by adding nodes to the patterns. FSG [21] generates pattern candidates by using edge-growth instead of vertex-growth. For the methods with depth-first order, gSpan [26] designs a DFS lexicographic order to support the mining algorithm. FFSM [14] develops a new graph canonical form and completely avoids subgraph isomorphism testing by maintaining an embedding set for each frequent subgraph. Gaston [23] adopts a step-wise approach using a combination of frequent paths, frequent free trees and cyclic graphs mining to discover all frequent subgraphs.

The drawback of these exact pattern mining approaches is that they will miss some patterns if all variations of the pattern are below the support threshold. In addition, noise in the data may also lead the frequency of the pattern lower than the support threshold.

2.2 Approximate Pattern Mining A few works focus on mining patterns allowing inexact matches. The APGM algorithm was proposed in [16] to mine frequent patterns considering a scenario where noisy data leads to wrong labels in vertices. To address the problem, APGM requires a substitution matrix M as an extra input, where the entry M_{ij} gives the probability of label i being mistaken by label j. Two graphs are defined as approximately isomorphic if their similarity (measured by a product of probabilities) is below a given threshold. VEAM [1] extends APGM to allow label replacements of both vertexes and edges. Both of these algorithms only allow label replacements and require the matching graphs have the same topology. In addition, they both require a prior knowledge of the substitution matrix, which is only available in a few applications of bioinformatics.

RAM [28] finds patterns whose instances are allowed to have missing edges. More precisely, the graph will be regarded as an approximate instance of the pattern iff this graph only misses at most β edges from the pattern, where β is an input. It does not allow any distortion of the vertices. It does not find patterns whose instances have label replacements of vertices or edges either.

2.3 Representative Pattern Mining A few works investigate summarizing frequent subgraphs using a smaller number of representative patterns (i.e. a subset of all frequent subgraphs). ORIGAMI [9] and GraphRank [10] mine a set of frequent subgraph patterns first, and find the representative patterns by post-processing afterwards. To avoid post-processing, RING [29] first computes the pattern distribution by clustering and then mine representative patterns based on the pattern distribution. To improve efficiency, RP-Leap [22] attempts to derive a set of representative patterns which can roughly cover the entire frequent subgraphs.

Different from all above pattern summarization works, which generate representative patterns based on exact matching, REAFUM studies pattern mining on the basis of approximate matching essentially. To avoid returning all kinds of variations of each single pattern, REAFUM selectively picks one dominant variation of each pattern as the representative.

3 Notations and Problem Statement

Table 1: List of symbols

symbol	meaning
D	a set of graphs
g	an undirected labeled graph in D
V_g/E_g	the vertices/edges in g
Γ_V/Γ_E	the set of vertex/edge labels
ed	an edit operation
$ED(g_1, g_2)$	a sequence of ed transforming g_1 into g_2
$\lambda(g_i, g_j)$	the graph edit distance between g_i and g_j
s	a star structure
r	the root of a star
L	the leaves of a star
$Bi(s_i, s_j)$	the bipartite graph built based on s_i and s_j
$T(r_i, r_j)$	the edit distance between r_i and r_j
$F(L_i, L_j)$	the edit distance between L_i and L_j
$\lambda(s_i, s_j)$	the star edit distance between s_i and s_j
$S(g)$	the star representation of g
$\mu(g_i, g_j)$	the graph mapping distance between g_i and g_j
R_D	a list of representative graphs
m	the number of representative graphs
P	a set of frequent subgraphs
p	a frequent subgraph pattern
i_p	an instance of the pattern p
I_p	the set of pattern p's instances
sup_g	the support of g
σ	the support threshold
β	# of ed allowed when matching approximately

Table 1 lists the notations we use in this paper. We now give the problem statement. In this paper, we consider simple graphs which do not contain self-loops and multi-edges.

DEFINITION 3.1. *An **undirected labeled graph** g is a five element tuple $g = \{V, E, \Gamma_V, \Gamma_E, H_g\}$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, Γ_V is the set of vertex labels, Γ_E is the set of edge labels, and the labeling function H_g defines the mappings $V \to \Gamma_V$ and $E \to \Gamma_E$. We use the term **graph** to refer to undirected labeled graph, unless otherwise specified.*

Informally speaking, an **edit operation** ed on a graph g is an insertion or deletion of a vertex/edge or relabeling of a vertex. An edit operation is valid if the resulting graph remains connected.

DEFINITION 3.2. *The **graph edit distance** between g_1 and g_2, denoted as $\lambda(g_1, g_2)$, is the minimum number of edit operations that are needed to transform g_1 into g_2. Formally, $\lambda(g_1, g_2) = min_{ED:<ed_1,...,ed_k>\in\mathcal{ED}(g_1,g_2)}|ED|$, where $ED :< ed_1, ..., ed_k >$ is an edit operation sequence that transforms g_1 to g_2, and $\mathcal{ED}(g_1, g_2)$ denotes the set of all possible edit operation sequences transforming g_1 into g_2.*

DEFINITION 3.3. *A graph g is β **isomorphic** to another graph g' iff their graph edit distance $\lambda(g, g')$ is less than or equal to β.*

DEFINITION 3.4. *A graph g is β **subgraph isomorphic** to another graph $\tilde{g}$, denoted as $g \subseteq_\beta \tilde{g}$, iff there exists a subgraph g' of $\tilde{g}$ such that g is β isomorphic to g'.*

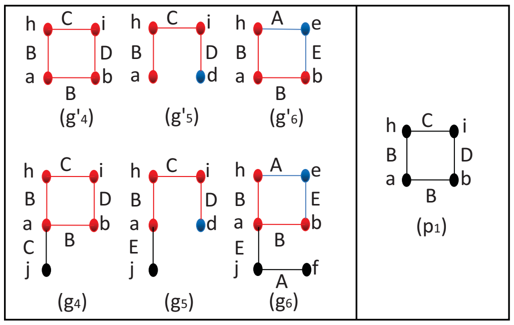


Figure 3: β isomorphism and β subgraph isomorphism

Figure 3 shows examples of β isomorphism and β subgraph isomorphism. g'_4, g'_5, and g'_6 are subgraphs of g_4, g_5, and g_6, respectively. Given a subgraph pattern p_1 on the right, the red vertices and edges in g'_4, g'_5, and g'_6 show the exact matches, while the blue vertices and edges in g'_4, g'_5, and g'_6 show the mismatches between p_1 and g'_4, g'_5, and g'_6, respectively. We can see that the graph edit distances $\lambda(p_1, g'_4) = 0$, $\lambda(p_1, g'_5) = 2$, and $\lambda(p_1, g'_6) = 3$. Therefore, p_1 is 3 isomorphic to all three graphs g'_4, g'_5, and g'_6. p_1 is 3 subgraph isomorphic to graphs g_4, g_5, and g_6.

DEFINITION 3.5. *Given a set of graphs D (referred as a graph database) and the allowed number of edit operations β, the **support** of a graph g_i, denoted as sup_{g_i}, is defined as the fraction of graphs in D to which g_i is β subgraph isomorphic.*

$$sup_{g_i} = \frac{|g \in D|g_i \subseteq_\beta g|}{|D|}$$

For example, in Figure 2, we have $p_1 \subseteq_3 g_4$, $p_1 \subseteq_3 g_5$, and $p_1 \subseteq_3 g_6$. Thus the support sup_{p_1} of p_1 is $\frac{|\{g_4,g_5,g_6\}|}{|D:\{g_1,g_2,g_3,g_4,g_5,g_6\}|} = 0.5$.

PROBLEM STATEMENT 1. *Given a set of graphs D, a support threshold ($0 < \sigma \leq 1$), and the allowed number of edit operations β, a subgraph g_i is a pattern if it satisfies the following two requirements: (1) It meets the support threshold σ (i.e. $sup_{g_i} \geq \sigma$.). (2) There exists at least one graph g, such that $p \subseteq g$ and $g \in D$.*

*The **approximate frequent subgraph mining** problem is to find all subgraphs in D, whose support is at least σ. We also refer to these subgraphs as frequent subgraphs.*

The first requirement ensures that the subgraph appears frequently within a certain number of approximations. The second requirement guarantees that only exhibited variations are considered for approximate matches. Considering the above two requirements, $sup_{p_1} \geq 0.5$ and $p_1 \subseteq g_4$ Thus, p_1 will be output as an approximate frequent pattern, and g'_4, g'_5, and g'_6 are the instances of p_1.

PROBLEM STATEMENT 2. *Given an approximate pattern p, the dominant form of p is defined as the instance of p that has the smallest average distance*[1] *to other instances in I_p, where I_p is the set of instances of p. This dominant form of p will be a representative approximate pattern.*

The **representative approximate pattern selection** problem is to find the dominant form of the approximate pattern.

[1]The distance will be defined in Section 4.1.

4 Proposed Solution

In this section, we present the proposed solution REAFUM to compute representative approximate patterns from a set of graphs. REAFUM has three components: representative graph selection, approximate pattern mining, and representative approximate pattern refinement. In the first component, a list of m diverse graphs is selected as the representative graphs. In the second component, a set of approximate patterns are mined incrementally based on the structure of each selected representative graph. In the last component, for each approximate pattern we mined, we compute the consensus form of its instances in the graph database as the representative pattern. The three components will be explained in Section 4.1, Section 4.2 and Section 4.3, respectively.

4.1 Representative Graph Selection In this section, we illustrate, given a set D of graphs, how to extract a list R_D of m diverse graphs.

There are $\binom{|D|}{m}$ ways of choosing m graphs from the pool of graphs D. Explicitly examining each of these options is intractable. Instead, we apply a greedy strategy and construct the list R_D incrementally.

Let R_D (which is initially empty) be the list of representative graphs our algorithm has picked so far. We want to add more graphs to R_D (one graph at a time) until the selection contains m graphs. At each step, our greedy heuristic picks the graph $g \in D - R_D$ that maximizes the score:

(4.1)
$$\Theta(D, R_D, g) = \sum_{g_i \in R_D} \frac{d(g, g_i)}{|R_D|} - \sum_{g_j \in D - R_D - \{g\}} \frac{d(g, g_j)}{|D - R_D - \{g\}|}$$

where $d(g_i, g_j)$ is a graph distance metric between g_i and g_j. The distance metric selection will be discussed afterwards. Algorithm 1 summaries the procedure.

Algorithm 1 Representative Graph Selection

```
input: $m, D$
ouput: a list of representative graphs $R_D =< g_1, g_2, \ldots, g_m >$
1: $\mathcal{X} \leftarrow D$; $R_D \leftarrow \emptyset$;
2: for $i \leftarrow 1$ To $m$ do
3:     $g \leftarrow \arg\max_{g \in \mathcal{X}} \Theta(D, R_D, g)$;
4:     $R_D \leftarrow R_D + < g >$;
5:     $\mathcal{X} \leftarrow \mathcal{X} - \{g\}$;
6: return $R_D$
```

Intuitively, before we pick the first representative graph, the selection R_D is empty. The first representative graph we select will be the one that has the smallest average graph distance to the other graphs in the database. This graph is likely to contain many subgraph patterns and these patterns are more likely to have many approximate matches in the database. The next representative graph to select will be the one that has small distances to the remaining graphs, yet is distinct from the selected representative graph(s). Therefore, the next representative graph tends to contain many new patterns.

To measure the label and structural differences $d(g_i, g_j)$ of graphs g_i and g_j, a number of graph distance metrics have been proposed [3, 8, 25]. Among these, graph edit distance and graph mapping distance are two widely used graph distance measures. However, the computation of graph edit distance is unaffordable since the problem of computing graph edit distance has been proved to be NP-hard in [18, 27]. Alternatively, the mapping distance, denoted as $\mu(g_i, g_j)$, not only has an acceptable time complexity $O(max(|V_{g_i}|, |V_{g_j}|)^3)$, but also can provide tight bounds of the graph edit distance [27].

In this paper, we adopt the graph mapping distance proposed in [27]. Unfortunately, the version in [27] does not support graphs with edge labels. We extend it to allow graphs with edge labels. In the following paragraphs, we will briefly explain the extended graph mapping distance. To compute the graph mapping distance $\mu(g_i, g_j)$ between g_i and g_j, we first represent each graph using a multiset of star structures, respectively. The star structure retains the label and structural information of the original graph.

DEFINITION 4.1. ***(Star Structure)*** *A star structure s is an attributed, single-level, rooted tree which can be represented by a 3-tuple $s = (r, L, l)$, where r is the root vertex, L is the set of leaves and l is a labeling function. Edges exist between r and any vertex in L and no edge exists between vertices in L.*

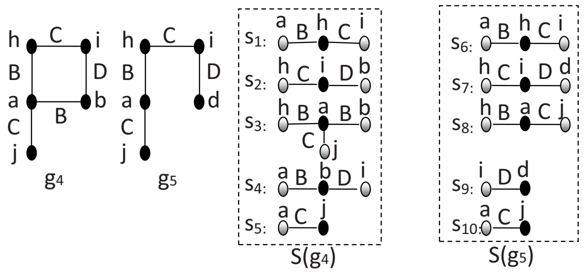


Figure 4: Star representation

In a star structure, the root vertex is the center and vertices in L can be considered as satellites surrounding the center. For any vertex v_i in a graph g, we can generate a corresponding star structure s_i in the following way: $s_i = (v_i, L_i, l)$ where $L_i = \{u | (u, v_i) \in E\}$. Accordingly, we can derive n star structures from a graph containing n vertices. Thus, a graph can be mapped to a multiset of star structures. This multiset is defined as the **star representation** of the graph g, denoted by $S(g)$. Figure 4 shows an example. $S(g_4)$ and $S(g_5)$ give the star structures of graphs g_4 and g_5, respectively.The star structures enable the edit distance between two star structures be computed easily in the following way.

DEFINITION 4.2. ***(Star Edit Distance)*** *Given two star structures s_1 and s_2, $\lambda(s_1, s_2) = T(r_1, r_2) + F(L_1, L_2)$. $T(r_1, r_2)$ gives the distance between the two roots, and $F(L_1, L_2)$ gives the distance between the two sets of leaves. Formally,*

$$T(r_1, r_2) = \begin{cases} 0 & \textit{if } l(r_1) = l(r_2), \\ 1 & \textit{otherwise.} \end{cases}$$

Algorithm 2 gives the details of the computation of $F(L_1, L_2)$[2].

For example, the star edit distance $\lambda(s_1, s_2)$ in Figure 5 can be computed in the following way. The two stars

[2]Due to the space limitation, we do not include the solution to the bipartite graph maximum matching problem [11]. The time complexity is O($|V| * |E|$), where $|V|/|E|$ is the number of vertices and edges of the bipartite graph, respectively.

Algorithm 2 Star Edit Distance

```
input: two stars $s_1$, $s_2$
ouput: $\lambda(s_1, s_2)$
 1: $\mathcal{X} \leftarrow D$; $R_D \leftarrow \emptyset$;
 2: // Exact matching removal
 3: for $v_i \in L_1$ do
 4:     for $v_j \in L_2$ do
 5:         if $l(v_i) = l(v_j)$ && $l(v_i, r_1) = l(v_j, r_2)$ then
 6:             remove $v_i$ from $L_1$;
 7:             remove $v_j$ from $L_2$;
 8:             break;
 9: // Bipartite graph construction
10: for $v_i \in L_1$ do
11:     for $v_j \in L_2$ do
12:         if $l(v_i) = l(v_j)$ || $l(v_i, r_1) = l(v_j, r_2)$ then
13:             $E_{Bi} \leftarrow$ e($v_i$,$v_j$);
14: $V_{Bi} = L_1 \cup L_2$;
15: // Finding maximum matching
16: $MM$= maximum matching of $Bi$;
17: $F(L_1, L_2) = MM + 2 * Max\{(|L_1| - MM), (|L_2| - MM)\}$;
18: $\lambda(s_1, s_2) = T(r_1, r_2) + F(L_1, L_2)$;
19: return $\lambda(s_1, s_2)$
```

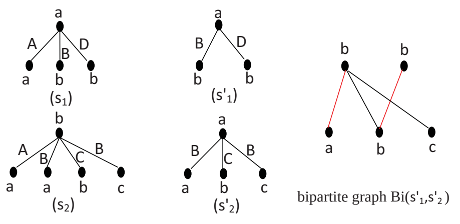


Figure 5: Example of computing star edit distance

s_1 and s_2 have different root labels, so $T(r_1, r_2) = 1$. After removing the exactly matched leaves from s_1 and s_2 simultaneously, we get s'_1 and s'_2. Then we start to construct a bipartite graph $Bi(s'_1, s'_2)$ based on the structures of s'_1 and s'_2. We first create five vertices (two from the leaves of s'_1 on the top and three from the leaves of s'_2 on the bottom). Knowing that all leaves of s'_1 have the same vertex label b as the label of the second leaf of s'_2, we create two edges connecting both vertices on the top to the second vertex on the bottom. We also find that the first vertex in s'_1 have the same edge label as the first and third vertices in s'_2, we create two more edges connecting the first vertex on the top to the first and third vertices on the bottom, respectively. After the construction of the bipartite graph, we calculate the maximum matching for it. In this case, one of the optimal maximum matching answers is shown in red lines, so the maximum matching for $Bi(s'_1, s'_2)$ is 2 (i.e. $MM = 2$.). $F(L_1, L_2) = 2 + 2 * max\{(2-2), (3-2)\} = 4$. Finally, the star edit distance $\lambda(s_1, s_2)$ is computed as $T(r_1, r_2) + F(L_1, L_2) = 1 + 4 = 5$. Subsequently, we will define the mapping distance between two graphs based on their star representations.

DEFINITION 4.3. ***(Mapping Distance)*** *Given two multisets of star structures S_1 and S_2 with the same cardinality, and assume that $B : S_1 \rightarrow S_2$ is a bijection. The mapping distance ρ between S_1 and S_2 is*
$\rho(S_1, S_2) = min_{\forall B} \sum_{s_i \in S_1} \lambda(s_i, B(s_i))$

The computation of $\rho(S_1, S_2)$ is equivalent to solving the assignment problem. The Hungarian algorithm [20] can obtain the optimal assignment solution in $O(n^3)$ time, where n is the cardinality of S_1 and S_2.

Now we formally give the definition of mapping distance between two graphs as on the distance of their star representations.

DEFINITION 4.4. ***(Graph Mapping Distance)*** *The mapping distance $\mu(g_1, g_2)$ between g_1 and g_2 is defined as:*

$$\mu(g_1, g_2) = \rho(S(g_1), S(g_2)) \tag{4.2}$$

The optimal mapping for computing the mapping distance between g_1 and g_2 is to approximate the mapping between the vertices of g_1 and g_2 in an optimal assignment.

Note that the two graphs may have different numbers of vertices. Assuming $|V_i| - |V_j| = k \geq 0$, $|S(g_i)| - |S(g_j)| = k$ must hold. In order to make g_i and g_j have the same number of vertices, we include k vertices with a special label ϵ into g_j.

Since we apply the graph mapping distance as the metric, by substituting Equation 4.2 into Equation 4.1, we get

$$\Theta(D, R_D, g) = \sum_{g_i \in R_D} \frac{\mu(g, g_i)}{|R_D|} - \sum_{g_j \in D - R_D - \{g\}} \frac{\mu(g, g_j)}{|D - R_D - \{g\}|} \tag{4.3}$$

Based on Equation 4.3, by incrementally adding graphs into the representative graph list R_D, we will be able to construct a list of diverse graphs which potentially contain most approximate patterns. In Section 4.2, we will explain how to enumerate patterns in each representative graph.

4.2 Pattern Mining

In this section, we first introduce the Apriori property in approximate pattern mining, and then describe the pattern enumeration strategy.

4.2.1 Apriori Property

The Apriori property claims that if a pattern meets the minimum support threshold, then any subgraph of it also meets the minimum support threshold. By our definition of frequent approximate patterns, we have the following theorem.

THEOREM 4.1. *For a frequent approximate graph g, any subgraph sg of g is also a frequent approximate pattern.*

Proof. For any subgraph sg of g, there is a sequence of edit operations transforming g to sg: $a_0 = g$, $a_1 = ed_1(a_0)$, $a_2 = ed_2(a_1)$, ..., $a_k = ed_k(a_{k-1})$, ..., $a_n = ed_n(a_{n-1}) = sg$, where $ed_k(a_{k-1})$ is an edit operation of deleting a vertex/edge from a_{k-1}. (1) By definition, when $k = 0$, $a_0 = g$ is a frequent approximate pattern[3]. (2) Suppose that a_k is a frequent approximate pattern, and $a_{k+1} = ed_{k+1}(a_k)$. We want to prove that a_{k+1} is frequent. For every instance i of a_k, if the vertex/edge (removed in ed_{k+1}) is embedded in i as it is in a_k, we define $i' = ed_{k+1}(i)$. Otherwise, $i' = i$. Based on this construction of i', i' is also β isomorphic to a_{k+1}. Thus, $s_{a_{k+1}} \geq s_{a_k} \geq \sigma$. Therefore, a_{k+1} is frequent. (3) By induction, we have $a_n = sg$ is a frequent approximate pattern.

Theorem 4.1 guarantees that if a graph g is not frequent, we do not need to examine any supergraphs of g. This theorem helps us prune the search space a lot.

4.2.2 Pattern Enumeration

In this section, we present the approximate pattern enumeration strategy.

[3]We define $a_0 = ed_0(a_{-1}) = g$.

Algorithm 3 Approximate Pattern Mining

```
input: $R_D, D, \sigma, \beta$
ouput: a set of patterns $P = \{p_1, p_2, \ldots, p_i\}$
 1: $P \leftarrow \emptyset$
 2: while $R_D$ is not empty do
 3:     $g \leftarrow$ the first graph from the list $R_D$
 4:     $P^g \leftarrow$ Pattern Mining ($g, P, D, \sigma, \beta$);
 5:     $P \leftarrow P \cup P^g$;
 6:     $R_D \leftarrow R_D - < g >$;
 7: return $P$
```

After constructing the list R_D of representative graphs, we mine a set P of approximate patterns incrementally. The graphs in R_D are ranked according to the order in which they are selected into R_D. Each time we retrieve the first representative graph g from the list R_D, and compute a set P^g of approximate patterns based on g. Then, P^g is inserted into P and g is removed from the representative list. We repeat this process until all the representative graphs are enumerated in R_D. Algorithm 3 summarizes the procedure.

Algorithm 4 Pattern Mining

```
input: $g, D, \sigma, \beta$
ouput: a set of patterns $P^g = \{p_1, p_2, \ldots, p_i\}$
 1: $P^g \leftarrow \emptyset$;
 2: for $i \leftarrow 1$ To $|V_g|$ do
 3:     $sup_p \leftarrow$ Counting support ($p, D, \sigma, \beta, I_p, v_i$)[4];
 4:     if $sup_p \geq \sigma$ then
 5:         if $p$ is not an instance before then
 6:             $P^g \leftarrow P^g \cup \{p\}$;
 7:         $P^p \leftarrow$ Mining bigger pattern($p, g, D, \sigma, \beta$);
 8:         $P^g \leftarrow P^g \cup P^p$;
 9:     remove $v_i$ from consideration;
10: return $P^g$
```

Algorithm 5 Mining Bigger Pattern

```
input: $p, g, D, \sigma, \beta$
ouput: a set of patterns $P^p = \{p_1, p_2, \ldots, p_i\}$
 1: $P^p \leftarrow \emptyset$;
 2: $N \leftarrow$ find-neighbors($p, g$);
 3: if $N$ is not empty then
 4:     for $j \leftarrow 1$ To $|N|$ do
 5:         $p' \leftarrow p \oplus n_j$; $//n_j \in N$
 6:         $sup_{p'} \leftarrow$ Counting support ($p', D, \sigma, \beta, I_p, n_j$);
 7:         if $sup_{p'} \geq \sigma$ then
 8:             if $p'$ is not an instance before then
 9:                 $P^p \leftarrow P^p \cup \{p'\}$;
10:             $P^{p'} \leftarrow$ Mining bigger pattern($p', D, \sigma, \beta$);
11:             $P^p \leftarrow P^p \cup P^{p'}$;
12: return $P^p$;
```

In the following, we explain how to mine approximate patterns for each representative graph. Given a representative graph g and two parameters σ and β, patterns are enumerated in the depth-first fashion. Algorithms 4 and 5 summarize the procedure. We first find frequent approximate vertices in g. Starting from these frequent vertices as the first set of patterns, we recursively grow vertices and edges to explore bigger patterns. If all possible candidate patterns grown from pattern p have been checked, we backtrack to the pattern from which p is generated, and grow that pattern by adding another vertex. After finding all patterns containing v_i, we will ignore v_i when examining the remaining patterns.

In Algorithms 4 and 5, if the support sup_p of the pattern candidate p meets the threshold σ, we further check whether p is an instance of other patterns mined in previous representative graphs or not. If not, p becomes a new pattern. Otherwise, p is just a variation of other patterns we mined before. As a redundant pattern, p will not be added to the pattern collection.

The function find-neighbors(p, g) in Algorithm 5 returns a set N of remaining vertices which are directly connected to the pattern p in g. For each vertex n_j in the neighbor set, we add it to the original pattern p to form a supergraph p' by a pattern growing operation $\oplus$. This supergraph p' becomes a pattern candidate. We then check the support $sup_{p'}$ of p'. If $sup_{p'} \geq \sigma$, p' will be used to grow patterns. Otherwise, we stop checking any supergraphs of p', since based on the Apriori property presented in Section 4.2.1 we know that the support of any supergraph of p' won't meet the support threshold either.

A pattern growing operation $\oplus$ is introduced in Algorithm 5. By $p' \leftarrow p \oplus n_j$, we not only add the vertex n_j to p but also include all the edges that connect n_j and any vertex in p.

Algorithm 6 Counting support

```
input: $p', D, \sigma, \beta, I_p, n_j$
ouput: the support $s$ of the pattern candidate $p'$
 1: $s \leftarrow 0$;
 2: for each $g_i \in D$ do
 3:     for each $i_p$ in $g_i$ do
 4:         //$n_j$ is the vertex added when extending $p$ to $p'$
 5:         for each $v$ in $S(n_j)$ do
 6:             //$S(n_j)$ is the set of vertices to check
 7:             $\hat{i}_{p'} \leftarrow i_p \oplus^i v$;
 8:             if the edit distance $\lambda(p', \hat{i}_{p'}) \leq \beta$ then
 9:                 if $\hat{i}_{p'}$ is the first instance of $p'$ in $g_i$ then
10:                     $s \leftarrow s + 1$;
11:                 $I_{p'} \leftarrow I_{p'} \cup \{\hat{i}_{p'}\}$;
12: return $s$;
```

To count the support of a pattern candidate p', we check all the instances I_p of the pattern p from which p' is generated. Only graphs that contain an instance i_p of the pattern p may further contain an approximate match for p'.

In Algorithm 6, an instance growing operation $\oplus^i$ is introduced. Given the newly added vertex n_j in g, a mapping vertex $v_{mapping(n_j)}$ is first located in g_i. The job of locating $v_{mapping(n_j)}$ has been done when we compute the graph mapping distance between g and g_i in the representative graph selection component. So it does not incur any additional computation. After locating the mapping vertex $v_{mapping(n_j)}$ and adding it to one instance i_p, the edge between $v_{mapping(n_j)}$ and $v_{mapping(v)}$ is added into i_p if there is a corresponding edge between n_j and v in p', where v is a vertex in p' and $v_{mapping(v)}$ is the mapping vertex of v in g_i. In this way, a potential instance $\hat{i}_{p'}$ of pattern candidate p' is constructed. Since we know the vertex mapping between the pattern candidate p' and its potential instance $\hat{i}_{p'}$, their graph edit distance $\lambda(p', \hat{i}_{p'})$ can be easily computed by scanning each graph once. The time complexity is $O(|V_{p'}| + |E_{p'}|)$.

Note that when mapping vertices between two graphs g_i and g_j, the Hungarian algorithm only returns one optimal bijection between V_{g_i} and V_{g_j}. This may lead to two potential issues in the pattern mining process. (1) For a graph g_i, it is possible that there are several stars which are identical in their star representations. For example, let

[4]Since v_i here is the first vertex to be added into the pattern candidate, p is an empty graph, and I_p is also empty instance set.

$s(v_m) = s(v_n)$. Based on the bijection, the vertex v_m in g_i will have only one mapping vertex $v_{mapping(v_m)}$ in g_j. This is not desirable since we can exchange the mapping vertices of v_m and v_n and get another optimal bijection. (2) To achieve the optimal alignment, two vertices which are directly connected in g_i may be mapped to two vertices which are far away from each other in g_j. To address the above two problems, for each vertex v_m in g_i, instead of deriving only one mapping vertex, we will find a set $S(v_m)$ of similar vertices in g_j according to their star structures. ($v_{mapping(v_m)}$ by Hungarian algorithm is also included in $S(v_m)$.) The vertices in $S(v_m)$ become the potential vertices to check when extending the instances.

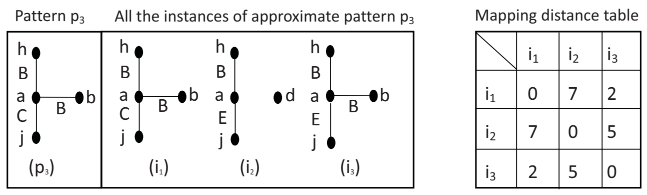
4.3 Consensus refinement

In this section, we explain how we do the consensus refinement to achieve the dominant form of the approximate patterns mined in Section 4.2.

Given each pattern p and all its instances I_p, the dominant form of p is defined as the instance of p that has the smallest average mapping distance to other instances in I_p. This dominant form of p will be a representative approximate pattern. Algorithm 7 summarizes the procedure.

Algorithm 7 Representative pattern consensus refinement

```
input: a set of patterns $P$
ouput: a set of refined representative patterns $P_{refine}$
1: $P_{refine} \leftarrow \emptyset$;
2: for $p$ in $P$ do
3:     $i \leftarrow \frac{1}{|I_p|-1} min_{i \in I_p} \sum_{i_p \in I_p, i \neq i_p} \mu(i_p, i)$;
4:     $P_{refine} \leftarrow P_{refine} \cup \{i\}$;
5: return $P_{refine}$
```



	i_1	i_2	i_3
i_1	0	7	2
i_2	7	0	5
i_3	2	5	0

Figure 6: Consensus refinement example

Consider the approximate pattern p_3 shown in the pattern mining component of Figure 2 as an example. Figure 6 shows all of its instances and the graph mapping distances between each pair of instances. From the graph mapping distance table, we know that i_3 achieves the smallest average graph mapping distances among all three instances. Therefore, the dominant form of p_3 is i_3, and p_3 is replaced by the representative pattern i_3 as shown in the consensus refinement component of Figure 2.

5 Experiments

In this section we present the experimental results for evaluating our proposed method. We first briefly describe the datasets used in Section 5.1. In Section 5.2, we give a comprehensive analysis of REAFUM in terms of the effectiveness, sensitivity and scalability using synthetic datasets. Then in Section 5.3 we compare the result of REAFUM with that of an exact subgraph mining algorithm, Gaston, on a chemical compound dataset. REAFUM is written in Java and Gaston was downloaded from the Gaston project website. All experiments were done on a Core^{tm} i7-3770 CPU@3.40GHz PC.

5.1 Data Sets

In this section, we describe the synthetic dataset and chemical compound dataset used in the experiments.

Synthetic Dataset. We built a data generator that takes six parameters as inputs to control the graph database:

1. the number of graphs in the database , denoted as $|D|$.
2. the average size (i.e. the number of vertices) of the database graph, denoted as $\overline{|g|}$.
3. the number of patterns embedded in the database, denoted as $|P_{refine}|$.
4. the average size (i.e. the number of vertices) of the pattern, denoted as ($\overline{|p|}$).
5. the support for the pattern, denoted as sup.
6. the number of edit operations allowed when matching approximately, denoted as β.

Table 2: Default parameter setting

Settings	$\|D\|$	$\overline{\|g\|}$	$\|P_{refine}\|$	$\overline{\|p\|}$	sup	β	$\|R_D\|$
setting 1	300	100	30	10	0.1	3	30

Based on the above input parameters, patterns are randomly generated and embedded in the graph database. The default parameter setting is shown in Table 2. Under this default setting, the database contains 300 graphs; the size of the database graph is 100 on average; 30 patterns are embedded approximately 30 (0.1×300) times each; the pattern and its approximate matches may have up to 3 edit operations; and we select 30 graphs as representative graphs. In our experiments, when we vary one parameter, the remaining parameters are set at their default values. All the results reported below are average of 5 runs.

Chemical Compound Dataset. The chemical compound dataset[5] is widely used in pattern mining algorithms [14, 21, 23, 26]. The dataset contains 340 chemical compounds, 24 different atoms, 66 atom types, and 4 types of bounds. The 340 chemical compounds form a graph database of 340 graphs. Atoms are modeled by vertices and the type of atoms forms the vertex labels. Vertices are then connected by edges to model the bound relationship. The graph database contains 27 vertices and 27 edges per graph on average. The largest one contains 214 edges and 214 vertices. The goal is to find the approximate frequent chemical compound substructures.

5.2 Performance on Synthetic Dataset

In this section, we investigate the effectiveness, sensitivity, and scalability performances of REAFUM based on the synthetic dataset. Since there is no golden standard in approximate pattern mining, the patterns embedded in the synthetic dataset will be used as the ground truth.

Effectiveness : We compare the the representative approximate patterns mined by REAFUM with the embedded patterns with the default parameter setting. We run the default setting. For each mined pattern, we find its most similar seed pattern. Then their graph edit distance is computed. The average graph edit distance is 0.3. Given that the average pattern size is 10, this small edit distance is expected

[5]The dataset is available at http://www.liacs.nl/ snijssen/gaston/

because each instance we embedded in the graphs may have edit distance of up to 3 from the seed pattern. In our experiments, 90% of the time the consensus pattern is identical to the seed pattern, and 10% of the time the consensus pattern is a variation of the seed pattern.

Sensitivity : To evaluate REAFUM's sensitivity to the edit operations allowed, β is set to 5 different values (β = 1, 2, 3, 4, 5). Figure 7 shows that REAFUM finds all thirty embedded patterns, while Gaston fails to find any of them. This demonstrates that REAFUM is robust to the change in parameter β and pattern mining algorithms based on exact matching can not handle approximate patterns.

We also test REAFUM's sensitivity to the number of representative graphs selected in R_D. $|R_D|$ is set to different values (R_D = 1, 2, 3, 4, 5, 6, 10, 20, 30). Figure 8 shows that when we only select three representative graphs, we find 21 of the 30 embedded patterns. All embedded patterns are discovered when we have 5 or more representative graphs. This shows that the greedy algorithm applied in the representative graph selection component is able to find a small number of diverse graphs as representative graphs which well cover the space of frequent subgraphs.

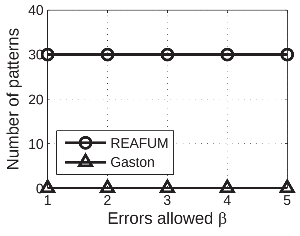


Figure 7: Sensitivity to β on synthetic data

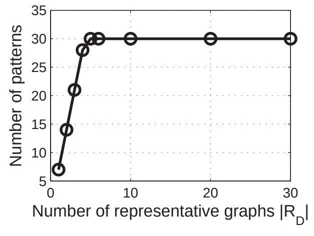


Figure 8: Sensitivity to $|R_D|$ on synthetic data

Scalability : In this part, we will test the REAFUM's scalability to the number of graphs in the database, the size of the database graphs, the size of embedded patterns, the number of embedded patterns, sequentially.

- To test the scalability performance to the number of graphs in the database, $|D|$ is set to different values $|D|$ = 150, 200, 250, 300, 350.
- To test the scalability performance to the size of the database graphs, $\overline{|g|}$ is set to different values $\overline{|g|}$ = 50, 75, 100, 125, 150.
- To test the scalability performance to the size of embedded patterns, $\overline{|p|}$ is set to different values $\overline{|p|}$ = 5, 10, 15, 20, 25.
- To test the scalability performance to the number of embedded patterns, $|P_{refine}|$ is set to different values $|P_{refine}|$ = 10, 20, 30, 40.

We divide the running time of REAFUM into two parts, representative selection time and patterning mining time. Representative selection time is the time needed to select the representative graphs, and patterning mining time is the time needed to find patterns from the those selected representative graphs. Figures 9, 10, 11, 12 show the running time as a function of varying one parameter. We can observe that the execution time of pattern mining scales linearly with respect to the number of graphs, graph size, number of embedded patterns, and pattern size. However, the representative selection is still improvable, because the running time can be

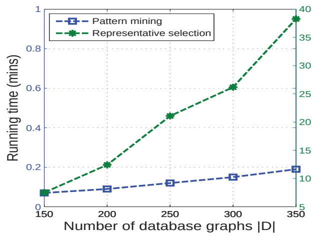


Figure 9: Scalability to database size on synthetic data

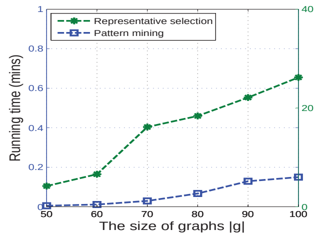


Figure 10: Scalability to graph size on synthetic data

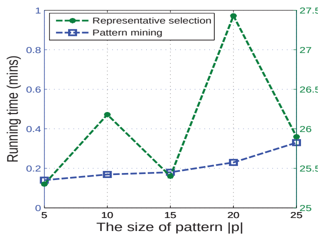


Figure 11: Scalability to pattern size on synthetic data

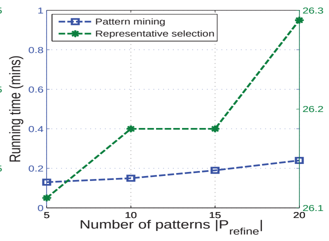


Figure 12: Scalability to number of patterns on synthetic data

dramatically reduced, since both the pairwise graph mapping distance and the pairwise star mapping distance can be calculated in parallel. We leave this for feature work.

5.3 Performance on Chemical Dataset

In this section we evaluate the performance of REAFUM on the real dataset. We compare REAFUM with Gaston, which is a pattern mining algorithm based on exact matching. We examine the number of patterns and the size of patterns mined by REAFUM and Gaston, respectively.

The support threshold σ ranges from 30% to 70%. For REAFUM, the number of edit operations allowed for approximate match, β, is set to 3, and the number of representative graphs is set to 34 (10%$|D|$). Only patterns of size 4 or larger are included in the comparisons.

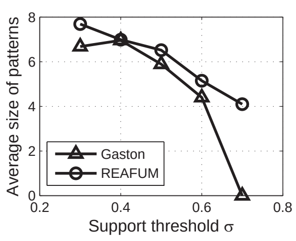


Figure 13: Average size of patterns on chemical data

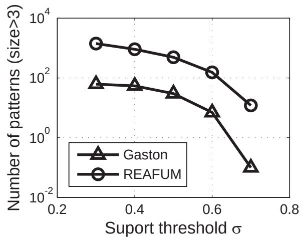


Figure 14: Number of patterns on chemical data

Figure 13 shows that the patterns reported by REAFUM are larger than that by Gaston on average. Figure 14 suggests that REAFUM is able to find more patterns than Gaston with the same support threshold. When $\sigma = 70\%$, there is no frequent pattern based on exact matching. However, REAFUM can still find 12 approximate patterns. This is because each approximate pattern discovered by REAFUM may summarize similar patterns found by Gaston or may represent the consensus of patterns that are small variations of each other but missed by Gaston.

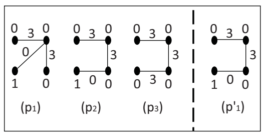


Figure 15: Pattern summarization example

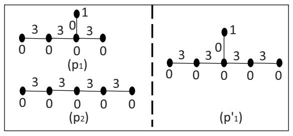


Figure 16: Pattern rescued example

Figure 15 shows an example that a REAFUM pattern may summarize several similar patterns found frequent by exact matching algorithms. When $\sigma = 60\%$, Gaston reports three similar patterns p_1, p_2, and p_3, while REAFUM only reports p'_1 and regards p_1 and p_3 as p'_1's instances. Figure 16 shows an example that REAFUM may rescue pattern variations found infrequent by exact matching algorithms. When $\sigma = 60\%$, Gaston finds two similar patterns p_1 and p_2, and all of their supergraphs are infrequent. REAFUM identifies a larger subgraph pattern p'_1 as an approximate pattern. From these observations, we expect that REAFUM discovers more and larger patterns than Gaston, which is demonstrated in Figure 14 and Figure 13, respectively.

6 Conclusion

In this paper we investigate the problem of representative approximate pattern mining. We discuss the limitations of existing pattern mining methods based on exact matching and propose an approximate subgraph mining algorithm REAFUM. REAFUM constructs a list of representative graphs, from which frequent representative subgraphs are enumerated allowing approximate matches, and consensus patterns are derived. Through a comprehensive experiment, we demonstrate the superior performance of REAFUM.

7 Acknowledgement

This work was partially supported by NSF IIS-1162369. We would like to thank the anonymous reviewers for their insightful comments.

References

[1] Niusvel Acosta-Mendoza, Andrés Gago Alonso, and José E. Medina-Pagola. Frequent approximate subgraphs as features for graph-based image classification. *Knowl.-Based Syst.*, 27:381–392, 2012.

[2] Christian Borgelt and Michael R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *ICDM*, pages 51–58, 2002.

[3] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998.

[4] Jason Cong, Hui Huang, and Wei Jiang. A generalized control-flow-aware pattern recognition algorithm for behavioral synthesis. In *DATE*, pages 1255–1260, 2010.

[5] Jason Cong and Wei Jiang. Pattern-based behavior synthesis for fpga resource reduction. In *FPGA*, pages 107–116, 2008.

[6] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971.

[7] Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. Knowl. Data Eng.*, 17(8):1036–1050, 2005.

[8] Mirtha-Lina Fernández and Gabriel Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6/7):753–758, 2001.

[9] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, Jérémy Besson, and Mohammed Javeed Zaki. ORIGAMI: mining representative orthogonal graph patterns. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*, pages 153–162, 2007.

[10] Huahai He and Ambuj K. Singh. Graphrank: Statistical modeling and mining of significant subgraphs in the feature space. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, pages 885–890, 2006.

[11] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[12] M. Shahriar Hossain and Rafal A. Angryk. Gdclust: A graph-based document clustering technique. In *ICDM Workshops*, pages 417–422, 2007.

[13] Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. In *ISMB (Supplement of Bioinformatics)*, pages 213–221, 2005.

[14] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*, pages 549–552, 2003.

[15] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23, 2000.

[16] Yi Jia, Jintao Zhang, and Jun Huan. An efficient graph-mining method for complicated and noisy data with real-world applications. *Knowl. Inf. Syst.*, 28(2):423–447, 2011.

[17] Chuntao Jiang, Frans Coenen, and Michele Zito. Finding frequent subgraphs in longitudinal social network data using a weighted graph mining approach. In *ADMA (1)*, pages 405–416, 2010.

[18] Derek Justice and Alfred O. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1200–1214, 2006.

[19] Mehmet Koyutürk, Ananth Grama, and Wojciech Szpankowski. An efficient algorithm for detecting frequent subgraphs in biological networks. In *ISMB/ECCB (Supplement of Bioinformatics)*, pages 200–207, 2004.

[20] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[21] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.

[22] Jianzhong Li, Yong Liu, and Hong Gao. Efficient algorithms for summarizing graph patterns. *IEEE Trans. Knowl. Data Eng.*, 23(9):1388–1405, 2011.

[23] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *KDD*, pages 647–652, 2004.

[24] Luc De Raedt and Stefan Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In *IJCAI*, pages 853–862, 2001.

[25] John W. Raymond, Eleanor J. Gardiner, and Peter Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *Comput. J.*, 45(6):631–644, 2002.

[26] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

[27] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

[28] Shijie Zhang and Jiong Yang. Ram: Randomized approximate graph mining. In *SSDBM*, pages 187–203, 2008.

[29] Shijie Zhang, Jiong Yang, and Shirong Li. RING: an integrated method for frequent representative subgraph mining. In *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, pages 1082–1087, 2009.