On Effective Presentation of Graph Patterns: A Structural Representative Approach[*]

Chen Chen[1] Cindy Xide Lin[1] Xifeng Yan[2] Jiawei Han[1]

[1]University of Illinois at Urbana-Champaign
{cchen37, xidelin2, hanj}@cs.uiuc.edu
[2]IBM T. J. Watson Research Center
xifengyan@us.ibm.com

ABSTRACT

In the past, quite a few fast algorithms have been developed to mine frequent patterns over graph data, with the large spectrum covering many variants of the problem. However, the real bottleneck for knowledge discovery on graphs is neither efficiency nor scalability, but the *usability* of patterns that are mined out. Currently, what the state-of-art techniques give is a lengthy list of exact patterns, which are undesirable in the following two aspects: (1) on the micro side, due to various inherent noises or data diversity, exact patterns are usually not too useful in many real applications; and (2) on the macro side, the rigid structural requirement being posed often generates an excessive amount of patterns that are only slightly different from each other, which easily overwhelm the users.

In this paper, we study the presentation problem of graph patterns, where *structural representatives* are deemed as the key mechanism to make the whole strategy effective. As a solution to fill the usability gap, we adopt a two-step *smoothing-clustering* framework, with the first step adding error tolerance to individual patterns (the micro side), and the second step reducing output cardinality by collapsing multiple structurally similar patterns into one representative (the macro side). This novel, integrative approach is never tried in previous studies, which essentially rolls-up our attention to a more appropriate level that no longer looks into every minute detail. The above framework is general, which may apply under various settings and incorporate a lot of extensions. Empirical studies indicate that a compact group of informative delegates can be achieved on real datasets and the proposed algorithms are both efficient and scalable.

*The work was supported in part by the U.S. National Science Foundation grants IIS-05-13678 and BDI-05-15813 and Office of Naval Research grant N00014-08-1-0565. Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications – Data Mining

General Terms: Algorithms

Keywords: frequent graph pattern, structural representative, smoothing-clustering

1. INTRODUCTION

Frequent graph pattern (subgraph) mining is an important data mining task, due to the prevalence of graph data in real applications (e.g., chemical compounds, XML and social networks), and also because graph is the most expressive data structure for modeling purposes. Mining substructures that frequently occur (for at least *min_sup* times) can help people get insight into the graphs, which has been used for indexing [23], classification [5] and many other applications. Recently, quite a few fast algorithms are developed to mine frequent patterns over graph data, with the large spectrum covering many variants of the problem. However, the real bottleneck for knowledge discovery on graphs is neither efficiency nor scalability, but the *usability* of patterns that are mined out, which hampers the deployment of effective individual and global analysis.

There are two sources of issues that lead to the above usability gap. First, on the micro side, due to various inherent noises or data diversity (e.g., data collection errors, insertions, deletions and mutations as the data evolves), exact patterns are usually not too useful in many real applications. Indeed, it is often crucial to allow imprecise matchings so that all potentially interesting patterns can be captured. For example, in biological networks, due to some underlying process, occasionally we may observe two subnetworks N_1 and N_2, which are quite similar in the sense that, after proper correspondence, discernable resemblance can be observed between respective vertices, e.g., with regard to their molecular compositions, functions, etc., and the interactions within N_1 and N_2 are nearly identical to each other. Exact patterns certainly cannot fit into such scenarios.

Second, on the macro side, there are usually too many number of patterns. Compared to less complicated data structures, the curse of combinations is even worse for graphs. Given a graph with n nodes, it has $O(n^2)$ possible edges, where a difference of just one edge can make two patterns not exactly matchable. Having such a large set of patterns that are only slightly different from each other, the mining result becomes lengthy but redundant, which easily overwhelms the users.

If we take a closer look at the above two difficulties: the limited expressiveness of individual patterns in a micro sense and the huge volume of mining results in a macro sense, they are actually both related to the same underlying problem, that is, the level on which we perform analysis is too low. The exact scheme treats a graph mining task with all structural details, which not only limits the amount of information any single pattern can cover, but also makes the output cardinality too big to be handled.

Thus, the solution we propose for this problem is to roll-up our attention to a more appropriate level through *structural representatives*, where two issues are simultaneously tackled. Following above discussions, it can be achieved by a so-called *smoothing-clustering* framework, i.e., we may consider working in both the micro direction and the macro direction. Note that, this is in vivid contrast to some previous studies that only solve one side of the problem, but not both (cf. related work in Section 2).

In the micro direction, based on graph structures, a given pattern is usually very similar to quite a few look-alikes (e.g., they might be identical except δ edges). For many real applications, it is often unnecessary and sometimes even inappropriate to assume the restrictive exact setting: This leads us to the introduction of an error tolerance δ, which can help blur the rigid boundaries among individual patterns. By such a *smoothing* procedure, patterns within the δ threshold are now treated as being equivalent, whose supports can be further aggregated to reflect the approximation thus achieved.

In the macro direction, we want to show as few patterns as possible so that the users' reviewing efforts are minimized. As we blur the exact boundary in smoothing, it now becomes very likely that structurally similar patterns will also have substantial overlaps with regard to their *smoothed* supports, an ideal property for cardinality reduction purposes. Now, instead of displaying all patterns, we perform another *clustering* step, so that the cluster centers are taken as *structural representatives* for the others, while the support information is also well-preserved.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Then, for ease of presentation, we start from a relatively simple and easy-to-understand setting, where frequent induced subgraphs are mined in the transaction database setting, to carefully study the smoothing-clustering framework, with Section 3 focusing on the smoothing step and Section 4 focusing on the clustering step. After laying out major algorithms, we further generalize our proposed methods to the non-induced case, the single graph case and many other extensions in Section 5. Experimental results are reported in Section 6, and conclusions are given in Section 7.

2. RELATED WORK

There is a wealth of literature devoted to mining frequent patterns over graph data, such as [11, 3, 16, 12]. Compared to them, our major aim in this study is to enhance the usability of patterns that are mined out. Previously, there have been researches that either relax the rigid structural requirement or reduce output cardinality by exact representatives (which are often associated with terminologies such as mining approximate patterns or pattern compression/summarization); however, as far as we know, ours is the first that considers the possibility of integrating both aspects and conquering them at once.

Following similar concepts in frequent itemset mining [2, 17], the data mining community has proposed algorithms to mine maximal [9] and closed [22] subgraphs from the database. The notion of closeness is proposed to deliver all information about the frequent patterns in a lossless way, where both structure and support are fully preserved. In contrast, motivated by Apriori, the maximal approach only focuses on those biggest ones, because all other frequent patterns must be contained by them. Apart from these two extremes, [20, 13] suggest a road in the middle, which differs from the maximal scheme in that it incorporates support consideration, and also differs from the closed scheme because sub-patterns and super-patterns are not required to have exactly the same support so that they can be collapsed. As we can see, there is one thing in common for all these methods, i.e., the graph structures are taken as a bottom line, which does not allow any relaxation. In this way, what we get is just a compressed output, whose patterns are still exact in the sense that no structural difference is tolerated.

On the other hand, especially in the area of Bioinformatics where noises are inevitable in the analysis of biological networks, algorithms [18, 4] have been proposed that are able to find those patterns in data whose appearances can be slightly different for each occurrence. They usually bear the name of mining approximate patterns, but "the stone only kills one bird", because the result cardinality is not reduced: Users are still viewing a lengthy but redundant list, except that each pattern is now supported by many imprecise matchings in the data.

There are other related studies that try to pick some (say top-k) delegates from the mined patterns [6, 14, 21, 19]. However, either the selection criterion being applied is not what we want [6, 14], e.g., [6] delivers the top-k frequent patterns, or the scheme is specifically designed for simple settings [21, 19], e.g., [21] assumes an independent "bag-of-words" model and summarizes patterns in the form of itemset profiles, which is hard, if not impossible, to be generalized to the graph scenario, without incurring a lot of complicated dependence modeling. Recently, [7] designs a randomized algorithm to mine maximal frequent subgraphs, which are then compressed to a few orthogonal representatives based on pattern structures. However, (1) it does not consider support information like all maximal schemes, and (2) the mining procedure still disables structural relaxation (i.e., no smoothing is done, in the terminology of this paper), a very important aspect as we observe.

3. THE SMOOTHING STEP

In this paper, we will use the following notations: For a graph g, $V(g)$ is its vertex set, $E(g) \subseteq V(g) \times V(g)$ is its edge set, L is a function mapping a vertex to a label, and $I : V(g) \times V(g) \rightarrow \{0, 1\}$ is an indicator such that $I(v_1, v_2) = 1$ if $(v_1, v_2) \in E(g)$, and $I(v_1, v_2) = 0$ otherwise. For now, there are no edge labels, i.e., an edge is just a plain connection; further extensions will be included in Section 5.

How to specify structural matchings between graphs is a core concept for knowledge discovery on graphs. Given a pattern graph p and a data graph d, the central theme of graph mining is to find out whether there is a subgraph of d that can be matched with p, and if the matching exists, where does it happen. As we discussed in above, previous methodologies usually make the assumption that an exact match must be established between p and d so that it can be identified, which is inevitably restricted in many real applications. Now, by Definition 1, we can relax this rigid constraint up to a δ-threshold so that as long as there are no more than δ interconnections among vertices (i.e., one has the edge while the other does not) that differ between p and d, the small error is neglected.

Definition 1. *(Approximate Graph Isomorphism). For two graphs g' and g, we say that g' and g are* approximately isomorphic *if there exists a* bijective *mapping $f : V(g') \leftrightarrow V(g)$, s.t.,: first, $\forall v' \in V(g'), L'(v') = L(f(v'))$; and second, the following symmetric difference:*

$$\sum_{v'_1, v'_2 \in V(g')} \left| I'(v'_1, v'_2) - I(f(v'_1), f(v'_2)) \right|$$

is less than or equal to δ. Here, L'/I' and L/I are the label/indicator functions of g' and g, respectively, while δ is a predetermined error tolerance *parameter.*

There are three things we want to emphasize. First, g' and g must have the same number of vertices (through bijective mapping), because we are defining approximate *graph* isomorphism here. When it comes to approximate *subgraph* isomorphism, which is defined in Section 4, the vertex set cardinalities can be different.

Second, the vertex labels of g' and g are exactly identical under the correspondence f (first condition), i.e., labels are characteristic values, which cannot be substituted. We will discuss further extensions in Section 5, which tries to remove this constraint.

Third, the symmetric difference of corresponding indicator functions is bounded by δ (second condition). Actually, for two graphs g' and g, there might be multiple bijective mappings that are also label-preserving. Let these mappings comprise a set $\mathcal{F}(g', g)$, then we can define the *structural difference* between g' and g to be:

$$\min_{f \in \mathcal{F}(g', g)} \left\{ \sum_{v'_1, v'_2 \in V(g')} \left| I'(v'_1, v'_2) - I(f(v'_1), f(v'_2)) \right| \right\},$$

which is denoted as *diff*(g', g), with a special case of

$$\textit{diff}(g', g) = +\infty \text{ for } \mathcal{F}(g', g) = \varnothing,$$

i.e., when the vertex labels of g' and g cannot be matched. Being a useful reminder, it is easy to verify that the approximate graph isomorphism concept we just gave is equivalent to *diff*$(g', g) \leq \delta$.

Definition 2. *(Smoothed Support). Given a set of n frequent patterns that are mined from the graph database $\mathcal{D}$: $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$, p_i's* original supporting transaction set *$\hat{\mathcal{D}}_{p_i}$ includes every $d \in \mathcal{D}$ such that p_i is an* induced subgraph *of d. Due to the δ error tolerance that was introduced in above, the* smoothed supporting transaction set *of p_i is defined as:*

$$\mathcal{D}_{p_i} = \bigcup_{p_j \in \mathcal{P}, |V(p_i)| = |V(p_j)|, \textit{diff}(p_i, p_j) \leq \delta} \hat{\mathcal{D}}_{p_j}.$$

Here, $|\hat{\mathcal{D}}_p|$ is called the original support *of p, and is denoted as $\widehat{sup}(p)$ (of course, we have $\widehat{sup}(p_i) \geq min\_sup$, since p_i is frequent); while $|\mathcal{D}_p|$ is called the* smoothed support *of p, and is denoted as $sup(p)$.*

The core concept of smoothing is to blur the exact boundaries among individual patterns, which is done by aggregating respective supporting transactions. Based on the pattern set $\mathcal{P}$ got from graph mining algorithms, we identify all patterns that are structurally similar to p_i, i.e., all p_j's such that *diff*(p_i, p_j) is less than or equal to δ, and combine corresponding $\hat{\mathcal{D}}_{p_j}$'s into $\mathcal{D}_{p_i}$. By doing this, the support of each frequent pattern is further enhanced by those of its approximately isomorphic counterparts.

Based on the idea of smoothing, Definition 2 augments the support concept of traditional graph mining technologies, which is no longer subject to precise matches. However, as careful readers might already notice, we did emphasize the phrase "induced graph" in above descriptions, which is due to the reason that: Throughout Definition 1, when calculating the structural difference of g' and g, all edges among a given set of vertices are taken into consideration – Such semantics is directly related to the *induced* notion of graph theory, which we formally give in below.

Definition 3. *(Induced Subgraph). For two graphs g' and g, g' is said to be an* induced subgraph *of g if there exists an* injective *mapping $f : V(g') \rightarrow V(g)$, s.t.,: first, $\forall v' \in V(g'), L'(v') = L(f(v'))$; and second, $\forall (v'_1, v'_2) \in E(g'), (f(v'_1), f(v'_2)) \in E(g)$, and* vice versa.

The purpose of choosing induced subgraphs is to make Definition 1 (calculation of structural difference) and Definition 2 (decision of smoothed support) consistent. By writing the first definition, we are imposing the following opinion regarding whether two graphs are look-alikes or not, i.e., we will only deem them as similar if the two sets of nodes are interacting in nearly the same way as each other. This requires us to extract all edges among a given set of vertices, which conforms to the induced scenario. However, whether to stick with such an induced scheme does depend on real applications. For some cases, where we are not necessarily interested in every interconnection in between, the non-induced setting might be more appropriate, because it does not require the pattern as a subgraph to cover every edge that appears in its data graph counterpart. Fortunately, despite all these considerations with respect to user intentions, the methodology we propose in this paper is general enough to handle both cases, which will be further discussed in Section 5.

Example 1 *Suppose we have a graph transaction database as depicted in the top part of Fig.1, i.e., there are 10, 2, 2, 2 copies of d_1, d_2, d_3 and d_4, respectively. Set min_sup=2, we can find 7 frequent induced subgraphs, with their original supports listed in the second row of Fig.2. When δ is 1, i.e., any one-edge difference is gracefully tolerated, the smoothed supports of the same set of patterns are shown in the third row of Fig.2. For example, p_1 can absorb the supporting transactions of p_2, p_3, p_4 and essentially enhance its support to 10+2+2+2=16.*

4. THE CLUSTERING STEP

After examining the micro aspect, where smoothing is applied to resolve the structural restrictiveness of exact patterns, we now turn to the macro aspect of the problem. Unfortunately, even though a pattern's support is no longer confined to the number of times it precisely appears in the data, there are yet too many number of patterns. Interestingly, if we think from the following perspective, the smoothing step actually hints a way for us to cut down result cardinality in this section: The fundamental basis for us to introduce some structural tolerance is that, by specifying a parameter δ, users are indeed willing to accept all minor variations within this threshold, i.e., they essentially treat two patterns p_1, p_2 as "equivalent" if $diff(p_1, p_2) \leq \delta$; now, based on the same rationale, since p_1, p_2 are deemed as interchangeable, can we just retain one of them in the final output and remove the other? Our answer to this question is yes, which leads to the clustering step we are going to elaborate.

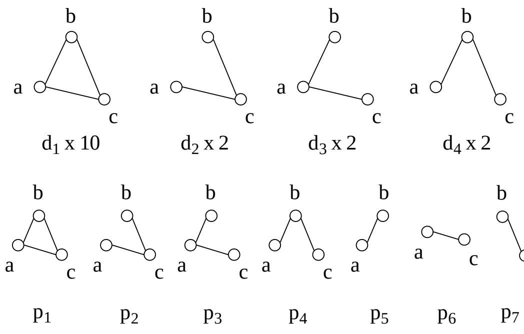


Figure 1: Example Graph Database and Patterns

pattern p_i	p_1	p_2	p_3	p_4	p_5	p_6	p_7
original support $\widehat{sup}(p_i)$	10	2	2	2	14	14	14
smoothed support $sup(p_i)$	16	12	12	12	14	14	14

Figure 2: Original and Smoothed Supports

Example 2 *Still looking at Fig.1 and Fig.2, the structures of p_2, p_3, p_4 are quite similar to that of p_1 (within a δ of 1), where p_5, p_6, p_7's structures can also be represented by p_1 because p_1 contains them. Meanwhile, as we can see, $p_2, p_3, \ldots, p_7$'s smoothed supports are all very close to p_1's. All these lead us to propose $\{p_1$: $sup(p_1) = 16\}$ as an effective presentation for the whole set of 7 patterns. Note that, without the smoothing step, the original supports of p_2, p_3, p_4 are far away from that of p_1, and thus in the exact sense, it will be better to add these patterns as three additional representatives.*

From the above example, we can see that, there is possibility for us to represent a pattern p_1 by another pattern p_2, so that the whole pattern set can be shrunk to a small group of delegates. Towards this objective, we will take a clustering approach. As one might expect, some qualifying conditions need to be satisfied, which justify a good presentation of p_1 by p_2. In this sense, we can cast the degree of such goodness to a distance function $d(p_1, p_2)$, and produce a much less number of clusters according to the patterns' mutual distances. Then, as a result of this processing, the cluster centers will be taken as final representatives and passed on to the users.

Considering that patterns produced by graph mining algorithms are indeed a combined list of topological structures and enumerated supports, in order to delegate p_1 by p_2 so that p_1 can be removed from the final output, conditions from the following two aspects must be satisfied.

- Structure Representability: In terms of graph topology, we must have p_1's structure well reflected by that of p_2.
- Support Preservation: In terms of significance measure, we must have p_1's support very close to that of p_2.

In the first place, let us look at structure representability. As we motivated in Example 2, there are two cases here: (1) If two patterns g', g comprise the same set of labeled vertices, and their mutual interactions are nearly identical to each other, i.e., the structural difference given in Definition 1 turns out to be less than δ, then we can use one of g', g to represent the other; and (2) Different from the circumstance in (1), if pattern g has more vertices than pattern g', a natural consideration is that g can represent g' if g' is "contained" in g, because the structure of g gives us full information about the structure of g'. According to Definition 3, this means that g' is an induced subgraph of g, and some previous works have also leveraged such subpattern-superpattern relationships in compressing/summarizing itemset patterns [20, 21]. Finally, these two cases can be unified, which generalize to a new concept: *approximate subgraph isomorphism.*

Definition 4. *(Approximate Subgraph Isomorphism). For two graphs g' and g, we say that g' is an* approximate subgraph *of g if there exists an* induced subgraph *g'' of g such that $diff(g', g'') \leq \delta$, i.e., g' and g'' are approximately isomorphic to each other.*

For the rest of this paper, we will write $g' \subseteq_{\delta}^{induce} g$ if g' is an approximate subgraph of g in the induced sense. Obviously, $g' \subseteq_{0}^{induce} g$ (i.e., no tolerance) degenerates to the simple rigid case of exact induced subgraph.

Looking at Definition 4, for any two graphs g' and g (without loss of generality, suppose $|V(g')| < |V(g)|$), if there exists a bijective, label-preserving mapping f between $V(g')$ and a subset of $V(g)$, then the mutual relationship between g' and g can always be decomposed into: (1) a structural difference part between g' and g'', and (2) a subgraph containment part between g'' and g. Now, since containment-based representation is not subject to any further conditions, we can disregard the second part from structure representability considerations, i.e., as long as the first part is bounded by a user-acceptable level δ, there would be no problem to represent g' by g in terms of graph topology.

Definition 5. *(Structure Representable). For two graphs g' and g, the* structural representability *of g in place of g' (i.e., use g as the delegate for g') is defined as:*

$$R_S(g', g) = \min_{|V(g')|=|V(g'')|, g'' \subseteq_0^{induce} g} diff(g', g''),$$

where $|V(g')| = |V(g'')|$ means that g' and g'' have the same number of vertices, $g'' \subseteq_0^{induce} g$ means that g'' is an induced subgraph of g, and R_S stands for representability on structure. We say that g' is structure representable *by g if $R_S(g', g)$ is less than or equal to the predetermined error tolerance δ.*

In above, g'', as an induced subgraph of g, is specifically chosen so that the structural difference $diff(g', g'')$ is minimized. This in fact suggests the most efficient way of editing (i.e., through edge insertions/deletions) g' to make it become an induced subgraph of g.

We now move on to support preservation, which is a bit more straightforward. Depending on different user intentions, i.e., what kind of information is expected to be conveyed by support, we have multiple choices here. Note that, all support mentioned here has already been smoothed.

First, if support is simply treated as a number that quantifies pattern frequencies in the dataset, then we do not need to worry much except the support itself. We can say that g''s support is *well-preserved* by g if

$$P_S(g', g) = \frac{|sup(g) - sup(g')|}{\max\{sup(g), sup(g')\}} \leq \epsilon,$$

where P_S stands for preservation on support, and ϵ is another tolerance parameter, whose function is comparable to that of δ.

Second, as one might think, other than purely numeric frequencies, pattern mining results are usually associated with a set of occurrence lists, which link each pattern to those places where it appears in the data. In the scenario of graph transaction databases, this means that people may often want a pattern's support to be conveyed in the sense of its supporting transactions. Having this in mind, we can lay down another definition and prescribe that g''s support is *well-preserved* by g if

$$P_S(g', g) = 1 - \frac{|\mathcal{D}_{g'} \cap \mathcal{D}_g|}{|\mathcal{D}_{g'} \cup \mathcal{D}_g|} \leq \epsilon.$$

Here, $P_S(g', g)$ takes the form of a Jaccard distance between the supporting transactions of g' and g.

Interestingly, if we ignore different transaction IDs and treat $\mathcal{D}_{g'}/\mathcal{D}_g$ as a set of $sup(g)/sup(g')$ ones: $\{1, 1, \ldots, 1\}$, which essentially means that users are not concerned about the specific locations where the pattern occurs, then

$$1 - \frac{|\mathcal{D}_{g'} \cap \mathcal{D}_g|}{|\mathcal{D}_{g'} \cup \mathcal{D}_g|} = \frac{|sup(g) - sup(g')|}{\max\{sup(g), sup(g')\}},$$

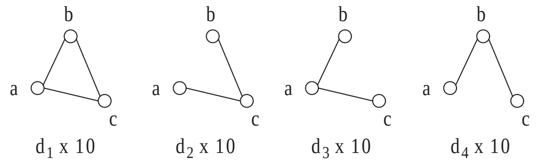
indicating that the above two definitions can indeed be consolidated under the same framework.

Example 3 *Let us consider a variation of Examples 1-2, which is depicted in Fig.3: Now, instead of 10, 2, 2, 2 copies of d_1, d_2, d_3, d_4, we have 10, 10, 10, 10 of them respectively. Based on the second option in above, i.e., support preservation is calculated based on supporting transactions, Fig.4 shows $P_S(p_j, p_i)$ for each pair of patterns in a symmetric matrix. Compared to the previous situation, it now becomes less desirable to delegate p_2, p_3, p_4 by p_1 because of their differences on support: Looking at corresponding entries in the matrix, $P_S(p_2, p_1) = P_S(p_3, p_1) = P_S(p_4, p_1) = 0.5$ is no longer a small number. In this sense, we choose to present the whole set of 7 patterns by*

$$\{ \quad p_1 : sup(p_1) = 40, \ p_2 : sup(p_2) = 20, \quad p_3 : sup(p_3) = 20, \ p_4 : sup(p_4) = 20 \quad \},$$

where p_5, p_6, p_7 are still delegated by p_1, after introducing p_2, p_3, p_4 as three additional representatives.

We may interpret this result as follows. If we denote label a as "UIUC", label b as "IBM Research" and label c as "Northwestern", while each of d_1, d_2, d_3, d_4 corresponds to a sample of three researchers taken from these three institutions, then an edge between two vertices can be thought as an indication of close collaborating relationship. For example, 10 copies of d_4 in Fig.3 means that there are 10 samples out of 40 where the UIUC person and the Northwestern person both work closely with the IBM person, though they do not collaborate much with each other. For the case of Examples 1-2, the majority of samples from three institutions have pairwise research connections among them (corresponding to pattern p_1). In comparison, the situation is much more diverse here, and intuitively this will necessitate more representatives.



pattern p_i	p_1	p_2	p_3	p_4	p_5	p_6	p_7
original support $\widehat{sup}(p_i)$	10	10	10	10	30	30	30
smoothed support $sup(p_i)$	40	20	20	20	30	30	30

Figure 3: A Variation of Examples 1-2

	p_1	p_2	p_3	p_4	p_5	p_6	p_7
p_1	0.00						
p_2	0.50	0.00					
p_3	0.50	0.67	0.00				
p_4	0.50	0.67	0.67	0.00			
p_5	0.25	0.75	0.33	0.33	0.00		
p_6	0.25	0.33	0.33	0.75	0.50	0.00	
p_7	0.25	0.33	0.75	0.33	0.50	0.50	0.00

Figure 4: The Matrix of $P_S(p_j, p_i)$

Now, we are ready to formalize the problem of presenting graph patterns through structural representatives, which is stated in Definition 6. Intuitively, clustering is a natural choice for such representative selection tasks: Based on aforementioned structure representability and support preservation, which are formulated as distance functions, we can partition all patterns into a bunch of clusters and pick their centers to approximately delegate the rest. Here, structural tolerance δ, number of clusters k, and support preservation threshold ϵ are all user-set parameters, which can control the clustering process and in turn reflect the amount of approximation that is desired.

DEFINITION 6. *(Presenting Graph Patterns through Structural Representatives). Given a set of n frequent patterns that are mined from the graph database $\mathcal{D}$: $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ and their associated supports, find a subset of representatives $\mathcal{R} \subseteq \mathcal{P}$ such that for each $p \in \mathcal{P}$, there exists at least one pattern $r \in \mathcal{R}$ satisfying: first, p is structure-representable by r; and second, p's support is well-preserved by r.*

In below, we will focus on actual implementations of the clustering step. Given the pattern set $\{p_i\}$ and their smoothed supporting transactions $\{\mathcal{D}_{p_i}\}$, we develop two algorithms to partition $\mathcal{P}$, which are constructed from different perspectives. The first one is called ϵ-bounded clustering, it inherits bottom-up ideas and uses a parameter ϵ to control the local tightness of each cluster. The second one applies k-medoids clustering, it has top-down properties and uses a parameter k to control the global number of clusters.

4.1 ϵ-bounded Clustering

As we suggested in above, in order to guarantee tightness, a parameter ϵ can be put on the clustering's local side, which directly bounds the maximal "radius" of each cluster. Intuitively, the larger ϵ is and the more amount of approximations that are allowed, the less number of clusters. Now, with fixed threshold ϵ, we say that a pattern p_j can be delegated by another pattern p_i, if: (1) p_j is structure representable by p_i (see Definition 5), and (2) p_j's support is well-preserved by p_i, i.e., $P_S(p_j, p_i) \le \epsilon$.

When these two conditions are satisfied, p_i is a qualified structural representative for p_j, which essentially creates an *ϵ-cluster* $C(p_i)$ centered at p_i:

$$\begin{aligned} C(p_i) \quad = \quad & \{p_j | p_j \in \mathcal{P}, P_S(p_j, p_i) \le \epsilon, \\ & \quad \text{and } p_j \text{ is structure representable by } p_i\}. \end{aligned}$$

$C(p_i)$ covers all instances in $\mathcal{P}$ that can be removed from the pattern set as long as p_i itself is retained, while the degree of information loss for doing so is upperbounded by ϵ.

At this stage, having n ϵ-clusters $C(p_i)$ for each pattern $p_i \in \mathcal{P}$, our task is transformed into the selection of $n' < n$ ϵ-clusters so that these n' cluster centers can represent the whole pattern set. Obviously, each $p_i \in \mathcal{P}$ is contained in at least one cluster $C(p_i)$, while it might also exist in some other clusters. Now, in order to promote result cardinality reduction and thus expose users to the least amount of information under a specific approximation level ϵ, we want to choose as few ϵ-clusters as possible, if they can fully cover all the patterns in $\mathcal{P}$.

Treating the total number of n patterns as a universal collection and each ϵ-cluster as a set that contains several elements, we resort to *set cover* for the implementation of ϵ-bounded clustering. As a classical NP-Complete problem [8], a well-known approximation algorithm for set cover is to greedily choose a set that contains the largest number of uncovered elements at each stage. Algorithm 1 transfers this strategy to our scenario here.

Algorithm 1 The ϵ-bounded Clustering

```
Input: the pattern set $\mathcal{P}$, the $\epsilon$-clusters $\{C(p_i)\}$.
Output: picked cluster centers $\mathcal{R}$.

1: $P_r = \mathcal{P}$;
2: $\mathcal{R} = \varnothing$;
3: while $P_r \ne \varnothing$ do
4:     select the $\epsilon$-cluster $C(p_i)$ that currently contains
       the most number of patterns in $P_r$;
5:     $\mathcal{R} = \mathcal{R} \cup \{p_i\}$;
6:     $P_r = P_r - C(p_i)$;
7: return $\mathcal{R}$;
```

4.2 k-medoids Clustering

In ϵ-bounded clustering, a pattern can exist in multiple clusters and thus be delegated by more than one structural representative that is selected; meanwhile, sometimes users may also feel difficult to specify an appropriate tightness ϵ, if they are not so familiar with the data. Thus, as an alternative method, we develop the k-medoids clustering algorithm, which is comparable to its counterpart in general unsupervised analysis, where users only need to designate the number of representatives k they want to see, and each pattern is assigned to only one cluster.

Define a distance function $d(p_i, p_j)$ between two patterns $p_i, p_j \in \mathcal{P}$ as

$$d(p_i, p_j) = \begin{cases} P_S(p_j, p_i) & \text{if } p_j \text{ is structure representable by } p_i \\ +\infty & \text{otherwise} \end{cases}$$

we have the k-medoids clustering strategy described in Algorithm 2. At first, k patterns $M = \{m_1, m_2, \ldots, m_k\}$ are randomly selected from $\mathcal{P}$ to act as the initial medoids; and then in each step, one medoid pattern is swapped with another non-medoid pattern, which aims to reduce the following distance-to-medoids:

$$D(M) = \sum_{p \in \mathcal{P}} \big[\min_i d(m_i, p) \big]^2.$$

The above procedure is iteratively continued, until reaching a local minimum of $D(M)$. Previously proposed k-medoids algorithms mainly differ in the way they perform swaps between steps: PAM [10] picks a pair reducing $D(M)$ the most at each step, which includes an extensive examination of every possible swapping; CLARANS [15] refuses to do this, it will continue as soon as it finds something that can decrease the objective function. We will follow CLARANS's idea in this paper, because it has been shown that the method gives similar clustering quality but is far more efficient than PAM due to its randomized search, which is ideal for large-scale applications.

Algorithm 2 The k-medoids Clustering

```
Input: the pattern set $\mathcal{P}$, the number of clusters $k$,
       and the distance function $d(\cdot, \cdot)$;
       a repetition parameter $maxtocheck$.
Output: $k$ cluster medoids $M = \{m_1, m_2, \ldots, m_k\}$.

1: start from $k$ patterns in $\mathcal{P}$ to initialize $M$;
2: $j = 0$;
3: consider a random pair $m \in M$ and $p \notin M$ to be swapped,
   let $M' = M \cup \{p\} - \{m\}$;
4: if $D(M') < D(M)$ then
5:     $M = M'$, goto step 2;
6: else
7:     $j = j + 1$;
8:     if $j < maxtocheck$ then goto step 3; else return $M$;
```

5. DISCUSSIONS

In this section, we discuss some variants and extensions of the smoothing-clustering framework that has been proposed.

5.1 The Non-Induced Case

Our previous discussions have been using Definition 1 to quantify the structural difference *diff*(g', g) between two patterns g' and g. Since all edges among a set of vertices are taken into consideration, it actually corresponds to the induced notion of graph theory. However, as we mentioned in Section 3, for some application scenarios, it might not be appropriate to stick with this setting if the users are actually expecting some non-induced scheme, which has been followed by quite a few knowledge discovery tasks on graphs.

In a similar fashion, we can define *approximate subgraph isomorphism* for the *non-induced* scenario. For two graphs g' and g, we say that g' is an *approximate subgraph* of g in the *non-induced* sense if there exists an *injective*, *label-preserving* mapping $f : V(g') \rightarrow V(g)$ such that the following *asymmetric difference*:

$$\sum_{v_1', v_2' \in V(g')} \max\left\{0, I'(v_1', v_2') - I(f(v_1'), f(v_2'))\right\}$$

is less than or equal to δ.

Due to the non-induced nature, the above asymmetric difference only considers edges that are present in g' but not in g, i.e., $I'(v_1', v_2') = 1$ and $I(f(v_1'), f(v_2')) = 0$, which is in evident distinction with respect to the induced scenario. Actually, if there are no such edges, then the calculated asymmetric difference would be 0 – consistent with the fact that g' is now an exact non-induced subgraph of g. Under this extreme case, the dissimilarities between g' and g are reflected through support, but not through structure (g have some additional vertices/edges compared to g', but they belong to the containment part and are thus not treated as structural differences): In fact, if their supports also coincide, then g can well represent g' without losing any information, which corresponds to the concept of closeness in mining theory.

Compared to $g' \subseteq_\delta^{induce} g$, we can write $g' \subseteq_\delta g$ if g' is an approximate subgraph of g in the non-induced sense. For the smoothing step, we need to enhance each p_j's support by that of p_i if $p_i \subseteq_\delta p_j$ and $|V(p_i)| = |V(p_j)|$; while for the clustering step, g' is said to be structure representable by g if $g' \subseteq_\delta g$. These are all we need to modify in the original framework.

5.2 The Single Graph Case

Conceptually, there are no difficulties to extend our framework to the single graph case, because its starting point is a set of patterns plus their supports, which are given by mining algorithms in both scenarios: We can still get these raw patterns, compute the smoothed supports and perform clustering. There is only one complication: Unlike the transaction database setting, where support equals the number of graph transactions that contain some pattern, a common strategy adopted here is to count the *maximal number of edge-disjoint embeddings* [12], i.e., different occurrences must not overlap, so that the anti-monotonicity is still guaranteed. Now, as we work with support preservation, it could be complex to adopt the second option, which relies on well-defined supporting transactions; while for the first option that reflects numeric supports, $P_S(g', g)$ takes the form of

$$\frac{|a \cdot sup(g) - sup(g')|}{\max\{a \cdot sup(g), sup(g')\}},$$

where a is an adjusting factor equivalent to g''s support in g: Since a pattern can appear multiple times in a single graph, it is straightforward to imagine that g''s support would be around a times $sup(g)$ if we use g to delegate g'.

5.3 Size-Increasing Structural Tolerance

In above, we have been using a uniform structural tolerance δ in all situations. However, intuitively, shall we allow a greater tolerance for larger patterns than for smaller patterns? For example, in a graph with 4 edges, a difference of 2 edges means 50% approximation; while in a graph with 20 edges, it is only 10%. To reflect this idea, when considering approximate subgraph isomorphisms $g' \subseteq_\delta^{induce} g$ and $g' \subseteq_\delta g$ in either the smoothing step or the clustering step, instead of a constant, we can let δ be an increasing function of g''s size, i.e.,

$$\delta = \delta(|V(g')|) \text{ or } \delta = \delta(|E(g')|),$$

which is called a *size-increasing structural tolerance*. Here, two interesting choices could be $\delta_1 = \alpha|V(g')|$ and $\delta_2 = \alpha|E(g')|$, where α is a percentage constant, with δ_1 being tolerance proportional to the number of vertices and δ_2 being tolerance proportional to the number of edges.

5.4 More Complex Approximate Matchings

Remember that, when two graphs g' and g's structural difference was characterized in Definition 1, the configuration is restricted in the following two senses. First, we stipulate that, even for two graphs to be approximately isomorphic, their vertex labels should still match exactly, which was meant to focus our discussions on the difference with regard to interconnecting topology. This is plausible if the label, as a categorical attribute, represents a strict classification of node entities, where two classes are not compatible to each other at all. However, based on different application scenarios, the above setting might be too rigid, e.g., in a protein network, though protein A is not identical to protein B, mutual substitutions can still happen between them under certain situations. Second, concerning edges, we have only differentiated two possibilities, i.e., there is an edge or not. In general, edges are also associated with labels, which incorporate relevant information, e.g., there could be l_e types of edges and $l_e + 1$ different cases: type-1, type-2, …, type-l_e plus *null*, where *null* means no edge exists.

Formally, we can define two distance matrices: (1) $dist_v[i, j]$, an $l_v \times l_v$ matrix, with l_v being the number of different vertex labels, and (2) $dist_e[i, j]$, an $(l_e + 1) \times (l_e + 1)$ matrix. These two matrices are symmetric, have zero diagonals, whose $(i, j)^{th}$ entry is a quantification of the dissimilarity between the i^{th} and j^{th} vertex/edge label. In practice, they can be determined by domain experts, reflecting their knowledge on the difficulty of having a corresponding replacement: For example, it might be easier for an edge with label 1 to change its label to 2 than to *null*, which means to completely diminish that edge; and this will result in a smaller penalty score for $dist_e[1, 2]$ than for $dist_e[1, null]$ in the matrix.

With all above preparations, we can rewrite the formula in Definition 1 as:

$$\begin{aligned} \mathit{diff}(g', g) \;=\; \min_{\forall f : V(g') \leftrightarrow V(g)} \Bigg\{ & \sum_{v' \in V(g')} dist_v\left[L'(v'), L(f(v'))\right] + \\ & \sum_{v_1', v_2' \in V(g')} dist_e\left[L'(v_1', v_2'), L(f(v_1'), f(v_2'))\right] \Bigg\}, \end{aligned}$$

where the label function L'/L is now applied on both vertices and edges. More clearly, for an edge between u and v, its label is $L(u, v)$, with $L(u, v) = null$ if there is no edge. This new way of modeling structural difference can be easily inserted into our current framework, which is also straightforward to be extended to the non-induced case.

6. EXPERIMENTAL RESULTS

In this section, we provide empirical evaluations on our algorithm of presenting graph patterns through structural representatives. We use two kinds of datasets in this study, one real dataset and one synthetic dataset. All experiments are done on a Microsoft Windows XP machine with a 3GHz Pentium IV CPU and 1GB main memory. Programs are compiled by Visual C++.

6.1 Real Dataset

Chemical Compounds. The first, AIDS anti-viral screen dataset contains the graph structures of chemical compounds, which is publicly available on the Developmental Therapeutics Program's website. It consists of more than 40,000 molecules, and has been used as an important benchmark for various knowledge discovery tasks on graphs. In this data, vertices are labeled with Carbon (C), Hydrogen (H), Oxygen (O), etc.; while an edge exists if there is a corresponding chemical bond. Based on the state-of-art graph mining tools, we use a minimum support of 8% and generate 5,304 frequent non-induced subgraphs, which are taken as the input $\mathcal{P}$ for our smoothing-clustering framework.

Experiment Settings. Clearly, given such a large amount of patterns that are nonetheless redundant, it is very hard for any users to make sense of them. To maintain consistency with the mining algorithm, we adopt the non-induced setting to characterize structural differences (see Section 5.1), while support preservation is computed based on supporting transactions (instead of pure frequency numbers), which are well-defined here. Note that, when performing the smoothing and clustering steps, we have to compare each pair of patterns so that structure representability and support preservation can be decided, especially for the smoothing step, where the determination of approximate subgraph isomorphism is somehow costly. We implement a few heuristics to accelerate this. First, we make sure that p_j's vertex label set is contained in that of p_i before any real matching is done, because otherwise p_j will not be structure representable by p_i anyway. Second, in computing $\mathit{diff}(g', g)$, which is the sum over a series of non-negative numbers, we designate an upperbound of δ_{max} and stop whenever the addition reaches this value; the result thus achieved can be reused for many rounds of clustering as long as the approximation parameter δ is less than δ_{max}.

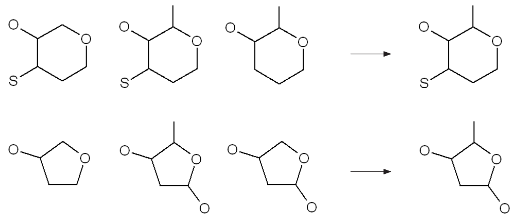


Figure 5: Sample Representatives

Fig.5 gives some sample representatives we get from this dataset, which are shown on the right-hand side; on the left-hand side, it lists a few original patterns that are delegated by them. In order to make the whole picture easy to read, we omit all Hydrogens from the drawing, while a vertex without label is by default a Carbon, e.g., the "O" with one edge is in fact an "OH", and the (unmarked) "C" with two edges is in fact a "CH_2". There are some chemical properties associated with each cluster, which are further epitomized by the corresponding representative: The hexagon shape of patterns on the first row is typical of hexose (e.g., glucose, galactose belong to this category), while the pentagon shape of patterns on the second row is typical of pentose (e.g., ribose, desoxyribose belong to this category).

Fig.6 is an examination on the effect of smoothing. Here, patterns are grouped into bins based on their number of edges, which is shown on the x-axis; meanwhile, we plot the average support of patterns within each bin, which is shown on the y-axis. Clearly, $\delta = 0$ corresponds to the original supports that are not smoothed. As we can see, even using $\delta = 1$ will largely boost the curve, which means that the pattern set is highly redundant where a lot of frequent subgraphs are so similar to each other (the difference is only one edge). In comparison, the gap between $\delta = 1$ and $\delta = 2$ is much smaller, hinting that $\delta^* = 1$ could be a reasonable setting for the chemical compound dataset. To this extent, we will focus more on the $\delta = 1$ case for the rest of Section 6.1. There is only one thing that might seem counter-intuitive: For $\delta = 1$ and $\delta = 2$, sometimes the support slightly goes up as the pattern size increases. Actually, this can happen after smoothing: An easy-to-think example is Fig.1, where p_5 is an induced subgraph of p_1, but $sup(p_1) > sup(p_5)$.

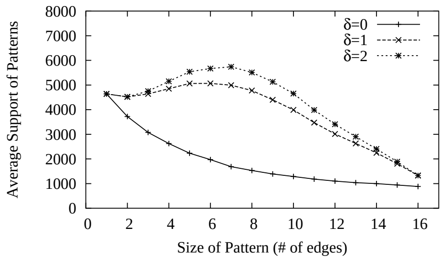


Figure 6: Effect of Smoothing

Fig.7 depicts ϵ-bounded clustering. It is easy to imagine that, as the threshold ϵ becomes larger, less and less cluster centers will suffice to represent the whole pattern set. Also, the large gap between $\delta = 1$ and $\delta = 0$ indicates that structural relaxation and thus structural representative is a necessary mechanism to achieve effective pattern presentation. Empirically, $\epsilon^* = 0.15$ or 0.2 could be a suitable parameter to work with, because from 0 to ϵ^*, the number of clusters reduces significantly, while beyond that range, the gain to further increase ϵ is less obvious.

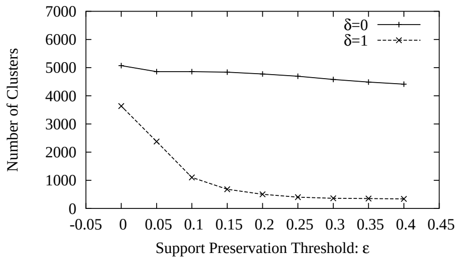


Figure 7: ϵ-bounded Clustering

Fig.8 depicts k-mediods clustering, where $maxtocheck$ is set as 2000. Similar to tests on ϵ-bounded clustering, we want to examine the relationship between number of clusters k and clustering quality, which is reflected by the following measure of average error:

$$E = \frac{1}{|\mathcal{P}|}D(M) = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\left[\min_i d(m_i,p)\right]^2,$$

where M is the final set of medoids. The general trend is quite similar to Fig.7 (suggesting $k^* = 300$ as a suitable parameter setting), except that the support preservation threshold there bounds on the maximal distance from each pattern to its cluster center, while the error E defined here calculates an average, which is somehow smaller. We decide to omit the $\delta = 0$ curve, because for such a zero structural difference case, k on the x-axis must be set very large so that each pattern is structure representable by at least one medoid, and this will dwarf the more interesting $\delta = 1$ curve.

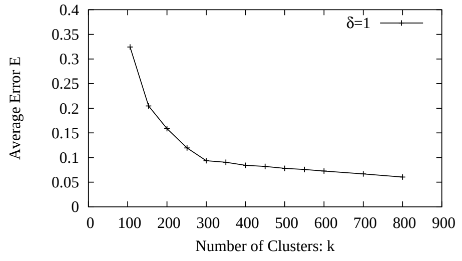


Figure 8: k-medoids Clustering

Fig.9 draws two distributions for the original patterns $\mathcal{P}$ and final representatives $\mathcal{R}$, respectively. Like in Fig.6, patterns are binned based on their number of edges, with its y-axis now showing the number of patterns for each bin. ϵ-bounded clustering with $\delta = 1$ and $\epsilon = 0.15$ is applied. The downward tendency of $\mathcal{R}$ with respect to $\mathcal{P}$ is very clear. Moreover, an interesting observation is that, the degree of descending is even bigger for the curve's mid-part, which corresponds to the high-frequency area if we look at the distribution of $\mathcal{P}$. This demonstrates the effectiveness of our algorithm in the dense region of data.

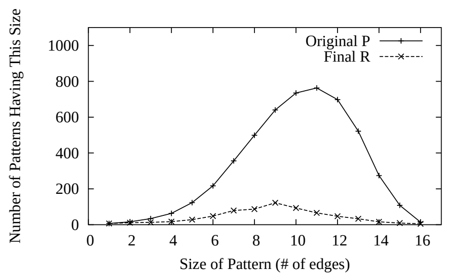


Figure 9: Distribution of Patterns

Fig.10 describes what happens when a size-increasing structural tolerance is used. Based on Section 5.3, we adopt the following function:

$$\delta = round(\alpha|E(g')|),$$

where α is set at 0.125. ϵ-bounded clustering is tested. Due to the rounding that is applied, δ starts from 0 for one-edge patterns and gradually augments to 1 (at 4-edge), 2 (at 12-edge), and so on. Looking at Fig.9, the fraction of patterns with less than 4 edges is pretty small, which means that in overall, the constraint imposed by the scheme here is less rigid than that of a uniform $\delta = 1$. This might explain the non-uniform curve's lower position in Fig.10.

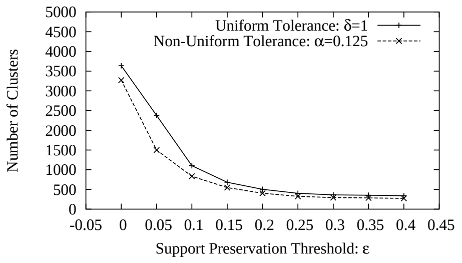


Figure 10: Non-Uniform Structural Tolerance

6.2 Synthetic Dataset

Generator Description. The synthetic graph generator follows a similar mechanism as that was used to generate transaction itemset data [1], where we can set the number of graphs (D), average size of graphs (T), number of seed patterns (L), average size of seed patterns (I), and number of distinct vertex/edge labels (V/E). First, a set of L seed patterns are generated randomly, whose size is determined by a Poisson distribution with mean I; then, seed patterns are randomly selected and inserted into a graph one by one until the graph reaches its size, which is the realization of a Poisson random variable with mean T. Due to lack of space, readers are referred to [11] for further simulation details.

The data set we take is D10kT20L200I10V(l_v)E1, i.e., 10,000 graphs with 20 vertices on average, which are generated by 200 seed patterns of average size 10; the number of possible vertex and edge labels are set to l_v and 1, respectively. Other experiment settings are as same as those in Section 6.1. We vary l_v from 6, 8, 10, 12, up to 15, and use a fixed $min\_sup = 6\%$ to mine frequent subgraphs from these five datasets, which are further processed by smoothing and ϵ-bounded clustering (use $\epsilon = 0.2$). The result is depicted in Fig.11, where in order to account for the fact that the original pattern set's cardinality $|\mathcal{P}|$ is different for each dataset, we normalize the number of final representatives $|\mathcal{R}|$ by dividing $|\mathcal{P}|$, which is called *reduction ratio* and shown on the y-axis. Based on the upward trend of curves, it can be seen that the benefit achieved by our algorithm shrinks as l_v becomes larger, which is natural, because in the normal case, vertex labels must be exactly matched (even increasing the error tolerance δ cannot change this), and more labels will surely increase the diversity of data. Section 5.4 offers a solution to alleviate this effect, if distinct labels are not absolutely non-substitutable.

Taking D($|\mathcal{D}|$)T20L200I10V10E1, we also tested the efficiency of our algorithms over five synthetic datasets by varying the number of transactions $|\mathcal{D}|$ from 5,000, 10,000, up to 25,000, which is shown in Fig.12. Here, a fixed $min\_sup = 6\%$ is used and around 1,500 frequent subgraphs are generated for each dataset. The total running time is composed of two parts: (1) the smoothing time (use $\delta = 1$), which enhances the support of each pattern by those of its look-alikes, and (2) the clustering time (use $\epsilon = 0.2$ for ϵ-bounded

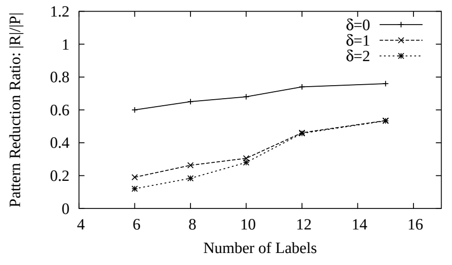


Figure 11: Performance w.r.t. Number of Labels

clustering and $k = 500, maxtocheck = 2000$ for k-medoids clustering) that produces a bunch of final representatives. It can be seen that the implementation is highly efficient, which can finish in tens of seconds, while both of our algorithms are linearly scalable.

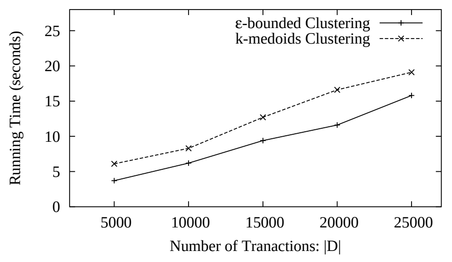


Figure 12: Scalability Tests

7. CONCLUSIONS

We examine the presentation problem of graph patterns and solve the usability issue raised at the beginning of this paper. Instead of too many exact patterns that are not so meaningful, a compact group of informative delegates are generated and shown to the users. Structural representatives play a key role in this novel, integrative approach, which essentially rolls-up our attention to a more appropriate level that no longer looks into every minute detail. The smoothing-clustering framework is nearly universal, which can handle many variants of the problem (induced/non-induced, transaction database/single graph) and incorporate other extensions as well. Finally, empirical studies confirm the effectiveness and efficiency of our proposed algorithms. As a promising future direction, since the method described here belongs to the post-processing category, we are now working on advanced algorithms that can directly mine such representative patterns from data.

8. REFERENCES

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

[2] R. J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD Conference*, pages 85–93, 1998.

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

[4] C. Chen, X. Yan, F. Zhu, and J. Han. gapprox: Mining frequent approximate patterns from a massive network. In *ICDM*, pages 445–450, 2007.

[5] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050, 2005.

[6] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *ICDM*, pages 211–218, 2002.

[7] M. Hasan, V. Chaoji, S. Salem, J. Besson, and M. Zaki. Origami: Mining representative orthogonal graph patterns. In *ICDM*, pages 153–162, 2007.

[8] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing, 1997.

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

[10] L. Kaufman and P. J. Rousseeuw, editors. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley and Sons, 1990.

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

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

[13] Y. Liu, J. Li, and H. Gao. Summarizing graph patterns. In *ICDE*, pages 903–912, 2008.

[14] T. Mielikäinen and H. Mannila. The pattern ordering problem. In *PKDD*, pages 327–338, 2003.

[15] R. T. Ng and J. Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

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

[17] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416, 1999.

[18] R. Sharan, S. Suthram, R. M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. M. Karp, and T. Ideker. Conserved patterns of protein interaction in multiple species. *PNAS*, 102(6):1974–1979, 2005.

[19] C. Wang and S. Parthasarathy. Summarizing itemset patterns using probabilistic models. In *KDD*, pages 730–735, 2006.

[20] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In *VLDB*, pages 709–720, 2005.

[21] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: a profile-based approach. In *KDD*, pages 314–323, 2005.

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

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