Taxonomy-Superimposed Graph Mining

Ali Cakmak and Gultekin Ozsoyoglu

Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, OH 44106
{cakmak, tekin} @case.edu

ABSTRACT

New graph structures where node labels are members of hierarchically organized ontologies or taxonomies have become commonplace in different domains, e.g., life sciences. It is a challenging task to mine for frequent patterns in this new graph model which we call taxonomy-superimposed graphs, as there may be many patterns that are implied by the generalization/specialization hierarchy of the associated node label taxonomy. Hence, standard graph mining techniques are not directly applicable.

In this paper, we present Taxogram, a taxonomy-superimposed graph mining algorithm that can efficiently discover frequent graph structures in a database of taxonomy-superimposed graphs. Taxogram has two advantages: (i) It performs a subgraph isomorphism test once per class of patterns which are structurally isomorphic, but have different labels, and (ii) it reconciles standard graph mining methods with taxonomy-based graph mining and takes advantage of well-studied methods in the literature. Taxogram has three stages: (a) relabeling nodes in the input database, (b) mining pattern classes/families and constructing associated occurrence indices, and (c) computing patterns and eliminating useless (i.e., over-generalized) patterns by post-processing occurrence indices. Experimental results show that Taxogram is significantly more efficient and more scalable compared to other alternative approaches.

1. INTRODUCTION

Graph databases are prevalent in various fields [1, 15] to represent and query complex relationships between objects. Mining frequent structures in graph databases has recently drawn considerable research attention [6, 7, 10, 11, 20, 22]. Traditional general-purpose graph mining approaches have focused on extracting frequent graph structures that explicitly appear in a graph database. Recently, in various fields, new graph structures have emerged, where vertex labels are members of a taxonomy defined by *is-a* or *part-of* relationships between a set of labels in a hierarchical manner. In this paper, we refer to this type of graphs as *taxonomy-superimposed graphs*. As one example, biological pathways are graphs of interacting proteins, and proteins are usually annotated with functionality concepts from Gene Ontology [4], which is a taxonomy containing around 20,000 concepts organized in a hierarchical manner. In such an environment, mining for frequent pathway annotation structures across organisms is important to understand common pathway functionality structures in different organisms as well as to predict pathways [2] in newly sequenced organisms, and categorize existing pathways into groups [3].

A major implication of the taxonomy-superimposed graph model is that, now, we are interested in discovering frequent graph structures which do not necessarily appear explicitly in a graph database, but can be discovered only when employing the hierarchical relationships defined in the associated taxonomy. In such a model, the traditional general-purpose graph mining algorithms are not directly applicable to mine implicitly occurring patterns. We give an example.

Example 1.1. Consider a sample "pathway annotation" graph database in Figure 1.2 where node labels are from a subgraph of Gene Ontology (GO) shown in Figure 1.1. Assume that we would like to mine pathway annotation patterns that appear in all graphs in the database. Traditional general-purpose graph mining algorithms do not return any patterns as there are no graph patterns that explicitly appear in Pathway1 and Pathway2 at the same time. However, taking advantage of the associated GO taxonomy in Figure 1.1, and using the generalization-specialization relationships between functionality concepts, it is possible to discover the implicitly-occurring patterns of P1 and P2 in Figure 1.3.

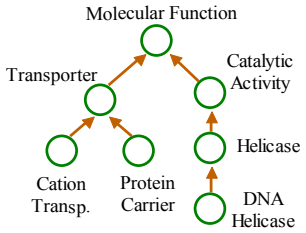

Figure 1.1: A subgraph of GO

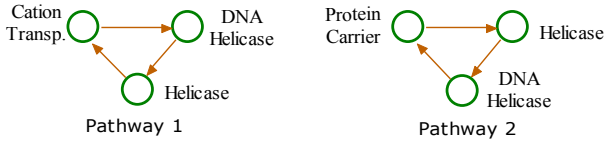

Figure 1.2: A Pathway Annotation Graph Database

Figure 1.3 Sample Patterns

There are two major challenges that are unique to taxonomy-superimposed graph mining in comparison with traditional graph

EDBT'08, March 25-30, 2008, Nantes, France.

mining. First, the number of patterns that can be extracted using a taxonomy of labels is enormous as any node in a pattern P can be replaced by any of its ancestors in the associated taxonomy to create a *generalized pattern* of P. Thus, efficiently enumerating possible candidates and calculating their support values is challenging. Second, not all extracted patterns are useful. For instance, the pattern P2 is *over-generalized* with respect to P1, since (a) it is a generalization (i.e., a generalized pattern) of P1, and (b) P1 appears in all the graphs that P2 appears in the database. Hence, efficiently detecting and eliminating over-generalized patterns is the second challenge.

Traditional graph mining has two major steps: (i) enumerating candidates, and (ii) computing the occurrence count (i.e., *support*) of each candidate in the database. Support computation consumes the majority of the processing time [19], mostly due to the fact that this step involves solving an NP-hard problem, namely, the subgraph isomorphism test. Taxonomy-superimposed graph mining involves *generalized subgraph isomorphism test* which extends the traditional subgraph isomorphism: given two graphs G_1 and G_2 to be tested for isomorphism, a node in G_1 with label *l* may match to any node of G_2 labeled by either *l* or any ancestor of *l,* as defined in the associated taxonomy. It can easily be shown by reduction that the generalized subgraph isomorphism problem is at least as hard as the subgraph isomorphism problem. Furthermore, pattern generalization and specialization based on a taxonomy make things worse computationally.

Many successful techniques have been developed for general-purpose graph mining. A desirable solution to taxonomy-superimposed graph mining should take advantage of the existing candidate enumeration techniques, and over-generalized patterns should be eliminated with low computational effort.

An obvious approach to the problem, which we call the *baseline approach*, is to directly use the existing general-purpose graph mining techniques by replacing the traditional subgraph isomorphism test with the generalized subgraph isomorphism test. In this baseline approach, first, all possible patterns are computed, and then over-generalized patterns are eliminated via a post-processing step. The downside of this approach is that, during the mining procedure, considerable amount of processing time is spent for computing the support values of over-generalized patterns that are then not included in the final pattern set.

Alternatively, a possible *bottom-up approach* would be to detect and eliminate over-generalized patterns at early stages of candidate generation so that such patterns would not be propagated to later iterations as seeds to generate larger patterns [9]. A major drawback of this approach is that, for a pattern and for each one of its generalized patterns that is not over-generalized, the support computation is performed independently resulting in high-time complexity. More specifically, since a pattern P and its generalized versions share common occurrences in a graph database, the same occurrence of P is counted more than once. And, each counting involves a separate subgraph isomorphism test. In fact, for a pattern P with *n* nodes, and a taxonomy T, the same occurrence of P in a taxonomy-superimposed graph is counted as many times as the number of generalized patterns of P, which is $O(d^n)$, where *d* is the average number of ancestors for the vertex labels of P in taxonomy T. Even though the bottom-up approach is more efficient than the baseline approach [9], it eventually suffers from the enormous increase in the total number of subgraph isomorphism tests during support computation. We give an example.

Example 1.2. Figure 1.4 shows a small database D = {G1, G2, G3} to be mined for frequent graph structures, where vertex labels are from the sample taxonomy of Figure 2.1. Figure 1.5 shows two non-over-generalized graph patterns in D along with their support values. Pattern P1.1 appears in all three graphs in D and has 4 implicit occurrences (1.1, 2.1, 2.2, 3.1) which are marked with dashed borders on the graph accompanied with occurrence numbers in the form of (*graph# .occurrence#)*. Note that two of these occurrences (1.1, 2.1) are also shared by the pattern P1.2. However, in a level-wise bottom-up approach, since the patterns are processed independently, in the support counting stage, 6 generalized subgraph isomorphism tests (namely, between P1.1 and each of its occurrences labeled with 1.1, 2.1, 2.2, 3.1, and between P1.2 and each of its occurrences labeled with 1.1 and 2.1) are performed although 4 would be sufficient (as 2 of the occurrences, namely, 1.1 and 2.1, are shared among the patterns). Hence, taking advantage of shared occurrences does provide performance gains for taxonomy-superimposed graph mining.

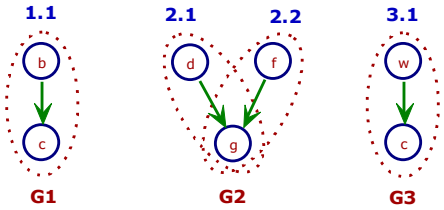

Figure 1.4: A graph database D

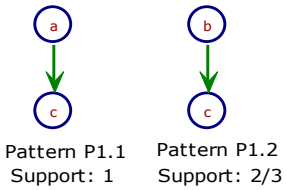

Figure 1.5: Size-2 patterns in D

In this paper, we propose and evaluate the Taxogram algorithm, a top-down support computation approach, which (i) takes advantage of the existing efficient techniques developed for general-purpose graph mining, and (ii) aims to perform the subgraph isomorphism test on each occurrence of a pattern P only once, and contribute the result of that isomorphism test to the support computation of not only P, but also all generalized patterns of P. The Taxogram algorithm consists of three stages. In the first step, vertices in the input graph database are relabeled with the most general ancestor of their label in the associated label taxonomy. The second step extends general-purpose graph mining approaches with taxonomy-projected "occurrence indices", and performs traditional graph mining on the relabeled database to mine classes of patterns. Finally, in the last step, new members are enumerated for each pattern class, and their support values are computed using the occurrence indices.

Contributions of this paper are as follows:

- The traditional graph mining is reconciled with taxonomy-superimposed graph mining via relabeling the input database, and mining for pattern classes using the existing approaches, before directly mining for actual patterns.

- Taxonomy-projected pattern occurrence indices are developed to capture the shared occurrences of patterns so that a single isomorphism test is performed per occurrence rather than per occurrence-pattern pair, and the result of the isomorphism test is shared by multiple patterns.
- An efficient specialized pattern enumeration algorithm is developed by taking advantage of hierarchically organized occurrence indices.
- Over-generalized patterns are eliminated without requiring expensive pairwise subgraph isomorphism tests among the extracted patterns.
- The proposed approach is experimentally evaluated in terms of its performance for different datasets.

This paper is organized as follows. In Section 2, we formally define the taxonomy-superimposed graph mining problem, and list its properties. In Section 3, we present the Taxogram algorithm, and its efficiency enhancements. Section 4 discusses the experimental setup and results. Section 5 gives an overview of the related work, and Section 6 concludes.

2. PROBLEM DEFINITION

Def'n *(Labeled Graph)*: *A labeled graph* $G(V_G, E_G, L_G, \partial_G)$ *consists of a set of edges* E_G*, and a set of vertices* V_G *where each vertex* υ *in* V_G *is assigned a label* l*,* $l \in L_G$*, by a labeling function* $\partial_G: V_G \rightarrow L_G$ *such that* $L_G = \{ \partial_G(\upsilon) \mid \forall \upsilon \in V_G \}$ *(i.e.,* $\partial_G()$ *is total).*

Optionally, edges of a graph may also be labeled, but to keep our definitions simple, we omit edge labels in our definitions without loss of generality.

Def'n *(Taxonomy)*: *Taxonomy* $T(V_T, E_T, L_T, \partial_T)$ *is a labeled directed acyclic graph where (a) an edge from vertex u to vertex v represents an is-a relationship such that v is an ancestor of u, and u is a descendant of v, and (b)* $\partial_T()$ *is one-to-one and onto.*

A taxonomy $T(V_T, E_T, L_T, \partial_T)$ defines a specialization-generalization hierarchy for the set of labels L_T. Let *Anc(l)* represent the set of all ancestors of *l*, and *Desc(l)* represent the set of all descendants of *l* in T. Then, the following properties hold for *ancestor*/*descendant* relationships in T.

- If *u* is an ancestor (descendant) of *v*, and *v* is an ancestor (descendant) of *w* then *u* is an ancestor (descendant) of *w*. (*Transitivity*)
- $\forall l \in L_T$, *l* is an ancestor of itself.

Example 2.1. Figure 2.1 depicts a sample taxonomy T, and Figure 2.2 lists a set of labeled graphs.

From now on, the phrase "graph G over taxonomy T" means that $L_G \subseteq L_T$. Similarly, the phrase "graph database D over taxonomy T" means that, for each graph G in D, $L_G \subseteq L_T$.

Def'n *(Generalized Graph Isomorphism)*: Given labeled graphs $G_1(V_1, E_1, L_1, \partial_1)$ and $G_2(V_2, E_2, L_2, \partial_2)$ over taxonomy T, G_1 is *generalized isomorphic* to G_2, denoted as $G_1 \langle IS\_GEN\_ISO \rangle G_2$, if there exists a one-to-one and onto mapping function $\delta: V_1 \rightarrow V_2$ such that (i) $\forall \upsilon \in V_1$, $\partial_1(\upsilon) = \partial_2(\delta(\upsilon))$ or $\partial_1(\upsilon) \in Anc(\partial_2(\delta(\upsilon)))$, and (ii) $\forall (\upsilon_i, \upsilon_j) \in E_1$, $(\delta(\upsilon_i), \delta(\upsilon_j)) \in E_2$. Furthermore, G_1 is called a *generalized* graph of G_2. Conversely, G_2 is called a *specialized* graph of G_1.

Example 2.2. In Figure 2.2, G_C is generalized isomorphic to G_A. Also note that G_B is not generalized isomorphic to G_A.

Remark 2.1. <IS_GEN_ISO> (a) is not commutative, (b) transitive, and (b) does not distribute over $\cup$, but it does over $\cap$.

Def'n *(Generalized Subgraph Isomorphism)*: Given labeled graphs $G_S(V_S, E_S, L_S, \partial_S)$ and $G(V, E, L, \partial)$ over taxonomy T, G is *generalized subgraph isomorphic* to G_S if there exists a subgraph G_S' of G_S such that $G \langle IS\_GEN\_ISO \rangle G_S'$.

Example 2.3. In Figure 2.2, G_B is generalized subgraph isomorphic to G_A.

Remark 2.2. Generalized subgraph isomorphism is (a) not commutative, (b) transitive, and (c) does not distribute over $\cup$, but it does over $\cap$.

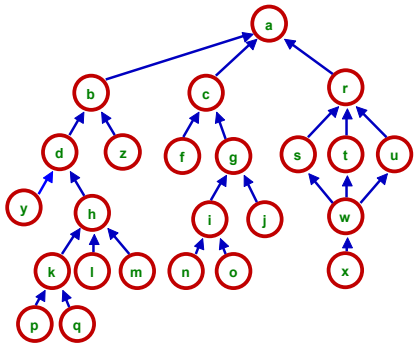

Figure 2.1: A sample taxonomy T

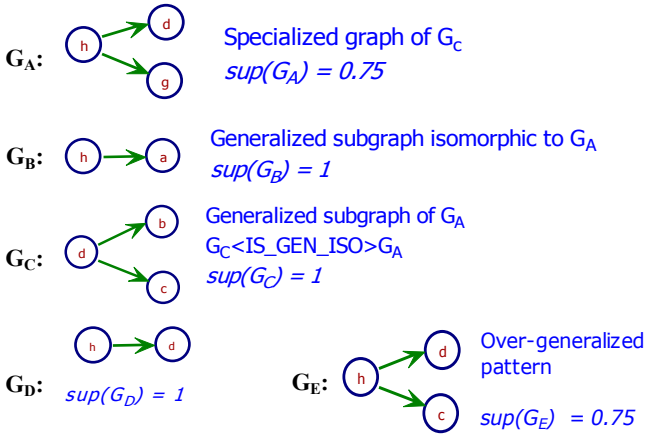

Figure 2.2: Some graphs with their support in DB of Fig. 2.3

Def'n *(Support)*: Given a graph database D over taxonomy T, and a graph G in D, let *GenSet*(G) = $\{G_i \mid G_i \in D$ and G is generalized subgraph isomorphic to $G_i\}$. Then, the *support* of G, denoted as *sup(G),* in D with respect to T is *sup(G)* = |*GenSet* (G)|/|D|.

Note that the support definition does not count the actual occurrences of G in graphs of D, but the number of graphs in D where G occurs at least once.

Example 2.4. In the database D of Figure 2.3 over taxonomy T of Figure 2.1, the support of G_A in Figure 2.1 is 0.75 (G_A is a generalization of the dark-colored subgraphs in G_1, G_3, and G_4).

Def'n *(Pattern)*: Given a graph database D over taxonomy T, and a support threshold ε, $0 < \varepsilon \leq 1$, a labeled connected graph G is called a *pattern* if *sup(G)* in D is at least ε, and G contains at least one edge.

A pattern together with all of its generalized and specialized graphs as patterns represents a *pattern class*.

Def'n *(Pattern Class)*: Given a pattern P over taxonomy T, P's *pattern class* contains all possible patterns that can be obtained by relabeling any node υ in P with any ancestor or descendant of υ's label in T.

Example 2.5. G_A and G_C (Figure 2.2) are in the same pattern class. Similarly, G_B and G_D are in the same pattern class.

Def'n *(Over-generalized Pattern)*: A graph pattern P_O is an *over-generalized pattern* if there exists another pattern P_S such that (i) $P_O \langle IS\_GEN\_ISO \rangle P_S$, (ii) $sup(P_O) = sup(P_S)$, and (iii) $P_O \neq P_S$.

Example 2.6. In Figure 2.2, graph pattern G_B with support 1 is over-generalized as there is a more *specialized* pattern G_D with the same support within the graph database D of Figure 2.3.

Def'n *(Taxonomy-Superimposed Graph Mining Problem)*: Given (i) a taxonomy T, (ii) a graph database D over T, and (iii) a support threshold ε, *the taxonomy-superimposed graph mining problem* is to locate the set H of graph patterns such that (a) $\forall P_i \in H$, $sup(P_i) \geq \varepsilon$, (b) H does not contain any over-generalized patterns (*minimality*), and (c) H contains all non-over-generalized patterns in D with support ≥ ε (*completeness*).

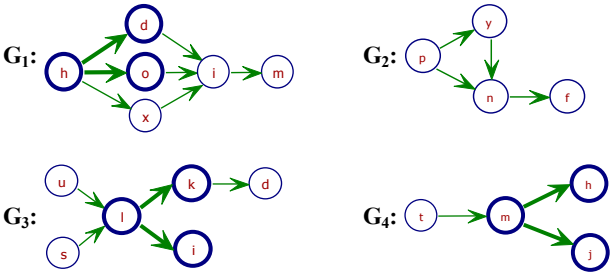

Figure 2.3: A Graph Database

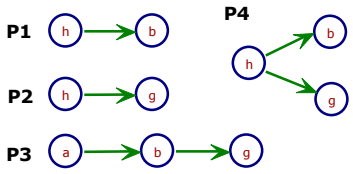

Figure 2.4: Generalized Graph Patterns with Support = 1

Example 2.7. Given the database D in Figure 2.3, let the support threshold ε = 1. Figure 2.4 shows all generalized patterns which have support 1 in D, minimal (i.e., no over-generalized patterns), and complete (i.e., has all non-over-generalized patterns).

2.1 Properties of Taxonomy-Superimposed Graph Mining

In this section, we present three properties that are unique to taxonomy-superimposed graph mining. Proofs are omitted for the lack of space.

Lemma 1. *Given a pattern P over taxonomy T, let n be the number of nodes in P. Then, the number of generalized patterns of P is $O(d^n)$ in the worst case, where d is the average number of ancestors of P's vertex labels in T.*

Lemma 2. *Given a pattern P, and the support set SS(P) of all graphs G in a database over taxonomy T, where P is generalized isomorphic to G, for any generalized pattern P_g of P, $SS(P) \subseteq SS(P_g)$.*

Therefore, a pattern P is not frequent (i.e., $sup(P) < \varepsilon$) if one of its generalized patterns is found to be infrequent.

Lemma 3. *If P is an over-generalized pattern, there may exist a generalized pattern P_g of P which is not over-generalized.*

Example 2.8. Consider the pattern G_E in Figure 2.2. G_E has a support of 0.75 in the graph database of Figure 2.3, and it is over-generalized as G_A (Figure 2.2) is a specialized pattern of G_E with the same support. However, G_E has a generalized pattern G_C (Figure 2.2) which is not over-generalized as its support, 1, is higher than its specialized patterns.

In other words, downward closure property does not hold on the axis of generalization/specialization of patterns. Therefore, locating an over-generalized pattern P_g is not sufficient to prune out all of P_g's generalized patterns from the search space.

3. TAXOGRAM ALGORITHM

In this section, we present the details of the Taxogram algorithm. Given a graph database D over taxonomy T, and a support threshold ε, we propose a three-step mining procedure. First, all vertex labels in graphs of D are replaced by their most general ancestors in T to create a "most-generalized" database D^{mg} (while also retaining the original labels of vertices). Second, we employ existing efficient candidate enumeration and support counting techniques developed for general-purpose graph mining to mine all frequent graphs in D^{mg}. Third, discovered patterns are post-processed to derive their specialized versions while eliminating over-generalized patterns so that the final pattern set is complete and minimal.

The three-step mining procedure has four features:

i. By relabeling vertices with the most general ancestor of their labels, each class of patterns is collapsed into a single pattern which is the most general pattern of that class. Hence, the total number of patterns to be extracted at the initial stage decreases, which in turn decreases the total number of database accesses and isomorphism tests.

ii. We employ one of the existing general-purpose graph mining techniques once at the beginning of second step of the framework, reducing the number of graph isomorphism tests.

iii. We reuse shared pattern occurrences among patterns of the same class. This reduces the number of costly graph isomorphism tests further.

iv. We use taxonomy-projected "occurrence indices" in the post-processing stage, to eliminate over-generalized patterns---without requiring an isomorphism test between each pair of created patterns, which further reduces the total number of graph isomorphism tests to be performed.

Next, we discuss each step of the algorithm in detail.

Step 1. Relabeling Input Graph Database

In the first step of the Taxogram algorithm, for each vertex label *l* in the graph database D, the most general ancestor l_g of *l* is located in label taxonomy T. Next, all vertices labeled with *l* are relabeled with l_g. Each vertex also internally stores its original label *l* to be used in later stages. We give an example.

Example 3.1. Consider the graph database D in Figure 1.4. After relabeling vertices in D with the most general ancestor of their original labels, the modified database D^{mg} is shown in Figure 3.1. Note that the original labels of vertices are kept (shown in parenthesis) to be used in later stages.

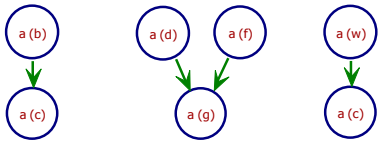

Figure 3.1: Graph database D^{mg} after relabeling D

This step is not redundant as the associated taxonomy may have multiple roots. When the taxonomy T has multiple roots and the roots have a common child node, in some cases, a label l may have a set $Ancs(l)$ of most general ancestors. In such cases, an artificial node with a unique label l_r is introduced as the common ancestor of nodes in $Ancs(l)$.

Time & Space Complexity of Step 1: This step processes all graphs in the database once to relabel them. Therefore, time complexity of this step, in the worst case, is $O(|D||G_{max}|)$ where $|D|$ is the number of graphs in the database, and $|G_{max}|$ is the size of the largest graph. Since the original labels are also retained during the relabeling process, the space requirement for extra label storage is also $O(|D||G_{max}|)$.

Step 2. Mining for Pattern Classes

In the second step, *extended* general-purpose graph mining is performed on the relabeled graph database D^{mg}. The output of this step is the most general patterns in each pattern class of the final pattern set. There are several efficient general-purpose graph mining approaches, e.g., FSG [11], gSpan [22], FFSM [6], etc. Any of these approaches can be extended for mining pattern classes on relabeled input graph. In this study, we choose to extend gSpan because its depth-first-search style candidate enumeration requires less memory, and its running time performance is better than or at least comparable to the other alternatives [19].

Remark 3.1. [21] *gSpan has time complexity of O(kFS + rF) where k is the number of occurrences of a frequent subgraph in a graph in the database, F is the number of frequent subgraphs, S is the database size, and r is the maximum number of duplicate codes (canonical representation scheme of gSpan) of a frequent subgraph that grow from other minimum codes.*

Our extension involves the creation of a (*taxonomy-projected*) *occurrence index* (OI) for each produced pattern. Given a pattern P, all of its occurrences in D^{mg} are stored together with the original labels of the vertices. For efficient enumeration of specialized patterns in the next step, occurrences of P are stored in subtaxonomies (subgraphs) of taxonomy T. A subgraph of T is created for each node in a pattern. Each occurrence is numbered in the form of (*graph#*.*occurrence#)*. Lastly, the set of occurrences for the whole pattern is also stored in an ordered list. We give an example.

Example 3.2. Consider the relabeled graph database D^{mg} in Figure 3.1, and let the support threshold be 2/3. In this step, traditional general-purpose graph mining (i.e., gSpan in this paper) extended with taxonomy-projected occurrence index creation is applied on database D^{mg}, which results in two patterns, P1 and P2, shown in Figure 3.2 together with their occurrence indexes (OI). Occurrence indexes are shown in dashed-bordered-regions. Each occurrence is represented by its id.

Def'n *(Occurrence of pattern)*: Given a graph G and a pattern P in the database, a subgraph G_S of G is called an occurrence of P in G if $P\langle IS\_GEN\_ISO\rangle G_S$.

One may argue that replacing node labels with their most general ancestor may dramatically increase the number of graph isomorphism tests. However, as we emphasize in Example 1.2, for completeness, the most general pattern always needs to be considered for support computation, even if we do not relabel the nodes. Hence, this is not an aspect that is caused by relabeling of nodes.

Def'n (*Taxonomy label covered by pattern*): Pattern P *covers* label *a* of taxonomy T when there exists an occurrence G(V, E, L, ∂) of pattern P in database D with $a \in \partial(v)$, $v \in V$.

Example 3.3. In database D of Figure 1.5, the label *c* of taxonomy T (Figure 2.1) is covered by pattern P1 of Figure 3.2 since *c* is in the graph occurrence G3.1 (in Figure 1.5) of P1.

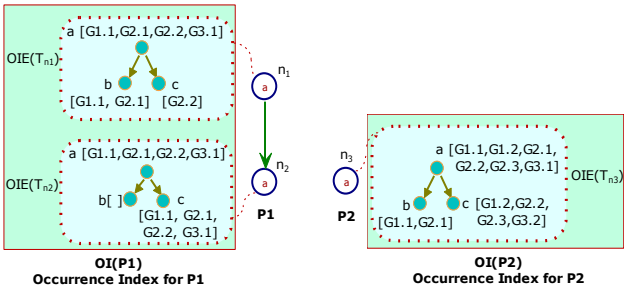

Figure 3.2: Patterns from D^{mg} with their occurrence indexes projected on a subgraph of T

Def'n *(Occurrence index of pattern, occurrence index entry)*: Given a pattern P (V, E, L, ∂) and an input database D over taxonomy T, the *occurrence index* OI(P) of P is a set $S(T_i)$ of taxonomies such that $S(T_i) = \{ T_i \mid T_i$ is a subtaxonomy of T, $root(T_i)=root(T)$, $1 \le i \le |V|$, T_i contains only those labels of T covered by pattern P$\}$. T_i is called an *occurrence index entry* (*OIE)*, and is assigned to node $v_i \in V$ of pattern P through a mapping function ϕ: $v_i \in V \rightarrow S(T_i)$, i.e., $\phi(v_i) = T_i$.

Example 3.4. In Figure 3.2, each of the patterns P1 and P2 has one occurrence index, namely, OI(P1) and OI(P2). Each node of each pattern is associated with one *Occurrence Index Entry* (OIE), which is a subtaxonomy of T. For instance, pattern P1 consists of single directed edge (a, a), and OI(P1), the occurrence index of P1, has two occurrence index entries, namely, OIE(top node of P1 with label a), and OIE(bottom node of P1 with label a).

Def'n (*OIE node label and occurrence set*): Each node of OIE T_i is (a) labeled with a taxonomy label, say *l*, covered by P, and (b) assigned an occurrence set OcS(*l*) containing the id's of occurrences of P, identified by *graph#.occurrence#* in database D.

Example 3.5. In Figure 3.2, consider the top OIE T_1 of OI(P1). Each node label *l* in T1 is a taxonomy label "covered" by P. As an example, label *c* is covered by P and the occurrence set OcS(*c*) of *c* is {G2.2} where G2.2 is illustrated in Figure 1.5.

Note that, in the above example, occurrence set for taxonomy label *a* in P1's occurrence index is different than the taxonomy label *a* in P2's occurrence index, mainly, due to the fact that the

P1 and P2 are two distinct pattern classes and their occurrences in the input database are not identical.

Space Complexity for Step 2: In the worst case, for each pattern node n, the associated occurrence index OI(n) contains as many nodes as the number of labels in taxonomy T. Since the traditional graph mining algorithm, gSpan, that we employ in this step enumerates patterns in depth-first manner, at any moment in step 2, there exists only one pattern and its associated occurrence index in the memory. Thus, regardless of the number of patterns in the final produced pattern set, in the worst case, the space requirement for occurrence indexes equals to $|P_{max}|\cdot|T|$ where $|P_{max}|$ is the number of nodes in the largest pattern P_{max} among all patterns, and |T| is the number of labels (concepts) in the label taxonomy T. $|P_{max}|$ is bounded by $|G_{max}|$ where G_{max} is the largest graph in the input database D. Hence, space complexity for occurrence indexes is $O(G_{max}* |T|)$.

Given a graph G in the input database D over taxonomy T, and a pattern P, the maximum number of occurrences of P in G is $\frac{|G|!}{(|G|-|P|)!}$. Hence, the size of occurrence set for each taxonomy label *i* in an occurrence index is $\sum_{G\in D}\frac{|G|!}{(|G|-|P|)!}$. Since, in the worst case there are |T| labels per occurrence index node, and there is one occurrence index node per pattern node, the space requirement for occurrence sets is $|P|\cdot|T|\cdot\sum_{G\in D}\frac{|G|!}{(|G|-|P|)!}$.

Lemma 4. *The space complexity for the step 2 of the Taxogram algorithm is* $O(|P|\cdot|T|\cdot\sum_{G\in D}\frac{|G|!}{(|G|-|P|)!})$ *where |P| is the number of nodes in the largest pattern, and |T| is the number of labels in the associated taxonomy.*

In order to minimize storage requirements, and allow for efficient set intersection in the next step, Taxogram implements occurrence sets as bit sets where each occurrence id is mapped to an integer value, and if an occurrence set contains a particular occurrence id, the corresponding bit is set to 1, and it is set to 0, otherwise.

Time complexity for step 2: The time complexity for mining pattern classes is bounded by the complexity of traditional pattern mining algorithms. More specifically, gSpan has time complexity of O(kFS + rF) (Remark 1). During occurrence index construction, for each occurrence O_i of a pattern P, occurrence sets of pattern node labels in each occurrence index node are updated by adding *id* of O_i. Each update operation per label *l* in an occurrence index node also involves updating the occurrence sets of ancestor labels of *l*. In the worst case, average number of ancestors per label in a taxonomy is $(|T|-1)/2$. Then, for each occurrence of P, $|P|\cdot(|T|-1)/2$ updates are performed. Since the maximum number for a pattern P is $\sum_{G\in D}\frac{|G|!}{(|G|-|P|)!}$), the number of required update operations during the occurrence index construction for a pattern in step 2 is $(|P|\cdot(|T|-1)/2)\cdot\sum_{G\in D}\frac{|G|!}{(|G|-|P|)!}$.

Lemma 5. *The time complexity for the step 2 of the Taxogram algorithm is* $O(N\cdot(|P|\cdot(|T|-1)2)\cdot\sum_{G\in D}\frac{|G|!}{(|G|-|P|)!})$ *where N is the number of patterns, |P| is the number of nodes in the largest pattern, and |T| is the number of labels in the associated taxonomy.*

Given a pattern P, a pattern node n ∈ P, and the occurrence index node OI(n) associated with n, in order to minimize the space requirements further, we take two actions: (i) OI(n) contains only the labels (and their ancestors) that appear in at least one occurrence of P in the position of node n, and (ii) labels that do not appear in at least ε·|D| distinct graphs in D, where |D| is the number of graphs in D, are not considered during the construction of OI(n).

Next, we show that the extended approach produces the same pattern class counts with taxonomy-superimposed graph mining.

Lemma 6. *Given a graph database D over taxonomy T, and the relabeled copy* D^{mg}*, and support threshold ε, 0 ≤ ε ≤ 1, let* C^{mg} *be the set of all pattern classes that have at least one member in the pattern set* H^{mg} *discovered by performing a general-purpose graph mining on* D^{mg}*, and let C be the set of distinct pattern classes that have at least one member in the pattern set H produced by applying taxonomy-superimposed graph mining on D. Then,* $C=C^{mg}$.

Thus, the pattern class counts in D and D^{mg} are the same.

Step 3. Enumerating Specialized Patterns from Pattern Classes

This step has two main goals: (i) enumerate specialized patterns from pattern classes produced in the previous step, and (ii) eliminate/prevent over-generalized patterns from the final pattern set. Step (i) together with Lemma 6 guarantees completeness. In order to carry out these two tasks in an efficient manner, we utilize taxonomy-projected occurrence indices.

Definition *(Occurrence Set of a Pattern):* Given a pattern P(V, E, L, ∂) which is a member of a pattern class PC with the occurrence index OI, occurrence set OcS(P) of P is the intersection of occurrence sets corresponding to each node label in OI(P), that is, $OcS(P) = \{\cap_{OcS(l)} \mid l \in T_i, T_i = \phi(\upsilon_i), \text{and } l = \partial_1(\upsilon_i) \text{ where } \upsilon_i \in V\}$.

Given a pattern P accompanied with its occurrence index OI, each node υ_i of the pattern is attempted to be replaced by one of its children from the corresponding subtaxonomy T_i in OI, where $\phi(\upsilon_i) = T_i$. Each replacement leads to a new specialized pattern P_s of P. Using the associated occurrence sets of each taxonomy label in the occurrence index, the support of P_s is obtained by computing the intersection of occurrence set of P with that of the taxonomy label which replaces its parent and leads to creation of P_s. We give an example.

Example 3.6. Consider the general pattern P1 (representing a pattern class) and its occurrence index OI which is derived in the previous step (Figure 3.2). Figure 3.3 shows creation of a specialized pattern P2 of P1 by replacing label "a" of node n_1 with its child label "b" from $\phi(n_1) = T_{n1}$ in OI. The occurrence set OcS(P2) of newly created pattern P2 is obtained by computing intersection of the occurrence set for pattern P1 with that of taxonomy label "b" in occurrence index node T_{n1}, that is, $OcS(P2) = OcS(P1) \cap OcS(T_{n1}(b)) = \{G1.1, G2.1, G2.2, G3\} \cap \{G1.1, G2.1\} = \{G1.1, G2.1\}$. Then, computing the support *sup*(P2) of P2 is counting the distinct graph ids in OcS(P2) (i.e., 2), and dividing it by the database size (i.e., 3), which results in *sup*(P2) = 2/3.

Lemma 7. *Given a pattern P, its occurrence index OI, and specialized pattern* P_s *of P where* P_s *is created by replacing label*

of node $n_i \in P$ with one of its child labels l_c from the corresponding OI node of $T_{ni} = \phi(n_i)$, then the occurrence set of P_s is $\text{OcS}(P_s) = \text{OcS}(P) \cap \text{OcS}(T_{ni}(l_c))$. *And, sup($P_s$) equals to number of distinct graph ids in* $\text{OcS}(P_s)$.

Thus, no database scan or isomorphism test is required during the support computation of more specialized patterns due to the use of the occurrence indices associated with pattern classes. If *sup*(P_s) equals to *sup*(P), then P is an over-generalized pattern, and P is eliminated from the final output set. The creation of more specialized patterns by replacement of a parent with one of its children from the occurrence index is performed on each newly created pattern until no specialized node with sufficient support can be created. Note, however, this procedure may lead to duplicate patterns due to different orders of node label replacements as shown in the following example.

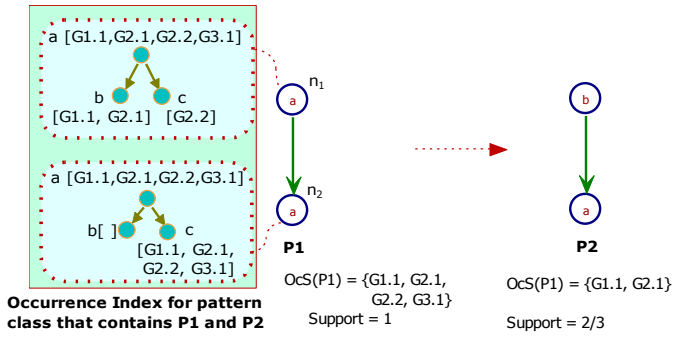

Figure 3.3 Support computation for specialized patterns

Example 3.7. Consider the pattern P1 and its occurrence index OI(P1) (Figure 3.3). Figure 3.4 shows a sample specialized pattern derivation procedure. For the sake of simplicity, we ignore support threshold considerations in this example (please ignore the sets associated with symbol "PNS" for this example). In the first step, label of node n_1 is replaced by its child label b to create pattern P2. Then, in the second step label of node n_2 is replaced by its child label "c", which results in pattern P3. P3 cannot lead to more specialized patterns as taxonomy labels "b" or "c" does not have a child in the occurrence index for this set of patterns. And, assume P2 also cannot be turned into other more specialized patterns. Next, we turn back to P1. In step 3, label of node n_2 in P1 is replaced with its child label "c" to create pattern P4. And, in step 4, label of node n_1 in P4 is replaced with its child label "b" to create pattern P5 which is the same as P3. Although P2 and P4 are different patterns, their specialization leads to the same pattern.

In order to eliminate such duplicates, at each enumeration branch, ids of pattern nodes whose labels have been replaced with one of their children labels are kept in a set called *processed-nodes set (PNS)* (see Figure 3.4 as an example where step 4 will not take place when PNS is employed as node n_1 is already included in PNS before step 4). Furthermore, within a single occurrence index, taxonomy labels may share a child n due to that DAG structure. In order to make sure that only one of n's parents utilizes n for a new specialized pattern creation, visited vertex labels within an occurrence index are also marked.

Employing PNS eliminates duplicates, but may lead to erroneous inclusion of some over-generalized patterns in the final pattern set. Such cases may occur if PNS is not empty when a pattern P is first created from a more generalized pattern (e.g., P4 in Figure 3.4), since the enumeration procedure stops before creating all possible specialized patterns of P, and there may be more specialized patterns of P that were created in other branches during enumeration, and have the same support as P. We give an example.

Example 3.8. Consider the specialized pattern enumeration in Figure 3.4. Suppose P4 and P5 have the same support. Since P5 will not be created when PNS is checked before taking step 4, we may erroneously conclude that P4 is a valid pattern as no specialized pattern was derived from P4.

In order to prevent such *hidden over-generalized* patterns produced as a side effect of employing PNSs, occurrence set for the labels of each node in PNS is inspected to see if the replacement of any processed node's label with one of its children label would lead to a more specialized pattern with the same support. If no such node exists in PNS, then P is added into the final pattern set.

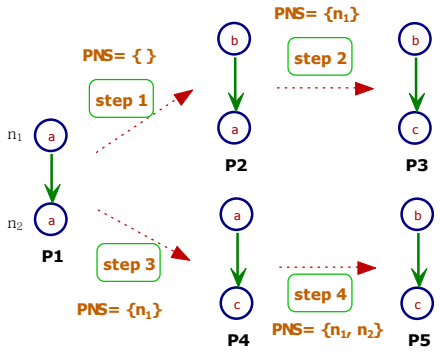

Figure 3.4 Creation of Duplicates during Pattern Enumeration

Since the Taxogram algorithm initiates the creation of a new pattern P_i from P through the replacement of any node label with any of its children, and compares the support of P_i against that of P, it is guaranteed that it will detect any possible case where *sup*(P_i) = *sup*(P), and mark such patterns as over-generalized. If P is over-generalized, then there must be a node $\upsilon_i \in P$ with label l such that replacing l with a child label l_c of l should lead to a new pattern P_s with the same support as P. Then there are two possible cases: (i) If $\upsilon_i \notin$PNS, then Taxogram will create a more specialized pattern P_s of P by replacing l with l_c for node υ_i and P will be marked as over-generalized as illustrated in . (ii) If $\upsilon_i \in$PNS, then a follow-up procedure (described above) attempts to replace label of each node in PNS with their children labels, and computes support values after each replacement to see if the new support value will be the same as that of P, in which case P is marked as over-generalized. Thus, we have

Lemma 8. *The final pattern set H produced by the Taxogram algorithm is minimal in the sense that H contains no over-generalized patterns.*

In addition, by Lemma 6, all possible pattern classes are included in H' which is the pattern set produced in the second step of the algorithm. Moreover, due to the relabeling of each node with its most general ancestor, each pattern class in H' is represented by the most general member of the class. Hence, all the other patterns can be obtained from the most general member of each class by performing node replacements with the descendants of the label assigned to each corresponding pattern node. In order for the algorithm to miss a pattern P, at least one descendant of a vertex's label in a most general pattern should be left out from

consideration during specialized pattern enumeration. However, the Taxogram algorithm considers replacements with all descendants which have enough support in the input database D. Finally, there is no need to consider replacements with descendants which have insufficient support in D since all subgraphs of a frequent graph pattern should be sufficiently frequent [10] (by Apriori property). Thus, we have

Lemma 9. *The final pattern set H produced by the Taxogram algorithm is complete in the sense that H contains all possible non-over-generalized patterns.*

Space & Time Complexity for Step 3: Enumerating specialized patterns from each pattern class, in the worst case, corresponds to replacing each node label with one of the labels in the associated label taxonomy, and computing support value after each label replacement. Since occurrence sets are implemented as Bit sets, cost of computing support of a newly created pattern P equals performing bitwise and operation between the occurrence bit set of the pattern that leads to P, and the occurrence set of the child label that replaced its parent label in P. The cost of bitwise "and" operation is bounded by the maximum number of occurrences for a pattern, which is computed as $\sum_{G \in D} Pr(|G|, |P|)$ in the previous step for an input database D and a pattern P.

Then, for a given pattern class $P(V_1, E_1, L_1, \partial_1)$, the number of specialized patterns is $\prod_{vi \in V1} |T| = |T|^{|V1|}$.

Additional Efficiency Enhancements and Pruning Methods: We employ additional techniques to enhance the performance of the Taxogram algorithm:

a. During specialized pattern enumeration, if replacement of a node label n with one of its child label c leads to a pattern with insufficient support, then there is no need to check patterns created via replacement of n with any descendant of c.

b. In order to minimize the size of occurrence indices, the taxonomy nodes whose labels are not frequent (i.e., when tested with generalized graph isomorphism for size-1 graphs) are removed from the taxonomy. Since a child cannot be frequent without having all of its parents frequent, the resulting taxonomy subgraph after the node removals is guaranteed to be a connected DAG.

c. Given a generalized pattern produced by the second step of the algorithm, before starting enumeration of specialized patterns in the third step, we pre-process the occurrence index of the pattern to see, for any node n in the pattern, if it is possible to replace n's label *l* with one of its descendants which has the same occurrence set as *l*. After performing all such replacements for every node, enumeration of specialized patterns is started on this modified set of patterns so that several over-generalized patterns can be eliminated before they are propagated to more specialized patterns created from the original pattern.

d. In taxonomy T, given a node *n* and one of its child node *c*, if *n*'s occurrence set is the same as *c*'s occurrence set, then n is removed from the taxonomy and a direct connection is created between c and each parent of n. This change preserves the minimality and completeness of the Taxogram, because any pattern P that has n as one of its node labels would be over-generalized since it is always possible to create a more specialized pattern of P by replacing c with n, and the support of the new pattern would be the same as P as the occurrence sets of c and n are the same.

4. EXPERIMENTS

In this section, we present an experimental evaluation of the Taxogram algorithm, and compare different approaches on both synthetic and real data. We first describe the experimental setting, and then, we discuss the experimental results and observations.

4.1 Experimental Setting

The experiments are performed on a PC with Pentium Dual Core (3.2 GHz) processor and 4 GB main memory. We have implemented Taxogram in Java 1.6, and compared its performance against the bottom-up approach (discussed in Section 1) which is represented by modified AcGM as proposed in [9], and called TAcGM in this paper. In addition, we have implemented a baseline algorithm which is the same as Taxogram except that the baseline algorithm does not utilize efficiency enhancements (e.g., removing taxonomy concepts with insufficient support) discussed at the end of section 3. The source code or executable files for TAcGM were not publicly available. Hence, we have implemented it ourselves in Java, and provided results from our implementation. We have built the implementation of Taxogram on gSpan implementation of ParMol package [19] which is publicly available.

Table 1. Properties of Experimental Data Sets

DB Id	DB Size (Graphs)	Avg. Graph Size (Node)	Avg. Graph Size (Edge)	Dist. Label Count	Avg. Edge Density
D1000	1000	9.3	10.9	5391	0.27
D2000	2000	9.4	10.9	7071	0.26
D3000	3000	9.4	11.1	7610	0.27
D4000	4000	9.4	11.1	7810	0.26
D5000	5000	9.4	11.0	7855	0.27
NC10	4000	6.3	6.1	7450	0.32
NC20	4000	9.2	10.7	7782	0.27
NC30	4000	12.3	15.9	7857	0.23
NC40	4000	15.4	21.2	7876	0.20
ED06	3000	14.1	6.5	7800	0.06
ED09	3000	13.0	8.6	7817	0.09
ED10	3000	12.9	9.2	7833	0.10
ED11	3000	12.9	10.3	7831	0.11
TD5	4000	15.1	20.9	1000	0.20
TD6	4000	15.0	20.6	1000	0.21
TD7	4000	15.2	21.0	1000	0.20
TD8	4000	15.3	21.2	1000	0.21
TD9	4000	15.2	21.1	1000	0.20
TD10	4000	15.3	21.1	1000	0.20
TD11	4000	15.4	21.3	1000	0.20
TD12	4000	15.0	20.7	1000	0.21
TD13	4000	15.2	20.9	1000	0.21
TD14	4000	15.0	20.6	1000	0.21
TD15	4000	15.1	20.8	1000	0.21
TS25	4000	15.3	21.1	25	0.21
TS50	4000	15.2	20.8	50	0.21
TS100	4000	15.0	20.7	100	0.21
TS200	4000	14.9	20.6	200	0.21
TS400	4000	15.1	20.9	400	0.21
TS800	4000	15.1	21.0	800	0.21
TS1600	4000	15.2	21.0	1600	0.21
TS3200	4000	15.3	21.1	3200	0.20
PTE	416	22.6	23.0	24	0.12

The experiments are performed on both synthetic and real data. We developed a synthetic graph generator, and a synthetic taxonomy generator, which provide the capability to create datasets with different characteristics. The synthetic graph generator expects a *label taxonomy*, *maximum node* and *edge*

counts for graphs. The edges are created based on an *edge density* parameter. We have employed same definition of edge density that was also used by Worlein et al. [19] for the evaluation of different data mining algorithms, where edge density is defined as 2* #edges/(#nodes)^2. The synthetic taxonomy generator expects *taxonomy size* which is characterized by both the number of concepts and relationships among concepts, *taxonomy depth* which defines the number of levels in the taxonomy. Table 1 summarizes the properties of data sets used in the experiments. Taxogram can handle both directed and undirected graphs, but since the current implementation is built upon gSpan's implementation and gSpan does not support directed graphs, all the experimental data sets consists of undirected graphs.

As real data, we have carried out evaluation on two distinct data sets from chemistry and biology domains. The first data set comes from the metabolic pathways [13] domain where nodes are functional annotations of enzymes catalyzing each reaction in a pathway, and the edges are created through shared substrate/products between the corresponding enzymes for each node. The node labels which are functional annotations come from the molecular function subontology of Gene Ontology, which contains over 7,800 concepts organized into a 14-level hierarchy. Figure 1.1 illustrates a small excerpt from the molecular function subontology of Gene Ontology. The pathways data involves 25 metabolic pathways from 30 prokaryotic organisms (downloaded in May 2007 from KEGG [13].

The second real data set is the molecular structure of carcinogenic compounds where atoms constitute nodes, and the bonds among atoms are edges between nodes. This data was made available within US National Toxicology Program (NTP) for The Predictive Toxicology Challenge (PTC) [14]. This data set contains 416 molecular structures where atoms are organized hierarchically as illustrated in Figure 4.1 where leaf level letters are atom labels while the labels in upper levels are logical groupings of atoms based on their similarity. Small-case letters represent aromatic atoms while upper-case letters stand for non-aromatic atoms.

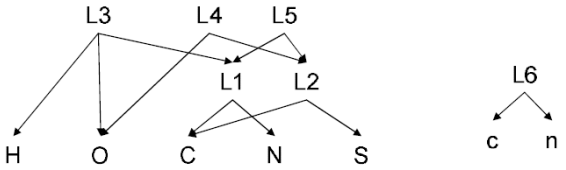

Figure 4.1 A taxonomy of atoms for PTE data

4.2 Results

Scalability against Database & Data Size:

In this section, we perform a number of experiments to test the scalability of Taxogram on data with different characteristics. As discussed in section 5.1, we generate synthetic data sets with varying database size (i.e., the number of graphs in a graph database). We also carry out measurements by changing properties of the graphs such as edge density, the average size of individual graphs. Then, we measure the running time on each data set and compare the results from Taxogram, TAcGM, and the baseline approach. We employ Gene Ontology molecular function subontology as the label taxonomy for the synthetic graphs.

In the first experiment of this section, we measure the performance of the algorithms against varying database size. To this end, we create five different synthetic databases with sizes ranging from 1,000 to 5,000 graphs. (See DBs whose ids start with "D" in Table 1). The support threshold is set to 0.2 (i.e., 20%), and the graph objects in the databases are allowed to have maximum of 20 edges. The number of distinct edge labels is set to 10. Figure 4.2 shows the running time performance of the three approaches, namely, baseline approach, TAcGM, and Taxogram.

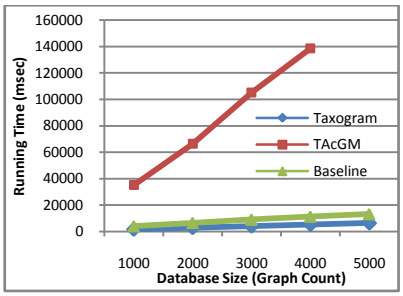

Figure 4.2 Running time performance for different DB size

Observation: *The running time of Taxogram stays almost constant with slight increase as the database and individual graph objects become larger.*

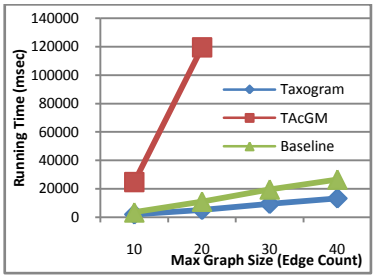

Figure 4.3 Performance Comparison while Changing Graph Size

As the database and graph sizes get larger, the number of occurrences that the graph miners need to deal with also increases. Since Taxogram first mines for pattern classes and keeps occurrences for the pattern classes rather than for individual patterns, the effect of increase in data and database size on Taxogram performance is considerably lower than that for the case of TAcGM.

Next, we study the scalability of Taxogram and the other approaches as the size of graph objects in a database increases. We keep the database size fixed as 4000, which is the maximum database size that TAcGM could return results without causing "out-of-memory" error in the previous experiment. The other parameters are also the same as the previous experiment (Support: 20%, distinct edge label count: 10, taxonomy: GO). Then, we create four different data sets by gradually increasing the max graph size (in terms of edge counts) in a database from 10 to 40. (See DBs whose ids start with "NC" in Table 1). Figure 4.3 presents the resulting running time performances.

Observation: *Taxogram is more scalable than TAcGM as TAcGM causes "out-of-memory" error when the maximum graph size becomes larger than 20 (Figure 4.3) or the database size is larger than 4,000 (Figure 4.2)*

Observation: *Consistent with the observed performance against changing the database size, the rate of running time increase for Taxogram is less than the rate of increase for TAcGM.*

Scalability is mostly related to the difference between pattern enumeration strategies of Taxogram and TAcgM. Since TAcGM generates patterns level by level, which is not memory efficient when the data gets somewhat larger, it uses the memory less efficiently, whereas Taxogram processes one pattern class at a time in depth-first-search manner.

Final experiment of this section aims to evaluate the performance of alternative approaches against varying edge density of the input database (See DBs whose ids start with "ED" in Table 1). Figure 4.4 shows the running time performance of the Taxogram and the number of produced patterns under different edge density values.

Observation: *Taxogram scales linearly against the edge density increase until the edge density value becomes greater than 0.10.*

Edge density has a direct impact on the number of possible pattern occurrences per graph. Therefore, high edge density significantly increases the size of occurrence indices and occurrence sets, and it takes more time to enumerate specialized patterns using such larger occurrence sets. In addition, as the graphs become highly connected the number of patterns also increases considerable leading to additional increase in running time.

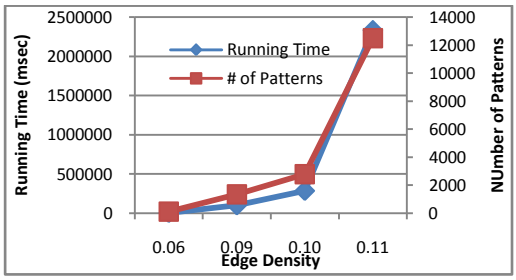

Figure 4.4 Running Time/Pattern Count at Different Edge Density Values

Scalability against Taxonomy Size:

In this experiment, we evaluate the performance of AcGM for different sizes of taxonomies. We consider taxonomy size along two distinct dimensions: (i) *Depth* and (ii) *Concept Count* of a taxonomy. In order to measure the effect of each dimension on the performance of Taxogram, we experiment with two synthetic sets of taxonomies. For the first data set, we keep the number of concepts fixed, and create different taxonomies with varying depths while the second data set consists of different taxonomies of the same depth but varying concept/relationship count. depicts running time performance and the number of discovered patterns of Taxogram for synthetic taxonomies with maximum depth ranging from 5 to 15. Each taxonomy is set to consist of 1000 concepts and 2000 relationships. Next, using each taxonomy, we created synthetic graph datasets of size of size 4000, and maximum graph size 40 with 10 distinct edge labels (See DBs whose ids start with "TD" in Table 1). Node labels for the database graphs are selected from each level of taxonomy with equal probability. does not present any results for TAcGM as it leads to "out-of-memory" error for all the datasets in this experiment.

Observation: *The running time of Taxogram stays almost constant for taxonomies where depth is less than 13.*

The main factor that influences the performance of Taxogram is the total number of patterns that can be extracted from a database. As the depth of a taxonomy reaches to higher values the number of patterns may increase dramatically. Hence, starting at max depth 13, as parallel to the number of patterns, the running time of Taxogram increases exponentially. Considering that most of the real life ontologies has less than 15 levels, (e.g., GO, the most popular annotation ontology in biology has 14 levels), the performance level of Taxogram is sufficiently practical to be used for real data.

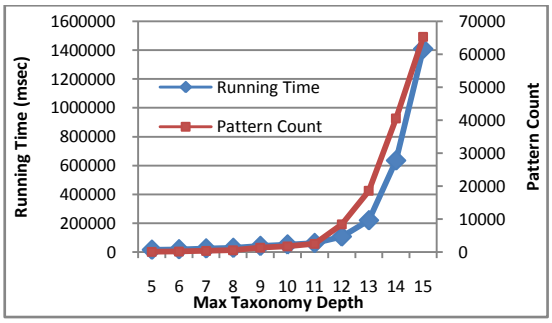

Figure 4.5 Performance for taxonomies of different depths

Finally, we study the effect of concept/relationship count of taxonomies on the performance. We create 8 different taxonomies where the size of the taxonomy (concept/relationship count) is doubled as each taxonomy is created. Similarly, for each taxonomy, we create synthetic graph datasets of size 4000, and maximum graph size of 40 with 10 distinct edge labels (See DBs whose ids start with "TS" in Table 1). Figure 4.6 illustrates the performance of Taxogram as the input taxonomy size changes. Once again, TAcGM does not run for the data sets used in this experiment. Hence, we only present the results for Taxogram.

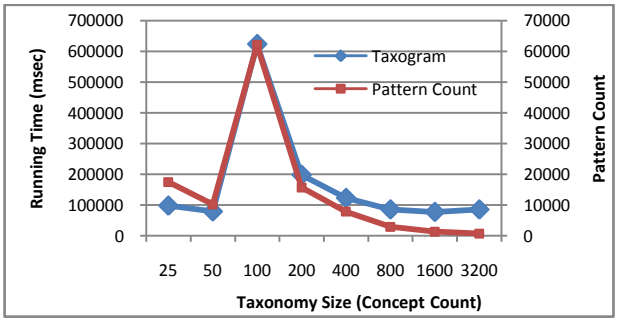

Figure 4.6 Performance for taxonomies of different size

Observation: *In general, running time decreases as the taxonomy size increases (while keeping the taxonomy depth fixed).*

This is mainly due to the fact that, for the synthetic datasets created based on these taxonomies, increase in concept count means that the number of distinct node labels increases which decreases the total number of patterns, hence, improves the running time.

A close inspection of pattern count (which completely depends on the input database) reveals the reasoning behind the unexpected peak in runtime at taxonomy size 100. The running time is closely related to the pattern set size which peaks for the data set where the taxonomy size is 100.

Effect of Support Threshold on Performance and the Number of Patterns:

Next, we assess the impact of support threshold on both the running time of Taxogram and the number of produced patterns. In order to get some results from TAcGM for comparison purposes, from the first experiment, we have selected the largest

data set (DBId: D4000 in Table 1) that TAcGM provides some results before leading to out-of-memory error. Similarly, Gene Ontology molecular function subontology is used as the taxonomy. Figure 4.7 presents the running time measurements for Taxogram and AcGM.

Observation: *Taxogram can handle lower support thresholds in comparison to TAcGM.*

Consistent with the previous experiments, Taxogram's running time increases linearly until the support threshold of 2 where the pattern set size increases dramatically. On the other hand, the running time of AcGM increases exponentially as the support threshold comes down below 30, and for the support thresholds less than 20, TAcGM does not complete as it leads to out-of-memory error. The reason is that, TAcGM joins size-k-1 frequent graphs to obtain size-k frequent graph candidates, and, at low thresholds, the number of patterns to be joined increases significantly.

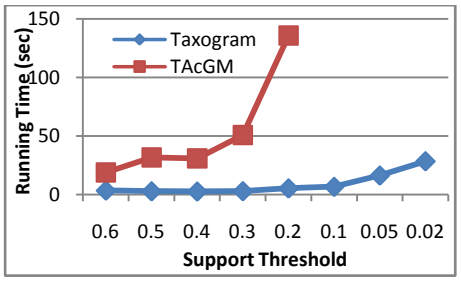

Figure 4.7 Taxogram vs TAcGM at Different Support Thresholds

Performance on Real Biological and Chemical Data Sets:

In this section, we carry out performance assessment on real data. First, we perform a small scale comparative genomics study, and employ Taxogram to mine for conserved pathway fragments among 30 prokaryotic (bacteria) organisms similar to our recent study in [2]. Given a pathway P, each organism has a different version of the pathway in terms of the structure as well as the genomic entities (i.e., proteins and genes) that involve in the catalysis of reactions. For 25 metabolic pathways, we collected the organism specific versions for 30 prokaryotic organisms from KEGG [13], constructed their pathway functionality templates [2], and run Taxogram to discover patterns that are common among organisms. The support threshold is set to 0.2. Table 2 shows the results for this experiment as well as the characteristics of each pathway data set.

The number of extracted patterns for each pathway can be used as a measure for the degree of conservation among organisms for that particular pathway. The higher the number of patterns, more conserved the pathway is through the lineage of the prokaryotic organisms.

Observation: *Nitrogen metabolism and Biosynthesis of Stereoids are the top most conserved pathways for bacterial organisms.*

Indeed, Nitrogen is a vital substance for almost all the organisms, but only bacteria can convert Nitrogen gas (N2) into the form that other living organisms can use [12]. Conservation of such a significant pathway is expected for the continuity of the life.

Observation: *The running time of Taxogram increases either in case of high conservation among organisms (e.g., beta-Alanine metabolism) or when the graph sizes gets larger in the input graph database (e.g., Pantothenate and CoA biosynthesis).*

This observation can be explained with the results presented in the scalability experiments section.

Table 2. Results on Pathways Data Set

Pathway Name	*Time (msec)*	*Pattern Count*	*Avg. Graph Size (Node)*	*Avg. Graph Size (Edge)*
Vitamin B6 metabolism	119	2	7.03	4.03
Inositol phosphate metabolism	140	7	4.33	3.33
Sulfur metabolism	156	7	5.17	3.23
Benzoate degradation via hydroxylation	206	60	7.60	5.30
Riboflavin metabolism	210	12	7.63	4.73
Nicotinate and nicotinamide metabolism	216	36	6.67	4.40
Thiamine metabolism	259	23	4.57	3.60
Lysine biosynthesis	314	61	8.73	7.67
Pentose and glucuronate interconversions	323	56	10.83	6.70
Synthesis and degradation of ketone bodies	353	31	4.97	4.10
Histidine metabolism	361	79	8.83	6.60
Tyrosine metabolism	529	57	7.93	6.13
Phenylalanine metabolism	613	32	5.80	4.40
Nucleotide sugars metabolism	693	106	7.57	6.30
Aminosugars metabolism	808	168	8.20	6.60
Citrate cycle (TCA cycle)	1011	174	10.80	8.63
Glyoxylate and dicarboxylate metabolism	1036	233	9.10	7.53
Selenoamino acid metabolism	1046	152	6.90	6.50
Valine, leucine and isoleucine biosynthesis	1069	75	5.23	4.70
Butanoate metabolism	1789	287	10.57	8.80
beta-Alanine metabolism	3562	661	5.10	5.60
Glycerolipid metabolism	6872	219	8.10	7.23
Biosynthesis of steroids	10609	830	7.97	8.87
Nitrogen metabolism	62777	1486	7.20	7.27
Pantothenate and CoA biosynthesis	215047	142	10.43	9.53

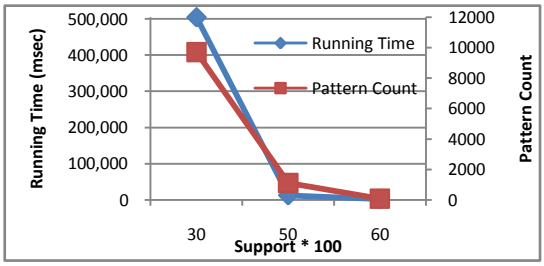

Figure 4.8 Performance on PTE Data

As a second real data set, we experiment with molecular structures of chemical data. This data set contains 416 graphs representing molecular structure of carcinogenic compounds [14]. Properties of this data set is listed in Table 1 (DBId: PTE) Figure 4.8 shows the performance on PTE data.

Observation: *Both the running time and the number of patterns quickly increases even at relatively high support thresholds (at support 30, 10,000 patterns are produced out of a database of 419 graphs).*

Due to the fact that most of the compounds are highly consist of three atoms, namely, C, H, and O, final pattern set sizes quickly increases.

5. RELATED WORK

Taxonomy-based data mining is first considered in the context of association rule mining in market-basket data [18] where each item in a transaction (i.e., itemset) is a member of is-a hierarchy of product categories. An interest measure based on expected support of itemsets is employed to prune out redundant patterns. In addition, Srikant and Agrawal [17] propose another algorithm for mining sequential patterns of itemsets where each item is a member of a is-a hierarchy. Although these algorithms are efficient and scalable, they are specifically designed for mining frequent itemsets, and are not directly applicable to taxonomy-superimposed graph mining.

Most work in graph mining [6, 7, 11, 20, 22] focus on extracting the exact frequent patterns. Canonical forms are utilized [11] to test whether two graphs are isomorphic. However, many of the existing frequent subgraph mining methods do not consider the graph structures where nodes are part of a well-defined hierarchy.

Inokuchi extended AcGM [9] to mine generalized substructures from labeled graphs. AcGM is a level-based graph mining algorithm [8]. More specifically, size-k graphs are created by joining size k-1 graphs where graphs are represented as adjacency matrices, and associated with canonical forms to identify the isomorphic graphs. The extension to AcGM involves replacing standard graph isomorphism test with the generalized isomorphism test which takes advantage of the is-a hierarchies provided by the associated taxonomy. And, over-generalized patterns are pruned if there is a more specific pattern with the same embedding set. However, AcGM suffers from two major issues: (a) it is a breadth-first level-wise algorithm, hence, cannot scale to large taxonomies or graphs, (b) it handles a pattern and its generalizes/specific versions independently, which causes repetitive isomorphism test for occurrences that are shared by a *class of patterns* that have the same structure with different specificity. Taxogram addresses these shortcomings of extended AcGM via (a) storing the shared occurrences of patterns that are members of the same pattern class, (b) enumerating specialized patterns in depth-first manner which enables mining under lower support thresholds, or larger size taxonomies in comparison with AcGM.

6. CONCLUSION AND FUTURE WORK

We explored the properties of taxonomy-superimposed graph mining, and propose an efficient and scalable algorithm, Taxogram, to mine frequent taxonomy-superimposed graph structures. Taxogram first relabels the input database, then mines for frequent pattern classes while constructing occurrence indices, and finally enumerates members of each pattern class using the constructed occurrence indices. We extensively evaluated our approach along with both synthetic and real data, and showed that Taxogram can handle a diverse set of graph databases associated with different taxonomies.

As shown with a discussion on space and time complexity, taxonomy-superimposed graph mining is costly, and requires enormous amount of computational resources. As future work, we plan to develop disk-based algorithms for taxonomy-based graph mining.

References

[1] Greg Butler, Guang Wang, Yue Wang, and Liqian Zou. "A graph database with visual queries for genomics." *In Proceedings of APBC*, 2005.

[2] Ali Cakmak and Gultekin Ozsoyoglu. "Mining Biological Networks for Unknown Pathways." Bioinformatics 23:20 (2007): 2775 - 2783.

[3] Ali Cakmak, Mustafa Kirac, Marc Reynolds, Meral Ozsoyoglu, and Gultekin Ozsoyoglu. "Gene Ontology-Based Annotation Analysis and Categorization of Metabolic Pathways." *In Proceedings of SSDBM*, 2007.

[4] Gene Ontology Consortium. "The GO database and informatics resource." *Nucleic Acids Res*, 2004: 32.

[5] Thomas Cover and Joy A. Thomas. *Elements of Information Theory.* New York: Wiley-Interscience, 2006.

[6] Jun Huan, Wei Wang, and Jan Prins. "Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism." *In Proceedings of ICDM,* 2003. 549-552.

[7] Jun Huan, Wei Wang, Jan Prins, and Jiong Yang. "SPIN: Mining Maximal Frequent Subgraphs from Graph Databases." *In Proceedings of SIGKDD,* 2004. 581-586.

[8] Akihiro Inokuchi, T. Washio, Y. Nishimura, and H. Motoda. *A Fast Algorithm for Mining Frequent Connected Graphs.* IBM Research Report, Feb. 2002.

[9] Akihiro Inokuchi. "Mining Generalized Substructures from a Set of Labeled Graphs." *In Proceedings of ICDM,* 2004. 415-418.

[10] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data." *In Proceedings of PKDD,* 2000. 13-23.

[11] Michihiro Kuramochi and George Karypis. "Frequent Subgraph Discovery." *In Proceedings of ICDM,* 2001. 313-320.

[12] *Nitrogen metabolism entry at Wikipedia.* http://en.wikipedia.org/wiki/Category:Nitrogen_metabolism.

[13] Kyoto Encyclopedia of Genes and Genomes http://www.genome.jp/kegg/.

[14] *Predictive Toxicology Challenge Web Site.* http://www.predictive-toxicology.org/ptc/.

[15] Sriram Raghavan and Hector Garcia Molina. "Representing Web Graphs." *In Proceedings of ICDE*, 2003.

[16] Philip Resnik. "Using Information Content to Evaluate Semantic Similarity in a Taxonomy." *In Prooceedings of IJCAI*, 1995.

[17] Ramakrishnan Srikant and Rakesh Agrawal. "Mining Sequential Patterns: Generalizations and Performance Improvements." *In Proceedings of EDBT*, 1996.

[18] Ramakrishnan Srikant and Rakesh Agrawal. "Mining Generalized Association Rules." *In Procedings of VLDB*, 1995.

[19] Marc Worlein et al. "A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston." *In Proceedings of PKDD,* 2005. 392-403.

[20] Xifeng Yan and Jiawei Han. "CloseGraph: mining closed frequent graph patterns." *In Proceedings of SIGKDD,* 2003. 286-295.

[21] Xifeng Yan and Jiawei Han. *gSpan, Expanded Version.* Urbana-Champaign: UIUC Technical Report, 2002.

[22] Xifeng Yan and Jiawei Han. "gSpan: Graph-Based Substructure Pattern Mining." *In Proceedings of ICDM,* 2002. 721-724.