Approximate Graph Mining with Label Costs *

Pranay Anchuri and Mohammed J. Zaki
CS Department, RPI, Troy NY, USA
{anchupa,zaki}@cs.rpi.edu

Omer Barkol, Shahar Golan, Arik Sityon, Moshe Shamy
HP Labs, Haifa, Israel
{omer.barkol, shahar.golan, arik.sityon, moshe.shamy}@hp.com

ABSTRACT

Many real-world graphs have complex labels on the nodes and edges. Mining only exact patterns yields limited insights, since it may be hard to find exact matches. However, in many domains it is relatively easy to compute some cost (or distance) between different labels. Using this information, it becomes possible to mine a much richer set of approximate subgraph patterns, which preserve the topology but allow bounded label mismatches. We present novel and scalable methods to efficiently solve the approximate isomorphism problem. We show that the mined approximate patterns yield interesting patterns in several real-world graphs ranging from IT and protein interaction networks to protein structures.

1. INTRODUCTION

Graphs are a natural way to model many of the modern complex datasets that typically have interlinked entities connected with various relationships. Examples include different types of networks, such as social, biological and technological networks. Tools for rapidly querying and mining graph data are therefore in high demand. Our focus is on graph pattern discovery methods that can simultaneously consider both the structure and content (e.g., node labels).

Whereas frequent graph mining has long been a well studied problem, most of the prior work has focused on exact pattern discovery. Graph mining involves two main steps. The first step is to generate non-duplicate candidate patterns, and the second is to compute the frequency of each candidate pattern. The former task requires graph isomorphism testing, whereas the latter requires subgraph isomorphism checking, since we need to count all the occurrences of a smaller graph within a much larger graph (or a set of graphs). Many efficient methods have been proposed for mining exact labeled graph patterns, including both complete search and sampling based approaches [**?**, **?**, **?**, **?**, **?**, **?**].

*This work was supported in part by an HP Innovation Award and an NSF Award CCF-1240646.

These exact methods require that there be an exact match between the labels of nodes in the candidate pattern and in the database graph. This can potentially miss many patterns where nodes may share a high label similarity, but may not match exactly. This is specially true for more complex labels (e.g., text data), or in cases where the nodes represent some real-world objects (e.g., proteins, IT infrastructure nodes), where it may be possible to easily design a meaningful cost or distance matrix between node "labels". Unfortunately, exact isomorphism based methods cannot leverage the rich information from the cost matrix. What is required is a new class of algorithms that can mine frequent approximate patterns via approximate subgraph isomorphism that satisfies some bound on the overall cost of the match between a candidate and the database graph(s). Only a few methods have tackled this problem [**?**, **?**, **?**], but they typically enumerate all isomorphisms, and are therefore not scalable to large graphs due to the combinatorial explosion in the number of isomorphisms.

In this paper we present a new approach to mine frequent approximate patterns in the presence of a cost matrix between the labels. In particular we make the following contributions:

- We propose a novel approach to effectively prune the space of approximate labeled isomorphisms. Instead of enumerating all the possible isomorphisms, we maintain a set of representatives (nodes in the database that match a candidate pattern) that is linear in the database and pattern size. Pruning is applied on this set to narrow down the search to only viable mappings.
- We propose several iterative label updating methods that yield derived cost matrices on the basis of which more effective pruning can be achieved. These are based on k-hop labels, neighbor concatenated labels and a combination of the two.
- Our method handles both arbitrary as well as binary cost matrices.
- We place our work within the pattern sampling paradigm, thereby avoiding complete search, which can be practically infeasible in real-world graphs, not to mention the information overload problem.

We study the effectiveness of the proposed methods on three real-world datasets. The first is a configuration management database graph, where the nodes represents entities comprising the IT infrastructure and the link represents relationships between them; approximate mining yields a richer

de-facto IT policies in the company. The second dataset is a graph dataset representing 3D protein structures; mined patterns represent approximate motifs. The last dataset comes from a protein interaction network, where the nodes are proteins and edges indicate whether they interact physically (i.e., they may bind together or they may be part of the same protein complex); the mined approximate patterns represent molecular subnetworks and molecular machines (the protein complexes) that take part in important cellular processes. We show that our proposed techniques are indeed scalable and fruitful, allowing us to mine interesting approximate graph patterns from large real-world graphs.

2. PRELIMINARIES

An undirected labeled graph G is represented as a tuple $G = (V_G, E_G, L)$ where V_G is the set of vertices, E_G is the set of edges and L: $V_G \rightarrow \Sigma$ is a function that maps vertices to their labels. The neighbors of a vertex v are given as $N(v) = \{u|(u, v) \in V_G\}$. A *path* in a graph G is a sequence of vertices $v_0, \ldots, v_k$ such that $v_i \in V_G$, $(v_i, v_{i+1}) \in E_G$ and $v_i \neq v_j$. We use $u \xrightarrow{k} v$ if there is a k edge path between u and v. A path is called a walk if the vertices are allowed to repeat. [can we define this when required ?]

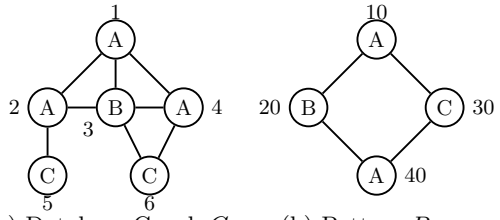
Cost matrix: We assume that there is a cost matrix C: $\Sigma^2 \rightarrow \mathbb{R}_{\geq 0}$. The entry $C[l_i][l_j]$ denotes the cost of matching the labels l_i and l_j. Although it is not required by the algorithm, C is usually symmetric and the diagonal entries are 0

Approximate subgraph isomorphism: A graph $S = (V_S, E_S, L)$ is a subgraph of G, denoted $S \subseteq G$, iff $V_S \subseteq V_G$, and $E_S \subseteq E_G$. S is an induced subgraph if $E_S = E_G \cap (V_S \times V_S)$. Given a database graph G and a pattern $P = (V_P, E_P, L)$, a function ϕ: $V_P \rightarrow V_G$ is called an *unlabeled subgraph isomorphism* provided ϕ is an injective (or one-to-one) mapping such that $\forall (u, v) \in E_P$, we have $(\phi(u), \phi(v)) \in E_G$. That is, ϕ preserves the topology of P in G. Define the cost of the isomorphism as follows: $C(\phi) = \sum_{u \in V_P} C[L(u)][L(\phi(u))]$, that is, the sum of the costs of matching the node labels in P to the corresponding node labels in G. We say that ϕ is an *approximate subgraph isomorphism* from P to G provided its cost $C(\phi) \leq \alpha$, where α is a user-specified threshold on the total cost. In this case we also call P an approximate pattern in G. Note that if $\alpha = 0$, then ϕ is an unlabeled subgraph isomorphism between P and G. From now on, *isomorphism* refers to *approximate subgraph isomorphism* unless specified otherwise.

Representative set and pattern support: Given a node $u \in V_P$, its *representative set* in the database graph G is the set

$$R(u) = \{v \in V_G | \ \exists \phi, \text{ such that } C(\phi) \leq \alpha \text{ and } \phi(u) = v\}$$

That is, the representative set of u comprises all nodes in G that u is mapped to in some approximate isomorphism. Figure 1 shows an example database, a cost matrix, an approximate pattern, and its approximate subgraph isomorphism for $\alpha = 0.5$. There are only two possible approximate isomorphisms from P to G, as specified by ϕ_1 and ϕ_2. For example, for ϕ_1, we have $\phi_1(10) \rightarrow 3$, $\phi_1(20) \rightarrow 1$, $\phi_1(30) \rightarrow 6$, and $\phi_1(40) \rightarrow 4$, as seen in Table 1d. The cost of the isomorphism is $C(\phi_1) = 0.4$, since $C(L(10), L(3)) + C(L(20), L(1)) + C(L(30), L(6)) + C(L(40), L(4)) = C(A, B) + C(B, A) + C(C, C) + C(A, A) = 0.2 + 0.2 + 0 + 0 = 0.4$. The



(a) Database Graph G (b) Pattern P

C	A	B	C	D
A	0	0.2	0.6	0.1
B	0.2	0	0.4	0.5
C	0.6	0.4	0	0.2
D	0.1	0.5	0.2	0

(c) Cost Matrix

	ϕ				
P	10	20	30	40	cost
ϕ_1	3	1	6	4	0.4
ϕ_2	4	1	6	3	0.4

(d) Approximate Embeddings

Figure 1: (a): sample database graph G, (b): approximate pattern P. (c): cost matrix. (d): approximate embeddings of P in G.

representative set for node $10 \in V_P$ is $R(10) = \{3, 4\}$. However, the support of P is $sup(P) = 1$, since node $20 \in V_P$ has only one mapping in G, namely $R(20) = \{1\}$.

2.1 Outline

Motivate the reason for computing the representative set and outline the rest of the paper.

3. RELATED WORK

In the past, many algorithms have been proposed to mine subgraphs from a given database of graphs. These algorithms can be mainly divided in to two classes depending on how the candidate patterns are generated. Algorithms like those in [**?**, **?**, **?**] are Apriori based methods, i.e., a candidate pattern of size $k+1$ is generated by combining two frequent graphs of size k that have a common $k-1$ sized subgraph. Algorithms like those in [**?**, **?**, **?**], on the other hand, belong to the class of pattern growth algorithms in which a candidate pattern is generated by extending a frequent pattern with an edge.

Mining subgraphs from a single graph is a related problem which is surprisingly difficult compared to mining from a database of graphs. In [**?**], they defined the support of a pattern in a single graph as the maximum number of edge disjoint isomorphisms, which is itself a NP-Hard problem. In [**?**], they proposed a definition of support based on overlapping ancestor isomorphisms. In [**?**], they proposed CMDB-Miner to mine frequent patterns from a single large graph. Support of a pattern is defined as the maximum flow in an appropriately constructed flow network with capacities. This method estimates the support of a pattern without enumerating its isomorphisms. The authors also proposed methods to summarize the maximal frequent patterns extracted from the graph. In [**?**], they proposed an algorithm to extract frequent patterns from dense graphs. It uses $GADDI$ index proposed in [**?**] to efficiently extract the isomorphisms of a subgraph.

There has not been much work in approximate subgraph mining. In [**?**], they proposed *gApprox* to mine approximate frequent subgraphs. The degree of approximation between a pattern and its isomorphism includes label mismatches and missing edges. The search space is explored in a depth first

order and the support of a pattern is computed by enumerating its isomorphisms. This approach is not feasible for large graphs with label multiplicities as there are potentially exponential number of isomorphisms [?]. In [?], they proposed *APGM* to approximate frequent subgraphs from a database of graphs. The method is similar to the *gApprox* method in that it stores the complete set of approximate embeddings of the current frequent pattern. The difference, however, is that in [?] the entire 1-hop neighborhood of the current embeddings is explored to enumerate all extensions of the frequent pattern and their corresponding embeddings, whereas *gApprox* enumerates the embeddings for a single extension in each step. Recently [?] applied frequent approximate subgraph mining to the task of image classification. The method is similar to *APGM* but allows for edge mismatches also. In [?], the authors proposed strategies to speed up the existing mining algorithms, by limiting the number of candidates and also the number of duplicate checks performed. In [?], they proposed a randomized algorithm to mine approximate patterns from a database graphs. In this method, an edge in the approximate pattern is required to have at least a given number of occurrences.

Graph querying is another problem that is related to subgraph mining. The goal is to find matches of a given query graph in a single graph or database of graphs. In [?], they proposed an indexing method to extract the approximate occurrences of a given graph query in large graph databases. The algorithm proposed in [?] extracts selective fragments from the query graph and queries against an index constructed from the fragments of the database graphs. In [?], they proposed a polynomial time algorithm for detecting isomorphism between spectrally distinguishable graphs. An isomorphism, if it exists, is obtained by matching the steady state vectors of Markov chains in both the graphs. In [?], they proposed indexing and retrieval methods for graph models. The problem with most of the indexing approaches is that they are efficient in retrieving a single match for the query graph but fail at retrieving all matches, and thus are not suited to mine frequent patterns. Furthermore, they assume that the query given, and thus they do not perform pattern enumeration as required in graph mining.

Frequent pattern mining algorithms usually return a large number of patterns and interpreting them is a big challenge. This is especially true if the output is presented to a human user for further analysis. Sampling approaches like those proposed in [?, ?, ?] mine a representative set of maximal patterns from a database of graphs or a single graph. In our work, we perform a random walk in the search space to enumerate a maximal pattern.

The concept of using derived labels based on the structure and the attributes is frequently used in detecting graph isomorphism [?] and computing graph kernels [?, ?]. Our methods to prune representative sets are somewhat similar to the wiesfieler lehman kernel to test isomorphism between two graphs G_1 and G_2 [?], where after every iteration the labels are sorted and renamed with a different string in such a way that the pair wise relationship is maintained. G_1 is not isomorphic to G_2 if the label set of the graphs differ. The specifics of how and what information we update is different, and also we use the information for subgraph isomorphism instead of graph isomorphism.

4. COMPUTING REPRESENTATIVE SETS

Representative vertex v of a pattern vertex u implies that there exists an isomorphism ϕ for which $\phi(u) = v$. One way to interpret it is that the neighborhood of u matches with that of v. By comparing the neighborhoods we can find vertices that are not valid representatives of u without trying to find an isomorphism exhaustively. Therefore, to compute the representative sets we will start with a candidate representative set denoted by $R'(u)$ and iteratively prune some of the vertices if the neighborhoods cannot be matched. The candidate set is a superset of the representative set, $R'(u) \supseteq R(u)$. An example of candidate set, $R'(u) = \{v | v \in V_G, C[L(u)][L(v)] \leq \alpha\}$ i.e., the isomorphisms of the single vertex pattern with label $L(u)$. In this section, we will describe different notions of neighborhood and show how they help us in computing the representative sets of vertices in a pattern.

The problem of checking whether a vertex $v \in R(u)$ involves solving isomorphism which is a NP complete problem. The pruning methods typically do not prune all the invalid vertices. So, we use an exhaustive enumeration method to prune these invalid vertices and reduce $R'(u)$ to $R(u)$.

4.1 k-hop Label

k-hop label is defined as the set of vertices that are reachable via a simple path of length k. In other words, k-hop label contains all vertices that are reachable in k-hops starting from u and by visiting each vertex at most once. Note that, we use the word label even though we refer to a set of vertices. Formally, the k-hop label of a vertex u in graph G, $h_k(u, G) = \{v | v \in G, u \xrightarrow{k} v\}$. We simply write it as $h_k(u)$ when the graph is evident from the context. For example, for pattern P in Fig. 2a, the 0-hop label of vertex 5 is $h_0(5) = \{5\}$, its 1-hop label is the multiset $h_1(5) = 2, 4, 6$ (we omit the set notation for convenience) and its 2-hop label $h_2(5) = 1, 3$. The minimum cost of matching k-hop labels $h_k(u)$ and $h_k(v)$ is

$$C_k[h_k(u)][h_k(v)] = \min \sum_{u' \in h_k(u)} C[L(u')][L(f(u'))] \quad (1)$$

where the minimization is over all injective functions f: $h_k(u) \to h_k(v)$ and $C[L(u')][L(f(u'))]$ is the cost of matching the vertex labels. In other words, it is the minimum total cost of matching the vertices present in the k-hop labels. The following theorem places a upper bound on the minimum cost of matching the k-hop labels of pattern vertex and any of its representative vertices.

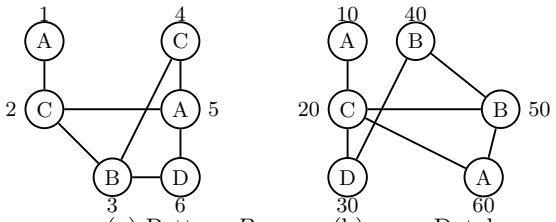
Theorem 1. *Given any pattern vertex u, a representative vertex $v \in R(u)$ and cost threshold α, the minimum cost of matching the k-hop labels, $C_k[h_k(u)][h_k(v)] \leq \alpha$ for all $k \geq 0$.*

Proof. *Consider any isomorphism ϕ such that $\phi(u) = v$. It is enough if we can show an injective function f: $h_k(u) \to h_k(v)$ with a cost (as defined in equation 1) $\leq \alpha$. We will argue that the function ϕ on the restricted domain $h_k(u)$ is one such function f. First, we know that $\sum C[L(u)][\phi(L(u)] \leq \alpha$, $u \in V_P$, since ϕ is an isomorphism. Second, let $u \xrightarrow{k} u'$ then $\phi(u') \in h_k(v)$ because for every edge (u_1, u_2) on a path between u and u' in P, $(\phi(u_1), \phi(u_2)) \in E_G$. Therefore the minimum cost of matching the k-hop labels is upper bounded by α.*

Based on the above theorem, a vertex v is an invalid vertex if $C_k[h_k(u)][h_k(v)] > \alpha$ for any $k \geq 0$. However, in practice, it enough to check the condition only for $k \leq |V_P|-1$ because $h_k(u)$ is the null set $\forall k \geq |V_P|$ and the condition is trivially satisfied.

Figure 2 shows an example for the k-hop label based pruning of the candidate representative set where the threshold $\alpha = 0.5$. Consider vertex $2 \in V_P$ and vertex $20 \in V_G$, we have, $C_0[h_0(2)][h_0(20)] = 0$, since the cost of matching vertex labels $C[L(2)][L(20)] = 0$, as per the label matching matrix C in Fig. 2c. The k-hop labels for $k = 1, 2, 3$ and the minimum of cost matching them are as shown in the table 1, and it can be verified that the minimum cost is within the threshold α.

Thus far, we cannot prune node 20 from $R'(2)$. However, $h_4(2) = 4, 5$ and $h_4(20) = 30, 60$ and the minimum cost of matching them is $0.6 > \alpha$. Thus, we conclude that $20 \notin R'(2)$. This example illustrates that k-hop labels can help prune the candidate representative sets.



(a) Pattern P (b) Database Graph G

C	A	B	C	D
A	0	0.7	0.6	0.1
B	0.7	0	0.3	1
C	0.6	0.3	0	0.8
D	0.1	1	0.8	0

(c) Cost Matrix

Figure 2: Pattern (a), database graph (b), and binary cost matrix (c).

k	$h_k(2)$	$h_k(20)$	$C_k[h_k(2)][h_k(20)]$
1	1, 3, 5	10, 30, 50, 60	0
2	4, 6	40, 50, 60	0.4
3	3, 5	40, 30, 50	0.1

Table 1: k-hop label of vertices 2 and 20

k	$h_k(3)$	$h_k(50)$	$C_k[h_k(3)][h_k(50)]$
0	3	50	0
1	2, 4, 6	20, 40, 60	0.4
2	1, 5	10, 20 , 30, 60	0
3	2, 4, 5	10, 20, 30, 40	0.3
4	1	10, 40, 60	0

Table 2: k-hop labels of vertices 3 and 50.

4.2 Neighbor Concatenated Label

In Neighbor concatenated label (NL) , the information regarding the candidates of a neighbor that were pruned in the previous iteration is used along with the current k-hop label to prune candidates in the current iteration. In contrast, the k-hop label pruning strategy for a vertex u works independently of the result of k-hop label pruning of other vertices in the pattern. This leads us to the following recursive formulation for NL.

The NL of a vertex in the $k+1^{th}$ iteration, $\eta_{k+1}(u)$, is defined as the tuple $(\{\eta_k(u')|u' \in N(u)\}, h_{k+1}(u))$. The first element(A) of the tuple is the NL of the neighbors of the vertex u in the previous iteration and the second element(B) is exactly same as the k-hop label defined in the previous section. We say that $\eta_{k+1}(v)$ dominates $\eta_{k+1}(u)$, denoted by $\eta_{k+1}(u) = (A, B) \preceq \eta_{k+1}(v) = (A', B')$, i) iff $C_{k+1}[h_{k+1}(B)][h_{k+1}(B')] \leq \alpha$ i.e., the minimum cost of matching the k-hop labels is within α ii) there exits an injective function g: $A \to A'$ such that $a \preceq g(a)$ for all $a \in A$ i.e., there is a one to one mapping between the NL labels (in the previous iteration, k) of neighbors of u and v. The base case $\eta_0(u) \preceq \eta_0(v)$ iff $C[L(u)][L(v)] \leq \alpha$. For example, in Fig 2 $\eta_1(2) \preceq \eta_1(20)$ because $C_1[h_1(2)][h_1(20)] \leq \alpha$ and the NL labels of vertices $1, 3, 5$ are dominated by the NL labels of vertices $10, 50, 30$ respectively. The following theorem states that the NL of a pattern vertex u is dominated by the NL of any of its representative vertex $v \in R(u)$.

Theorem 2. *Given any pattern vertex u, a representative vertex $v \in R(u)$ and cost threshold α, $\eta_k(u) \preceq \eta_k(v)$ for all $k \geq 0$.*

Proof. *Let ϕ be any isomorphism such that $\phi(u) = v$. We prove the theorem by using induction on k.*
Base case: *$\eta_0(u) \preceq \eta_0(v) \iff C[L(u)][L(v)] \leq \alpha$ is true because $v \in R(u)$.*
Inductive Hypothesis: *Assume that $\eta_k(u) \preceq \eta_k(v)$ holds true for all $u \in P$ and $v \in R(u)$.*
Now consider $\eta_{k+1}(u) = (A, B)$ and $\eta_{k+1}(v) = (A', B')$, from theorem 1 we know that $C_{k+1}[h_{k+1}(B)][h_{k+1}(B')] \leq \alpha$, for all $k \geq 0$. Let u' be a neighbor of u and let $v' \in \phi(u')$, then from the inductive hypothesis $\eta_k(u') \preceq \eta_k(v')$. Therefore, the injective function ϕ maps the elements $a \in A$ to $\phi(a) \in A'$. The theorem follows from the definition of the NL label.

Based on the above theorem, a vertex v can be pruned from $R'(u)$ if $\eta_k(u) \not\preceq \eta_k(v)$ for some $k \geq 0$. In Fig 2 , consider the vertices $3 \in P$, $50 \in G$ and let $\alpha = 0.5$. The NL labels, $\eta_0(3) \preceq \eta_0(50)$ as $C[B][B] = 0 \leq \alpha$. Similarly it is also true for the pairs $(2, 20)$, $(4, 40)$ etc. It follows that $\eta_1(3) \preceq \eta_1(50)$ as the neighbors $2, 4, 6$ can be mapped to $20, 40, 60$ respectively and the minimum cost of the matching the 1-hop label is 0.4 which is less than the α threshold. But $\eta_2(3) \not\preceq \eta_2(50)$ because the NL label $\eta_1(6)$ is not dominated by the NL label of $20, 40$ or 60 in the previous iteration. So, there is no mapping between the neighbors of vertices 3 and 50 in the current iteration. Hence, the vertex 50 can be pruned from the candidate representative set of vertex 3. Note that using the k-hop label in the same example will not prune the vertex 50 because the minimum cost of matching the k-hop labels is within α as shown in table 2. Therefore, NL label is more efficient compared to k-hop label as it subsumes the latter label.

4.3 Candidate set verification

The pruning methods based on the k-hop and the NL labels start with a $R'(u)$ and prune some of the candidate vertices based on the conditions described in theorems 1 and 2. The verification step reduces $R'(u)$ to $R(u)$ by retaining only those vertices for which there exists an isomorphism ϕ in which $\phi(u) = v$. It does this by checking if the pattern P can be embedded at v such that total cost of label mismatch is at most α.

Let $w_p = u_0, \ldots, u_m$ be a walk in the pattern that covers each edge of the pattern atleast once starting from vertex u i.e. $u_0 = u$. The following three conditions are satisfied iff the vertex v is a representative of the vertex u. i) there exists a walk $w_d = v_0, \ldots, v_m$ such that $v_i \in R(u_i)$ and $v_{i+1} \in R(u_{i+1})$, $\forall (v_i, v_{i+1}) \in w_d$. ii) $v_i = v_j$ implies that $u_i = u_j$. iii) $\sum C[L(u_i)][L(v_i)] \leq \alpha$ where $u_i \in V_P$. These conditions can be verified by following the definition of approximate subgraph isomorphism. Using an example we will show how these conditions can be used to check definitely whether $v \in R(u)$.

Consider checking whether the vertex 3 is a valid representative of the vertex 10 in the pattern in the figure 1 and let $\alpha = 0.5$. The walk $w_p = 10, 20, 40, 30, 10$ covers each edge of the pattern atleast once. A Walk that satisfies the above three conditions should start mapping the vertex 10 to 3. The cost of matching the labels of vertices 3 and 10 equals 0.2 and is subtracted from the cost matrix. We will cover the walk w_p one edge at a time. For any $(u_i, u_{i+1}) \in w_p$, if u_i and u_{i+1} are mapped, then we verify that there is an edge between the database vertices v_i, v_{i+1} to which the pattern vertices are mapped. If however, u_{i+1} is not mapped then we map it to some vertex in $v_{i+1} = R'(u_{i+1})$ and subtract the cost of this mapping from the remaining α cost. For example, in the first step we can map 20 to 2. The remaining cost is then $0.3 - 0.2 = 0.1$. In the next step $(20, 30)$, the vertex 30 cannot be mapped to any vertex in the database without violating the third condition. In such a case, we back track one step and choose a different mapping say 1 for the vertex 20 which is the last vertex that was mapped. Proceeding this way, we can arrive at the mapping corresponding to ϕ_1 as in Table 1d. This isomorphism not only guarantees that $3 \in R(10)$, it also implies that the verification check between the pairs $(20, 1), (30, 6)$ and $(40, 4)$ can be avoided because of the approximate isomorphism ϕ_1 that was found. The above procedure can extended to enumerate the complete set of isomorphisms.

4.4 Label costs and dominance checking

Candidate representative vertices are pruned by checking for dominance between the NL labels which requires computing the cost between the k-hop labels. Both these tasks involve finding an injective function that tries to optimize some objective function. These problems can be modeled as network flow problems for which there are many efficient algorithms.

Computing k-hop label cost: To compute the minimum matching cost between the k-hop labels $h_k(u)$ and $h_k(v)$ we compute the maximum flow with minimum cost in the flow network F defined as follows. F contains two groups of vertices in addition to designated source and sink nodes. First group contains a node for each label that appears on a vertex $u' \in h_k(u)$ and it is connected to src node with zero weight and the capacity is equal to the number of vertices in $h_k(u)$ that have the same vertex label as that of u'. The second group contains nodes for labels that are present on $v' \in h_k(v)$ and are connected sink node with a weight of 0 and a capacity equal to the number of nodes in $h_k(v)$ that have the same label as that of v'. The only other edges are those between the vertices in different groups. Consider vertices corresponding to labels l_u and l_v in the two groups respectively. The weight of the edge is $C[l_u][l_v]$ and the infinite capacity. The cost between the k-hop labels is equal to minimum cost for maximum flow if the total flow is equal to $|h_k(u)|$ and ∞ otherwise.
Figure 3 shows the flow network required to compute the minimum cost of matching the k-hop labels $h_2(2)$ and $h_2(20)$ as shown in Table 1. The vertices connected to the source, s correspond to the graph labels of $h_2(2)$ and the vertices connected to the sink, t, correspond to the graph labels of $h_2(20)$. The numbers on the edges represent the capacity and the weight of the edges. Note that the number of vertices connected to the sink is only 2 is even though $|h_2(20)| = 3$ because the vertices 40 and 50 have the common graph label B. The vertex corresponding to B is therefore connected to t with a capacity of 2. The maximum flow in the network is 2 and the minimum cost of sending 2 units of flow 0.4 is achieved by pushing a unit flow along the paths s, C, B, t and a, D, A, t. Therefore, the cost of matching the labels $h_2(2)$ and $h_2(20)$ is 0.4.

Dominance check: The k-hop label part of the NL label can be compared using the above network flow formulation and matching the recursive part of the NL label can be done using maximum bipartite matching. Consider $\eta_{k+1}(u) = (A, B)$ and $\eta_{k+1}(v) = (A', B')$, the existence of injective function between A and A' can be checked by computing the maximum matching in a bipartite graph with edges (a, a') where $a \in A$ and $a' \in A'$. The NL label $\eta_k(v)$ therefore dominates $\eta_k(u)$ if the cost between the k-hop labels is within α and the size of maximum bipartite matching is $|N(u)|$.

Optimization: The candidate pattern may contain groups of symmetric vertices that are indistinguishable with respect to the k-hop label. In such a scenario, the candidate representative sets of all these vertices are exactly the same. Utilizing the symmetry, we apply the pruning strategy only on one vertex per symmetry group and replicate the results for all other vertices in the group. For example, the vertices 10 and 40 in figure 1b are symmetric and the representative sets $R'(10)$ and $R'(40)$ are exactly the same. In abstract algebra terms such groups are called orbits of the graph and can be computed using nauty algorithm. Even though computing the orbits is expensive, we can avoid $(|g|-1) \times |R'(u)|$ k-hop label cost computations where g is the size of the orbit. Note that the payoff is zero if all the vertex orbits are of size 1.

4.5 Precomputing database k-hop labels

The k-hop label of the database vertices is independent of the candidate pattern. Hence, even though computing the cost of matching the k-hop labels using maximum flow is expensive, it can be amortized over all the candidate patterns if we precompute the k-hop labels of the database vertices. Also, to construct the flow network we only need the aggregate information about the number of vertices of a given database label (TODO) in the k-hop label which can be stored in the main memory.

Theorem 3. *k-reachable (KR) : Given a graph G, k and*

$u \in V_G$. *Compute* $h_k(u)$. *k-reachable cannot be solved in polynomial time unless* $P = NP$.

Proof. *We prove this by reducing hamiltonian path (HP) to KR. Hamiltonian Path : Given a graph G, is there a simple path of length* $|V_G| - 1$ *i.e. is there a path that visits each and every vertex exactly once. The problem of finding a Hamiltonian path is proven to be an NP-Complete problem [?].*
Assume that algorithm $X(k)$ *can compute KR in polynomial time. Let* $|V_G| = n$ *and* u *be the starting vertex in HP if it exists. Given an instance of HP, we first get a vertex* v, $u \xrightarrow{n-1} v$ *using* $X(n-1)$. *The vertex* v *is removed from the graph and we find a vertex* v' *such that* $u \xrightarrow{n-2} v'$ *and* $(v', v) \in E_G$. *We repeat this process* $n-1$ *times. If at any stage* $X(j) = \{\}$ *then we restart from a different starting vertex. The vertices selected in each iteration lie on a path of length* $n-1$ *if it exists.*

To compute k-hop label of a vertex u, we check for each vertex v whether $v \in u \xrightarrow{k} v$ by enumerating all possible k length paths until a path is found. This procedure is exponential, we therefore fix a maximum value k_{max} and use the k-hop label based pruning only for values of $k \leq k_{max}$. It only takes a couple of minutes to compute the k-hop label for $k \leq 6$ for all the vertices in the database graph. This is significantly less than the overall run time of the algorithm. Once $h_k(u)$ is computed we store in memory only the tuples (m, l) where m is the number of vertices $v \in h_k(u)$, $L(v) = l$.

4.6 Complexity

Memory and time complexity of the entire algorithm.

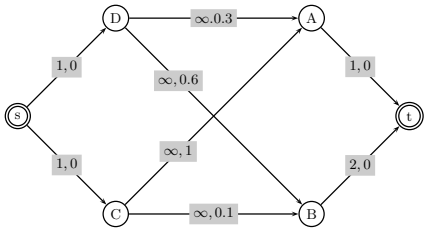


Figure 3: Flow network for $h_2(2)$ and $h_2(20)$

5. MINING ALGORITHM

The mining algorithm involves candidate generation, support computation in addition to finding the set of approximate subgraph isomorphisms. Representative sets of vertices described in the previous section are a view of all the isomorphisms of the pattern in the input graph. In this section, we will show how the representative sets can be used in conjunction with different candidate generation and support computation techniques to yield approximate graph mining with different properties.

5.1 Candidate Generation

The search space of the frequent patterns forms a partial order. It can be explored in a depth first or breadth first order but doing so requires computing canonical code to avoid duplicates. Since, the search space is exponential, sampling methods have gained traction in recent times [**?**, **?**] . The candidate patterns these methods generate from a frequent pattern depend on a random variable.

In our experiments we employed the random walk strategy proposed in [**?**] to mine exact patterns from a single large graph. Each random walk starts with an empty pattern and repeatedly adds new edges (to new vertices) or connects two existing vertices in the pattern to generate a new candidate. For each candidate pattern we decide, if its support $sup(P) \geq minsup$. If the support function is anti-monotonic, i.e., a supergraph cannot have higher support, we can prune the entire subtree under a pattern P if it is not frequent. A walk terminates when no extension is frequent, in which case the pattern is output, since it must be maximal. The algorithm terminates when K walks have been done, or alternatively, when K distinct maximal approximate patterns have been output. But, if the application requires a complete set of maximal patterns an ordered exploration of the search space may be employed.

5.2 Support Computation

The support of a pattern is a function on the set of approximate subgraph isomorphims. The anti-monotonicity of this function helps pruning the otherwise exponential search space. When mining from a database of graphs, a function as simple as the total number of graphs having atleast one isomorphism is anti-monotonic. This approach cannot be used when mining from a single graph as it leads to a binary support function which is not very informative. On the other hand, counting the number of isomorphisms is not anti-monotonic because a graph can have more isomorphisms compared to its sub graph.

An anti-monotonic support function for a single graph is the maximum number of vertex disjoint isomorphisms. However, this requires computing the maximum independent set (MIS) of graph where each node represents an isomorphism. Clearly, it is not feasible when the input graphs are large and patterns have large number of embeddings. An easy upper bound for the MIS support is the size of the smallest representative set of a vertex in the pattern. Define the *support* of pattern P in a database graph G as

$$sup(P) = \min_{u \in V_P} \{|R(u)|\}$$

That is, the minimum cardinality over all representative set of nodes in P. With this definition of support, any subset of isomorphisms with cardinality greater than $sup(P)$ is not pairwise disjoint. Since, there are only $sup(P)$ distinct mappings for atleast one vertex in the pattern. Hence, $sup(P)$ is an upper bound on the MIS support. Other upper bounds for the MIS value have been proposed in gApprox and CMDB-Miner algorithms. The support function used in gApprox can be computed from the representative sets by enumerating the isomorphims as described in the Section 4.3. The support function used by the CMDB-Miner algorithm can also be used by constructing an appropriate flow network on the representative sets.

In conclusion, we can mix and match different techniques for candidate generation and support computation to produce different versions of the approximate graph mining algorithm even though the isomorphisms are stored as representatives.

6. EXPERIMENTAL EVALUATION

We ran experiments on real world datasets to evaluate the performance of the algorithm. All the experiments were

run on an 4GB Intel Core i7 machine with a clock speed of 2.67 GHz running Ubuntu Linux 10.04. The code was written in C++ and compiled using g++ version 4.4 with -O3 optimization flag. The default number of random walks is $K = 500$.

Dataset	$\|V\|$	$\|E\|$	$\|\Sigma\|$	Preprocessing time(sec)
CMDB	10466	15122	84	329.31
SCOP	39256	154328	20	17.377
PPI	4950	16515	4950	339.45

(a) Dataset Statistics

Dataset	$\|V\|$	$\|E\|$	Degree
CMDB	11	12.24	2.223
SCOP	5.965	6.725	2.225
PPI	6.453	5.956	1.655

(b) Maximal Pattern Statistics

Figure 4: (a): Input graph statistics, (b): Maximal pattern statistics (the numbers shown are average values).

6.1 Configuration Management Database (CMDB)

A CMDB is used to manage and query the IT infrastructure of an organization. It stores information about the so-called configuration items (CIs) – servers, software, running processes, storage systems, printers, routers, etc. As such it can be considered to be a single large multi-attributed graph, where the nodes represent the various CIs and the edges represent the connections between the CIs (e.g., the processes on a particular server, along with starting and ending times). Mining such graphs is challenging because they are large, complex, multi-attributed, and have many repeated labels. We used a real-world CMDB graph for a large multi-national corporation (name not revealed due to non-disclosure issues) from HP's Universal Configuration Management Database (UCMDB). Table **??** shows the size of the CMDB graph.

Cost Matrix: The set of labels in a CMDB form a hierarchy which can be obtained from HP's UCMDB. In the absence of domain knowledge, one way to obtain a cost matrix is by assigning low costs for pairs of labels that share many ancestors in the hierarchy and high costs otherwise. The algorithm is general in that it doesn't depend on how the label matching costs are assigned, the range of these values or whether the cost matrix is symmetric . Consider any two labels l_1, l_2 and their corresponding paths p_1, p_2 to the root node in the hierarchy. We first define the similarity between the labels to be proportional to the number of common labels in $p_1 \cap p_2$, as follows

$$sim(l_1, l_2) = \frac{|p_1 \cap p_2|}{2} \times \left(\frac{1}{|p_1|} + \frac{1}{|p_2|} \right)$$

The cost of matching the labels is then $C[l_1][l_2] = 1 - sim(l_1, l_2)$.

Results: Figure 5 shows the time for random walks for different values of *minsup* and $\alpha = 0.5$. Interesting, and somewhat counter-intuitively, the time increases for higher minimum support values. The reason is that with higher minimum support, the random walk in the search space goes through the nodes that have a large number of embeddings and it takes more time to enumerate a single pattern.

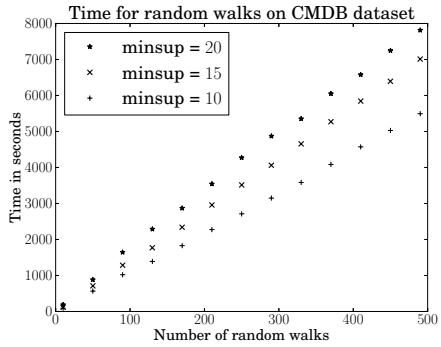
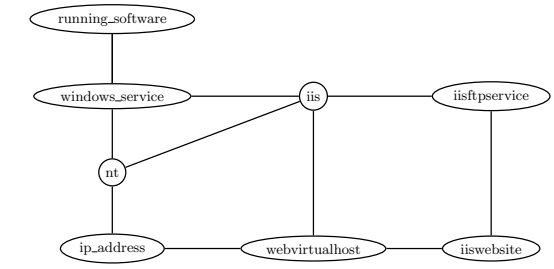
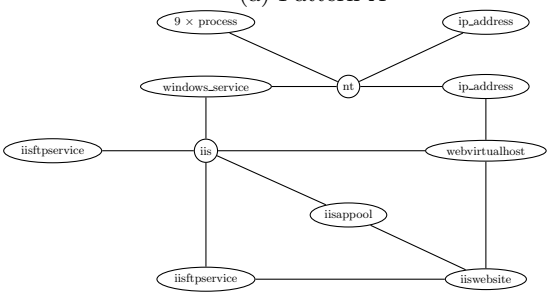


Figure 5: CMDB: Time for different values of *minsup*



(a) Pattern A



(b) Pattern B

Figure 6: CMDB: Approximate Patterns

Example Patterns: Figure 6 shows maximal approximate patterns from the real world CMDB graph. Both these patterns show typical "default" configurations of the IT infrastructure in this company. They show the connection between some services running on an NT server, and also the web/ftp services. The node with label *9 × process* in figure 6b indicates that there are nine nodes in the maximal pattern with label *process* all of which are connected to a common node. This is an example where the run time for computing the representative sets is significantly reduced by the optmization proposed in section 4.4. All of the nine nodes belongs to the same orbit and hence their representative sets are identical.

To show the effectiveness of the pruning based on labels, we compared the time taken to enumerate a single maximal pattern in $CMDB$ database. We compared the time with and without label-based pruning. Both the methods terminated the random walk with the maximal pattern shown

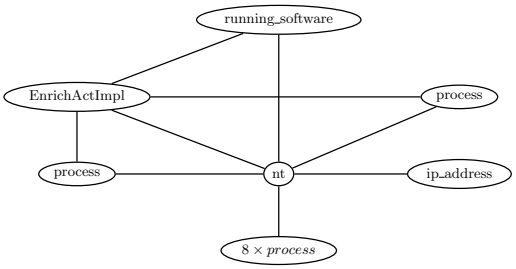


Figure 7: Complete Enumeration Expensive

in Figure 7. However, the total time taken to enumerate the pattern without using any derived label is 18306 secs whereas by using the NL label the total time reduced to only 15.5776 secs. The huge difference between the times arises due to the multiplicity effect in $CMDB$ graphs.

6.2 Protein Structure Dataset (SCOP)

SCOP (`scop.mrc-lmb.cam.ac.uk/scop/`) is a hierarchical classification of proteins based on structure and sequence similarity. The four levels of hierarchy in this classification are: class, fold, superfamily and family. The $3D$ structure of a protein can be represented as an undirected graph with the vertex labels being the amino acids, with an edge connecting two nodes if the distance between the 3D coordinates of the two amino acids (their α-Carbon atoms) is within a threshold (we use 7 Angstroms). We constructed a database of 100 protein structures belonging to 5 different families with 20 proteins from each family. We chose the proteins from different levels in the SCOP hierarchy, and we also focused on large proteins (those with more than 200 amino acids). The 3D protein structures were downloaded from the protein data bank (`http://www.rcsb.org/pdb`). The database can be considered as a single large graph with 100 connected components. For the SCOP dataset, the support is redefined as the number of proteins containing the pattern, i.e., even if a protein contains multiple embeddings we count them only once for the support.

Cost Matrix: Since there are 20 different amino acids, we need a 20×20 cost matrix. BLOSUM62 [**?**] is a commonly used substitution matrix for aligning protein sequences. The i, j entry in BLOSUM denotes the log-odd score of substituting the amino acids a_i and a_j, defined as

$$B[i][j] = \frac{1}{\lambda} \log \frac{p_{ij}}{f_i \cdot f_j}$$

where p_{ij} denotes the probability that a_i can be substituted by a_j; f_i, f_j denote the prior probabilities for observing the amino acids; and λ is a constant. We compute f_i and f_j from the database, and then reconstruct $p_{ij} = f_i f_j e^{\lambda B_i}$. Next, we define the pair-wise amino acid cost matrix as $C[i][j] = 1 - \frac{p_{ij}}{p_{ii}}$, which ensures that the diagonal entries are $C[i][i] = 0$.

Results: Figure 8 shows the time taken for enumerating approximate maximal patterns for different values of α (with fixed $minsup = 20$). The plots show the time for random walks with and without the label pruning. It can be seen that by using the label-based pruning the time for random walks reduces significantly (by over 100%). As expected, the time increases as the values of α increases, since the number of isomorphisms clearly increases for a more relaxed (larger) cost threshold. When $\alpha = 0.01$, the patterns are exact as $C[i][j] > \alpha \ , \forall i \neq j$.

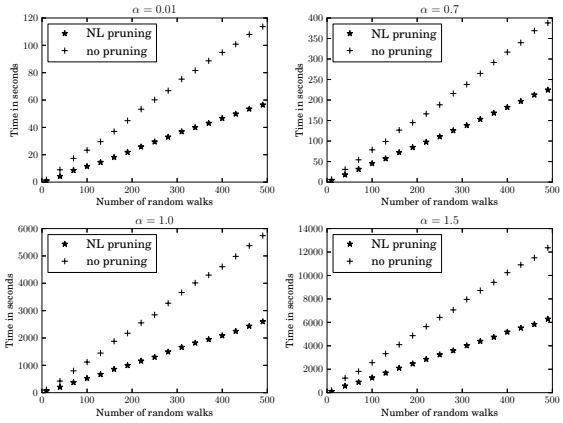


Figure 8: SCOP: Effect of α (increasing in clockwise order from top right)

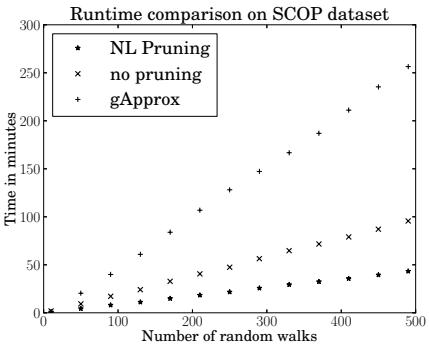


Figure 9: SCOP: Runtime comparison

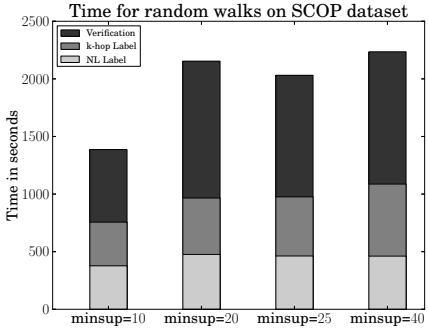


Figure 10: SCOP: Effect of *minsup*

Figure 9 compares the time taken to mine 500 maximal patterns from the SCOP dataset using the NL label algorithm and two naive isomorphism enumerating algorithms. The value of *minsup* is 15 and the threshold α is chosen as 0.7. The *no pruning* algorithm computes the representative sets from the candidate representative sets directly using the verification procedure described in section 4.3. The *gApprox* algorithm is based on [**?**] and stores all isomorphisms during the course of enumerating a maximal pattern. For each candidate pattern, it computes the isomorphisms from the isomorphisms of the frequent pattern from which the candi-

date pattern is generated. It can be see that, the run time for the NL based algorithm is significantly less compared to the naive enumeration algorithms as it prunes invalid candidates without performing an expensive verification procedure or storing potentially an expensive number of isomorphims.

Figure 10 shows the time taken for $K = 500$ random walks for various values of *minsup*, but with a fixed $\alpha = 0.7$. The bar plot shows time spent in k-hop matching (Hops), NL matching (Neighbors) and pattern verification (Enumeration). In general, the time increases as the *minsup* increases because the representative sets $R(u)$ become larger. However, there is no fixed trend as the total time depends on the regions of pattern space that the random walk explores.

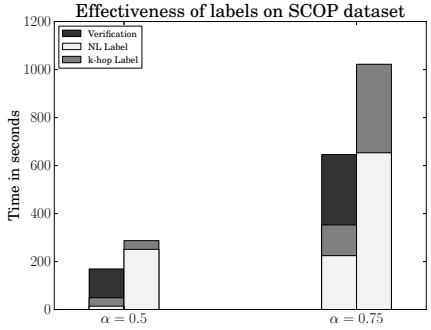


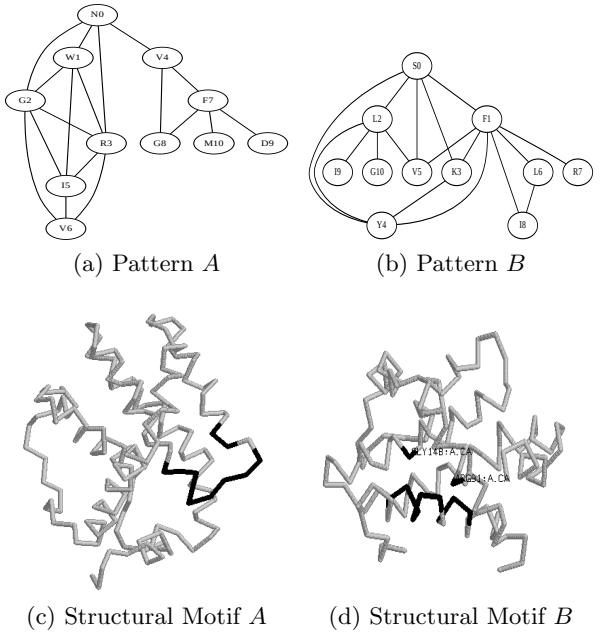
Figure 11: SCOP: Effectiveness of labels

Figure 11 compares the effectiveness of the NL label and k-hop label for different values of the threshold α on the SCOP dataset. For each value of α, the left bar shows the time with NL label, whereas the right bar shows the time using only the k-hop label. The NL label clearly reduces the time taken. In fact, it reduces the time for both the k-hop matching and the pattern verification steps, since NL is very effective in pruning the representative set. This effect is best seen for $\alpha = 0.75$, where the total time for the enumeration reduces even though matching the neighbors takes more time compared to k-hop matching. This shows the effectiveness of the NL label versus k-hop label in isolation.

Example Patterns: Figure 12 shows examples of approximate protein graph patterns and their corresponding 3D structure extracted from the SCOP dataset. For example, the graph in 12a appears in only one of the families. This pattern occurs in 18 of the 20 members, and the structure of one its occurrences, in protein PDB:1YSW, is shown in 12c. The common motif corresponds to the black colored amino acids. Another approximate pattern is shown in 12b, and its structure in PDB:1R2E is shown in 12d; it has support 19. It is important to note that the cost of this isomorphism is $C(\phi) = 0.4541$, indicating that exact isomorphism cannot find the motif.

6.3 Protein-Protein Interaction Network (PPI)

We ran experiments on a yeast (Saccharomyces cerevisiae) PPI network. The list of interacting proteins for yeast was downloaded from the DIP database (`http://dip.doe-mbi.ucla.edu`). As seen in Table **??**, the PPI network has 4950 proteins and 16,515 interactions. Unlike the other datasets, each node in the PPI network essentially has a unique label, which is the protein name.



(a) Pattern A (b) Pattern B

(c) Structural Motif A (d) Structural Motif B

Figure 12: SCOP: Approximate Graph Patterns and their Structures

Cost Matrix: To construct the cost matrix for the protein network we consider the similarity between the protein sequences for any two adjacent nodes. Sequence similarity is obtained via the BLAST alignment score [**?**], that returns the expected value (E-value) of the match. A low E-value implies high similarity, thus we create a binary cost matrix between the proteins by setting $C[p_i][p_j] = 0$ iff the proteins p_i and p_j have high similarity, i.e., iff $E - value(p_i, p_j) \leq \epsilon$. We empirically set $\epsilon = 0.003$.

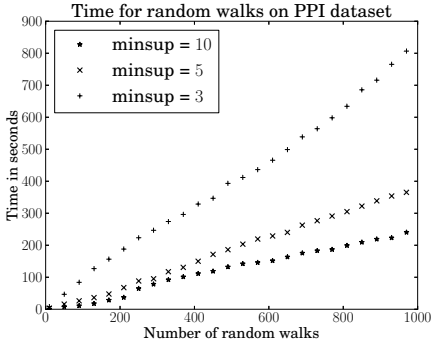
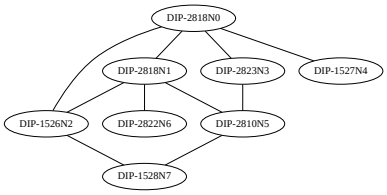


Figure 13: PPI: Time for different values of *minsup*

Results: Figure 13 shows the time for random walks in the yeast PPI network for different values of *minsup*. It can be seen that the time for random walks decreases as the support value increases. One of the differences for the PPI graph is that we do not utilize the k-hop labels. The complexity of matching the k-hop labels depends on the number of literals in the k-hop label. As each label(protein) is unique in the PPI graph, the number of literals in the k-hop label of a vertex v in a PPI network is equal to the number of vertices reachable in k-hops. This increases the run time for the k-

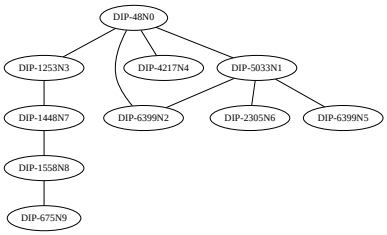
hop label matching. Therefore, for mining PPI networks we use only the NL labels.



(a) Pattern A

GO Terms	Description
BP:0051603	proteolysis involved in cellular protein catabolic process
MF:0004298	threonine-type endopeptidase activity
CC:0034515	proteasome storage granule

(b) GO Terms for A



(c) Pattern B

GO Terms	Description
BP:0004674	protein serine/threonine kinase activity
MF:0016301	kinase activity
MF:0005524	ATP binding

(d) GO Terms for B

Figure 14: Approximate PPI Patterns and GO Enrichment

Example Patterns: Figures 14a and 14c show two of the mined maximal frequent approximate patterns (using $minsup = 5$). The proteins are labeled with their DIP identifiers (e.g., DIP-2818N); the last number in the label is just a sequential node id. It is worth emphasizing that exact subgraph isomorphism would not yield any patterns in this dataset, since each label is unique. However, since we allow a protein to be replaced by a similar protein via the cost matrix $\mathbf{M}$, we obtain interesting approximate patterns. To judge the quality of the mined patterns we use the gene ontology (GO; `www.geneontology.org`), which comprises three structured, controlled vocabularies (ontologies) that describe gene products in terms of their associated biological processes (BP), molecular functions (MF), and cellular components (CC). For each of the mined approximate patterns we obtain the set of all the GO terms common to all proteins in the pattern. This serves as an external validation of the mined results, since common terms imply meaningful biological relationships among the proteins. Figure 14b shows the common GO terms for pattern A. This subgraph comprises proteins involved in proteolysis as the biological process, i.e., they act as enzymes that lead to the breakdown of other proteins into amino acids. Their molecular function is endopeptidase activity, i.e., breakdown of peptide bonds of non-terminal amino acids, in particular the amino acid Threonine. These proteins are located in the proteasome storage granule, and most likely comprise a protein complex (proteasome) – a molecular machine – that digests proteins into amino acids. The common GO terms for pattern B in Figure 14d indicate that the proteins function as Kinases, proteins that are responsible for adding a phosphate to an amino acid. The biological process is Phosphorylation, the post-translational modification of proteins corresponding to adding a phosphate, in particular modifying amino acids Serine and Threonine.

7. DISCUSSIONS AND CONCLUSIONS

We presented an effective approach to mine approximate frequent subgraph patterns from a single large graph database in the presence of a label cost matrix.

There are two main parameters in our method: K, the number of random walks, and α the cost threshold. The value of K is directly proportional to the number of maximal approximate patterns we desire, and is relatively easy to set. On the other hand, choosing an appropriate value of α is very important as it affects the quality of patterns mined. Depending on the application domain and the purpose of the graph mining, let t be the number of vertices in the pattern for which we allow label mismatches in the subgraph isomorphism. One reasonable value of α is $t \times IMQ$ where IMQ is the inter-quartile mean i.e., the mean of the entries between the first quartile (25^{th} percentile) and the third quartile (75^{th} percentile) of the entries in the cost matrix arranged in sorted order. t can be chosen by first enumerating maximal patterns with $\alpha = 0$ and computing the average size m of the maximal patterns mined from the graph. The value of t then is a fraction of the average size m. Care has to be taken not to choose a very large α as it leads to patterns of poor quality and also increases the run time of the algorithm significantly as can be seen in the Figure 8.

In terms of future work, we plan to increase the efficiency of our method by exploiting parallelism. Obviously different walks can be carried out in parallel. However, more interesting is the parallelization of the approximate isomorphism generation and label-based pruning steps, including verification. We also want to explore the idea of label based pruning for more general definitions of approximate isomorphism including edge mismatches.