Discriminative Subgraph Mining for Protein Classification

Ning Jin
University of North Carolina at Chapel Hill, USA
Calvin Young
University of North Carolina at Chapel Hill, USA
Wei Wang
University of North Carolina at Chapel Hill, USA

ABSTRACT

Protein classification can be performed by representing 3-D protein structures by graphs and then classifying the corresponding graphs. One effective way to classify such graphs is to use frequent subgraph patterns as features. However, the effectiveness of using subgraph patterns in graph classification is often hampered by the huge search space of subgraph patterns. In this paper, we present two efficient discriminative subgraph mining algorithms: COM and GAIA. These two algorithms directly search for discriminative subgraph patterns rather than frequent subgraph patterns which can be used to generate classification rules. Experimental results show that COM and GAIA can achieve high classification accuracy and runtime efficiency. Additionally, they find substructures that are very close to the proteins' actual active sites.

Keywords: discriminative subgraph mining, protein classification

INTRODUCTION

3-D protein structures can be represented as graphs by replacing each amino acid with a node labeled with the amino acid type and connecting two nodes with an edge labeled with the distance between the two corresponding amino acids if the distance is not too long. Such graph representations preserve important structural information of the proteins and thus a classification of these graphs is in effect a protein classification based on their structures. Performing graph classification by hand is computationally intractable because of the complexity of graphs and the rapidly increasing amount of structural data, so much attention has been devoted to developing graph classification methods.

One solution to graph classification is to use frequent subgraph patterns as graph features and represents each graph as a vector of features. Thus, the problem of graph classification converts to classification of high dimensional data points and many existing generic classification algorithms can be applied. However, one of the major drawbacks of this approach is that when the frequency threshold is low, the number of features may be so large that no existing frequent subgraph mining algorithm can enumerate them with a reasonable amount of computational resource, which is due to the fact that the number of subgraphs is exponential to the number of nodes and

edges in graphs. Using high frequency threshold can lead to significant reduction in the number of frequent subgraph patterns, but the discriminative subgraph patterns with lower frequencies than the threshold will be omitted.

To overcome this problem, many approaches have been proposed to mine only discriminative subgraph patterns as features instead of frequent subgraph patterns because only the discriminative subgraph patterns are of use in generating graph classifiers and the number of discriminative subgraph patterns is considerably less than the number of frequent subgraph patterns. SubdueCL (Gonzalez et al., 2002) uses beam search to look for the most discriminative subgraph pattern iteratively until each positive graph can be covered by at least one subgraph pattern. However, its efficiency is low for it calculates subgraph frequency by computing subgraph-isomorphism, which is an NP-complete problem. In (Kudo et al., 2004), the authors integrate the AdaBoost algorithm with gSpan (Yan and Han, 2002), an efficient frequent subgraph mining algorithm. They adapt gSpan to implement a branch-and-bound search to find the subgraph feature with the highest gain. Leap (Yan et al., 2008) is another algorithm developed to mine the most discriminative subgraph pattern. Leap uses two novel mining techniques, structural proximity pruning and frequency-descending mining, to excel the aforementioned branch-and-bound search algorithm. The structural proximity pruning takes into account the fact that subgraph patterns that share a large subgraph have similar discrimination power and uses it to calculate a tight upper-bound of their discrimination scores. The frequency-descending mining takes advantage of the observation that subgraphs with higher frequency are more likely to be discriminative and thus may reach the optimal solution faster. Another algorithm, gPLS (Saigo et al., 2008), uses subgraph pattern mining to find features for partial least squares regression so that the mining process only searches for patterns that can improve the accuracy of the resulting classifier. The algorithms mentioned above are not efficient because they find one discriminative subgraph pattern at a time and must be invoked repeatedly to find enough discriminative subgraph patterns to generate graph classifiers, thereby limiting their ability to process large datasets.

CORK (Thoma et al., 2009) uses the number of correspondences to measure the discrimination power of a subgraph pattern and thereby achieves a theoretically near-optimal solution. Given a set of subgraph patterns, the number of correspondences for a set of subgraph patterns is the total number of pairs of graphs that these subgraphs cannot discriminate. Although the solution is near-optimal in terms of minimizing the number of correspondences, using the number of correspondences as the measurement of discrimination power may be problematic because subgraphs of significantly different discrimination power can have the same number of correspondences. GraphSig (Ranu and Singh, 2009) tackles the problem of explosive number of subgraph patterns from a different perspective. Instead of looking for discriminative subgraph patterns in the whole dataset, it clusters similar graphs into small groups and searches for frequent subgraph patterns within each group. First, it uses random walk to generate feature vectors for all graphs and groups together graphs with similar feature vectors. Discriminative subgraph patterns with low frequencies among all the graphs can therefore have high frequencies within these groups. Then, graphSig uses relatively high frequency thresholds to mine frequent subgraph patterns from each group. Its

efficiency completely depends on the quality of clustering in the first step. If it fails to group similar graphs together because the feature vectors generated by random walk cannot capture the characteristics of the graphs, it will still miss discriminative subgraph patterns of low frequency. GraphSig significantly outperforms Leap (Yan et al., 2008) and the optimal assignment kernel method (Fröhlich et al., 2005) in terms of runtime efficiency and accuracy as is reported in (Ranu and Singh, 2009).

In addition to subgraph pattern mining, another direction for graph classification is using graph kernel methods. In (Kashima and Koyanagi, 2002), feature vectors are composed of the counts of subgraph patterns. In (Gärtner et al., 2003) direct product kernel is defined as the count of the identical walks that can be taken in both graphs. In (Kashima et al., 2003), marginalized kernel is defined as the inner product of the count vectors averaged over all possible paths produced by random walks on graphs. An efficient computation can be performed by solving simultaneous linear equations. (Fröhlich et al., 2005) presents a graph kernel named optimal assignment kernel for chemical compound graphs. The idea is to compute an optimal assignment from nodes in one graph to those in another graph such that the overall matching score is maximized. (Smalter et al., 2008) defines a diffusion kernel for graph classification. They map frequent subgraphs to the graphs and then use "pattern diffusion" to assign values to nodes in the graphs. After "pattern diffusion," a graph alignment algorithm is used to compute the inner product of two graphs.

Generally speaking, when comparing similarity of two graphs, most graph kernel methods concentrate on global structural similarity while subgraph-based methods focus more on local similarity. As a result, subgraph-based methods are more suitable for protein structure classification than most graph kernel methods because proteins with the same function usually have certain highly conserved substructures (Huan et al., 2004; Bandyopadhyay et al., 2006) containing the active sites that most proteins without the function do not have. The conserved substructures are typically very small compared with the whole protein structures and thereby have little impact on the result of graph kernel computation, so they can only be identified by subgraph-based methods.

In this paper, we present two efficient discriminative subgraph pattern mining algorithms for protein classification: COM and GAIA. COM (Jin et al., 2009) uses a heuristic subgraph exploration order to find discriminative patterns faster. It also takes into account co-occurrences of subgraph patterns to boost the discrimination power of features. COM significantly outperforms Leap and graphSig in speed and produces higher accuracy in classifying protein structures. GAIA (Jin et al., 2010) is the most recent development in efficient graph classification. It utilizes evolutionary computation to search for discriminative subgraph patterns and achieves superior performance in terms of classification accuracy and runtime efficiency.

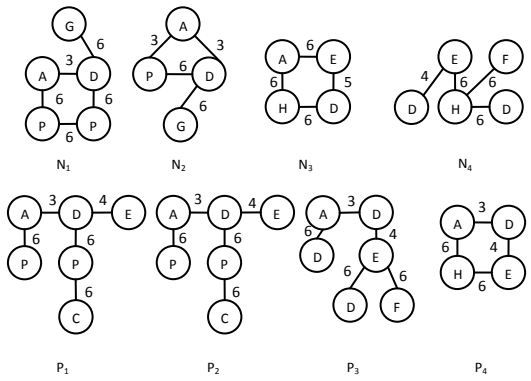
The remainder of this paper is organized as follows. In Section 2, we introduce the preliminaries about subgraph pattern mining. In Section 3, we analyze the problem of graph classification. Section 4 and Section 5 describe how COM and GAIA search for discriminative subgraph patterns and use them to generate graph classifiers, respectively. Experimental results are shown and analyzed in Section 6. Section 7 concludes the paper.

2. PRELIMINARIES

DEFINITION 1 (Graph). A graph is denoted by $g = (V, E)$, where V is a set of nodes (vertices) and E is a set of edges connecting the nodes. Both nodes and edges may have labels.

Figure 1 shows an example of two sets of graphs: $\{N_1, N_2, N_3, N_4\}$ and $\{P_1, P_2, P_3, P_4\}$. The node set of a graph g is denoted as $V(g)$ and the edge set of g is denoted as $E(g)$. For example, in graph N_3, the node set $V(N_3)$ is $\{A, D, E, H\}$ and the edge set $E(N_3)$ is $\{A6H, A6E, D5E, D6H\}$.

Figure 1: an example of graphs



DEFINITION 2 (Subgraph). The label of a node u is denoted as $l(u)$ and the label of an edge (u, v) is denoted as $l((u, v))$. For two graphs g and g', if there is an injection $f: V(g) \rightarrow V(g')$, such that for any node v in $V(g)$, $l(u) = l(f(u))$ and for any edge (u, v) in $E(g)$, $l((u, v)) = l((f(u), f(v)))$, then g is a subgraph of g' and g' is a supergraph of g, or g' supports g.

For example, in Figure 1, $A3D4E$ is a subgraph of graph P_1.

DEFINITION 3 (Frequency). Given a graph set G, for a subgraph p, let $S = \{g \mid g \text{ is in } G \text{ and } g \text{ supports } p\}$, then the *frequency* of g is $|S| / |G|$.

For example, in the graph set $\{N_1, N_2, N_3, N_4\}$ in Figure 1, the supporting set of $A3D$ is $\{N_1, N_2\}$ and its frequency is 2/4 = 0.5.

DEFINITION 4 (Positive frequency and negative frequency). Given two sets of graphs, namely the positive set G_p and the negative set G_n, the positive (negative) frequency of a subgraph pattern p is its frequency in the positive (negative) graph set, denoted as $pfreq(p)$ ($nfreq(p)$).

In protein classification, the positive set is usually composed of the graphs representing proteins that exhibit a certain function/attribute and the negative set consists of graphs of proteins that don't exhibit the function/attribute. In most cases, the negative set is much larger than the positive set for only functions/attributes that most proteins do not have are interesting to users.

In Figure 1, let $\{N_1, N_2, N_3, N_4\}$ be the negative set and $\{P_1, P_2, P_3, P_4\}$ be the positive set. The positive frequency of $A3D$ is 1.0 and its negative frequency is 0.5.

DEFINITION 5 (Connectivity). For two nodes v_0 and v_n in graph g, if there exists a sequence of nodes $v_0, v_1, v_2, \ldots, v_n$ such that there is an edge in g connecting v_i and v_{i+1}, for any i, $0 \leq i < n$, then v_0 and v_n are connected. For a graph g, if every two nodes in g are connected, then g is connected.

All graphs in Figure 1 are connected graphs. Most subgraph mining algorithms look for connected subgraph patterns only, but COM is able to find disconnected subgraph patterns by assembling connected subgraph patterns into co-occurrences, which is defined as follows.

DEFINITION 6 (Co-occurrence). A co-occurrence of subgraphs, C, is defined as a set of subgraph(s) that occur concurrently in a set of graphs. A co-occurrence can consist of only one subgraph pattern.

For example, in the graph set $\{P_1, P_2, P_3, P_4\}$ in Figure 1, $\{A3D, D4E\}$ is a co-occurrence and its frequency is 4/4 = 1.0. $\{A3D\}$ is also a co-occurrence and its frequency is 1.0.

3. PROBLEM ANALYSIS

The input of a graph classification problem is composed of two sets of graphs, denoted as the positive set G_p and the negative set G_n. Each graph g in the input has a label $L(g)$ to indicate which graph set it belongs to. If g is in the positive set, $L(g)$ is 1; otherwise, $L(g)$ is -1.

The goal of graph classification is to learn a classifier function R: $G_p \cup G_n \rightarrow \{1, -1\}$. The classifier R is then used to predict labels of unlabeled graphs. In protein classification, the labels indicate whether the proteins have a certain function/attribute. Graph classification algorithms learn classifiers based on labeled proteins and the classifiers are applied to protein function/attribute prediction. To evaluate the accuracy of a graph classification algorithm, the graph set $G = G_p \cup G_n$ is usually partitioned into $G_{training}$ and G_{test}, where $G = G_{training} \cup G_{test}$ and $G_{training} \cap G_{test} = \emptyset$. The classification algorithm learns a classifier R solely based on $G_{training}$ and the accuracy of R can be measured by its normalized accuracy in G_{test}.

DEFINITION 7 (Sensitivity, Specificity and Normalized Accuracy).

$$Sensitivity(R) = \frac{|\{g_i \mid L(g_i) = 1 \text{ and } R(g_i) = 1\}|}{|\{g_i \mid L(g_i) = 1\}|}$$

$$Specificity(R) = \frac{|\{g_i \mid L(g_i) = -1 \text{ and } R(g_i) = -1\}|}{|\{g_i \mid L(g_i) = -1\}|}$$

$$Normalized\ accuracy(R) = \frac{Sensitivity + Specificity}{2}$$

We use normalized accuracy rather than raw accuracy to prevent the measurement from being heavily biased by imbalance between the sizes of positive set and negative set. For example, if there are 100 negative graphs and only 10 positive graphs, a classifier R that always predicts a graph as negative has a high accuracy equal to 100/110=0.91 while R is obviously a poor classifier. On the contrary, using normalized accuracy can show the poor performance of R as its normalized accuracy is (1.0+0.0)/2 = 0.50.

Subgraph patterns are of great use in graph classification when there exists a strong correlation between the existence of certain subgraph patterns and the labels of graphs. Proteins are typically a good example of this because whether a protein exhibits a certain function/attribute depends heavily on whether it has the active site for this function/attribute. However, there may exist a huge number of subgraph patterns, of which only few may be useful in generating classifiers. Subgraph patterns are of interest only if they are discriminative, which means they occur in many graphs in one set but rarely appear in graphs in the other set. In protein classification, discriminative subgraph patterns usually occur frequently in the positive set and infrequently in the negative set because the negative set is so diverse that few frequent subgraph patterns can be found in it. Therefore, both COM and GAIA focus on searching for discriminative patterns in the positive set. However, it is straightforward to extend them to mine discriminative patterns in the negative set if necessary.

The discrimination power a pattern p is can be quantitatively measured by an objective function of its positive and negative frequencies. In this paper, we define the objective function as follows.

DEFINITION 8 (Discrimination score).

$$score(p) = \log\left(\frac{pfreq(p)}{nfreq(p)}\right)$$

When the negative frequency of p is zero, we replace its negative frequency with a very small value ε to avoid being divided by zero.

For example, in Figure 1, the score of pattern *A3D* is log(2) and the score of pattern *A3D4E* is log(1/ε).

The goal of COM and GAIA is to find a set of subgraph patterns with high

discrimination scores and use them to generate graph classifiers.

4. COM: DISCRIMINATIVE SUBGRAPH MINING BASED ON PATTERN CO-OCCURRENCES

Given a graph set $G = G_p \cup G_n$, COM searches for co-occurrences whose positive frequencies are no less than t_p and whose negative frequencies are no greater than t_n, where t_p and t_n are user specified parameters. The intuition is to find co-occurrences that are frequent enough in the positive set and infrequent enough in the negative set. The reasons why COM looks for co-occurrences instead of individual subgraph patterns are as follows. First of all, sometimes the discriminative substructures are so flexible that their conformations vary in each graph and thereby can hardly be detected by subgraph mining algorithms. If these flexible substructures can be decomposed into several smaller parts and each part has the same conformation in graphs, then the flexible substructures can be detected as co-occurrences by COM. Secondly, extending small subgraph patterns into large subgraph patterns can be a very expensive operation. Using co-occurrences of small patterns to approximate large patterns can save the expensive pattern extension operation. For example, in Figure 1, subgraph pattern *A3D4E* can be approximated by co-occurrence {*A3D*, *D4E*}. Although the co-occurrence approximation does not include as much structural information as the large pattern, *A3D4E* and {*A3D*, *D4E*} have the same supporting graphs and thus the same discrimination score. Therefore, this approximation provides the same amount of information for graph classification.

To find discriminative co-occurrences, COM explores candidate subgraph pattern space in a heuristic order and assembles individual subgraph patterns into co-occurrences if the assembly can improve discrimination score.

COM terminates when it finds a set of co-occurrences such that these co-occurrences satisfy the frequency requirements and each positive graph supports at least one co-occurrence in the set. The rationale for this termination condition is that the number of co-occurrences that satisfy the frequency requirements is still very large and these co-occurrences are redundant in that many of them are supported by the same or similar set of graphs. For example, in Figure 1, {*A3D4E*} and {*A3D*, *D4E*} are supported by the same set of graphs and thus they are essentially the same feature when only the supporting set is concerned.

4.1 Pattern Exploration Order

With a scoring function, we can rank all subgraph patterns by their discrimination scores. We want to organize the order in which subgraph patterns are visited so as to increase the probability that we visit patterns with higher score ranks earlier than those with lower score ranks because co-occurrences composed of subgraph patterns with higher ranks are more likely to satisfy the frequency requirements.

We take advantage of the following observation: let p be a pattern p' be the parent pattern from which p is extended, the score rank of p is correlated with the value of $\Delta(p) = score(p) - score(p')$. For patterns with two nodes, we set their Δ values equal to their scores $score(p)$.

Therefore, when we explore the pattern space, we first enumerate all patterns with 2 nodes as candidates and insert them into a heap structure with the candidate having the highest Δ value at the top. Ties are broken by favoring higher positive frequency. Then we always take the pattern at the top of the heap and generate all of its super-patterns with one more edge by performing the CAM

extension operation (Huan et al., 2003). We insert new patterns into the heap structure. In this way, we are able to visit patterns with high score ranks early.

4.2 Co-occurrence Generation

COM uses the resulting co-occurrences to generate classification rules. Each resulting co-occurrence C leads to a classification rule: if a graph g supports all the subgraph patterns in $C \rightarrow L(g) = 1$. The final graph classifier produced by COM is composed of a set of such classification rules. Given a graph g, if any rule in the classifier indicates $L(g)$ is 1 (positive), the classifier labels g as 1 (positive); otherwise, g is labeled as -1 (negative).

The goal of COM is to find a set of co-occurrences to maximize the number of graphs that can be classified correctly, where each co-occurrence has positive frequency no less than t_p and negative frequency no greater than t_n, where t_p and t_n are user-specified parameters to describe how discriminative users require the co-occurrences to be.

This problem can be proved to be equivalent to the set cover problem and is therefore NP complete. It is intractable to find an optimal solution in the enormous pattern space. Therefore, we adopt a greedy approach for co-occurrence generation.

Let the candidate co-occurrence set be T_t and the resulting set be T. The algorithm explores the pattern space with the aforementioned heuristic order and whenever it comes to a new pattern p that has not been processed before, if there exists one positive supporting graph of p that does not support, or in other words is not covered by, any resulting co-occurrence generated so far, the algorithm generates a new candidate co-occurrence containing only p and examines the possibility of merging this new co-occurrence into one of the existing candidate co-occurrences. Given a new pattern p and a candidate co-occurrence C_t, $\Delta(p, C_t) = score(C_t \cup \{p\}) - score(p)$. Pattern p is to be inserted into candidate co-occurrence C', $C' = \text{argmax}_{Ct \text{ is in } T'}(\Delta(p, C_t))$, $\Delta(p, C_t) \geq 0$. If there are patterns in C' whose supporting sets are supersets of the supporting set of p, then inclusion of p into C' will make these patterns redundant. These patterns will be removed from C' when p is inserted. Then, for either the newly generated co-occurrence $\{p\}$ or the updated C', if it has $pfreq \geq t_p$ and $nfreq \leq t_n$ and it can cover at least one positive graph that is not covered by any co-occurrence in T, it will be removed from T_t and inserted into T. The algorithm terminates either when all patterns are explored or when all positive graphs are covered by some resulting co-occurrences. Although in the worst case the algorithm is still exhaustive, experiments show that it is time efficient in practice.

5. GAIA: DISCRIMINATIVE SUBGRAPH MINING USING EVOLUTIONARY COMPUTATION

GAIA applies evolutionary computation, which is a randomized searching strategy for optimal solution and simulates biological evolution, to look for discriminative subgraph patterns. GAIA does not consider co-occurrences of subgraphs because the individual subgraph patterns found by GAIA are so discriminative that their discrimination scores can hardly be improved by assembling them into co-occurrences. In addition, although GAIA also uses patterns to generate classification rules as graph classifiers, unlike COM, GAIA adopts a feature selection process after pattern mining to maximize the normalized accuracy of the classifiers in $G_{training}$. These differences enable GAIA to outperform COM in terms of classification accuracy and runtime efficiency.

5.1 Evolution of Subgraph Patterns

Evolutionary computation can be viewed as a generic search process for solutions of high quality or fitness, which begins with a set of sample points in the search space and gradually biases to regions of high fitness. In the problem of discriminative subgraph pattern mining, we use discrimination score as the fitness function to measure the potential discrimination power of a pattern and larger patterns that can be generated by subgraph extension. As a result, our evolutionary search process here is directed toward subgraph patterns with high discrimination power.

For each graph *g* in the positive graph set G_p, we store a representative subgraph pattern and a list of up to *s* candidate subgraph patterns, where *s* is a user-specified parameter. The representative pattern is the subgraph pattern with the highest discrimination score among all patterns that are subgraphs of *g* found during pattern evolution. The candidate lists are initialized with one-edge patterns for they are the most primitive candidates.

The motivation of the design of this framework is to create selection pressure which can significantly speed up the convergence of evolutionary search. When the candidate lists are not large enough to hold all patterns that can be found in positive graphs, one resource that candidate patterns need to compete for is a slot in candidate lists and candidate patterns that fail to find a slot in candidate lists are eliminated from further consideration. Generally speaking, the larger the candidate lists are, the less selection pressure there is and thereby the more patterns are considered in the search. When the candidate lists are infinitely large, the search process becomes an exhaustive search. How the competition is carried out will be discussed later in this subsection.

Another resource that candidate patterns compete for is the opportunity to extend or, analogous to biological evolution, to produce offspring. All subgraph pattern mining algorithms start with small subgraph patterns and then extend them into larger patterns. However, pattern extension is a costly operation and not every pattern extension leads to a discriminative pattern. In an evolutionary search process, candidate patterns compete for the opportunity of pattern extension according to their fitness, which enables the search process to focus on candidate patterns that are more likely to lead to discriminative patterns. This competition is implemented as follows.

Each candidate pattern currently in the candidate lists has a non-zero probability of being selected for pattern extension. To perform pattern evolution, GAIA runs for *n* iterations, where *n* is a parameter set by the user. In each iteration, GAIA selects one pattern from each candidate list to extend. The probability for pattern *p* in the candidate list of graph *g* to be selected for extension is proportional to the score of *p* and is calculated as follows:

$$Probability(pattern\ p\ gets\ extended) = \frac{score(p)}{\sum_{p' is\ in\ the\ candidate\ list\ of\ g} score(p')}$$

The probability is always between 0 and 1 if only patterns with positive scores are allowed in candidate lists. Note that when *s* = 1, each candidate list only holds 1 pattern. The probability of this pattern being selected for extension is 1. When *s* > 1, multiple patterns may be held in a candidate list. A random number generator is used to determine which pattern is selected for extension according to their probabilities.

For an extension operation of pattern *p*, GAIA generates a pattern set *X(p)* and each pattern *p'* in *X(p)* has one new edge attached to *p*. Unlike many previous subgraph pattern mining algorithms that only extend patterns with certain types of edges in order to

efficiently maintain their canonical codes, GAIA considers all one-edge extensions of pattern *p* that occur in the positive graphs. This difference enables GAIA to explore the candidate pattern space in any direction that appears promising.

In most cases, an extension operation on one pattern generates many new patterns and as a result the number of patterns found by the algorithm grows. Sooner or later the number of patterns will exceed the number of available positions in the candidate lists. It is also possible that the number of one-edge patterns already exceeds the number of available positions in the candidate lists at the very beginning if *s* is small. Therefore some rules are needed to determine which patterns should survive in the candidate lists and which candidate list they should dwell in.

First, a pattern that has already been extended should not stay in the candidate lists any longer because it has served its role in producing new patterns.

Second, some pattern in the candidate list may migrate to the candidate list of another graph if such migration will increase its chance of survival. Let *p* be the candidate pattern for migration and *S*(*p*) be the set of positive graphs supporting *p*. Let *g* be the graph in *S*(*p*) which has the lowest value of $\sum_{p' is\ in\ the\ candidate\ list\ of\ g} score(p')$. Pattern *p* will migrate to the candidate list of *g*. The rationale for this pattern migration is that if a pattern wants to survive then it should go to a candidate list with the least fierce competition. In GAIA, the fierceness of competition of a candidate list is measured by the sum of scores of patterns in the list.

If the candidate list of *g* still has vacant positions, then *p* can move into one vacant position directly. However, if the candidate list is already full, then *p* has to compete with the patterns that already exist in the list. The score of *p* is compared against the score of a pattern *p'*, which is randomly selected with probability 1/*s* from the candidate list. If the score of *p* is higher, then *p'* is discarded from the list and *p* takes over the position of *p'*; otherwise, *p* is eliminated from further consideration.

Again, the exhaustive extension operation is of great importance to allow pattern competition and elimination. When we eliminate a pattern *p*, we lose not only this pattern but also the patterns generated by extending *p*. In previous subgraph pattern mining algorithms, such as gSpan (Yan and Han, 2002) and FFSM (Huan et al., 2003), a pattern *p* can only be extended from one of its subpatterns, *p'*. If *p'* is lost, then the algorithms will never find *p*. These algorithms surely lose many patterns as a result of pattern elimination, some of which are discriminative. But in GAIA, eliminating *p'* does not necessarily lead to the loss of *p* because the exhaustive extension operation allows *p* to be extended from many different patterns. As a result, the risk of missing discriminative patterns is much lower than other subgraph mining algorithms.

5.2 Generating Classification Rules

GAIA uses the representative subgraph patterns to generate classification rules. Each representative pattern *p* corresponds to one classification rule: if a graph *g* supports pattern $p \rightarrow L(g) = 1$, which is very similar to the rules generated by COM. However, unlike COM's graph classifiers consisting of all resulting co-occurrences, GAIA has a feature selection process to choose a subset of all representative patterns to compose its graph classifiers in order to maximize the normalized accuracy of classifiers in $G_{training}$.

Given the representative subgraph patterns from pattern evolution, GAIA's rule generation process selects patterns to compose the graph classifier by sequential coverage. First GAIA sorts the

representative patterns by their scores in decreasing order. Then it traverses all representative patterns in the sorted order (that is GAIA visits patterns with higher scores first). For each representative pattern *p*, GAIA evaluates whether inclusion of a new classification rule "if a graph *g* supports pattern $p \rightarrow L(g) = 1$" in the classifier can increase the normalized accuracy of classification of the training set. The new rule "if a graph *g* supports pattern $p \rightarrow L(g) = 1$" is to be included in the classifier if and only if the normalized accuracy increases. The rule generation process terminates when all representative patterns have been evaluated.

5.3 Improving Performance by Generating Consensus Model with Parallel Computing

Because GAIA is a randomized algorithm (when $s > 1$), each single run of pattern evolution may generate different representative patterns and consume varying amount of CPU time. Some runs of pattern evolution may find better representative patterns than others and thus lead to classifiers with higher normalized accuracy. Therefore, if we run many instances of pattern evolution in parallel and generate a classifier based on all representative patterns found by these instances of pattern evolution, it is very likely that we can build a better classifier than using representative patterns from one instance of pattern evolution alone. Therefore, by generating a consensus model based on many parallel instances of pattern evolution and only using the fastest instances of pattern evolution, we can improve the classification accuracy and expected response time by taking advantage of parallel computing. We choose to start classification rule generation before all instances complete because, if we wait for all the pattern evolution instances, the runtime of GAIA will be determined by the runtime of the slowest instance (The time for sequential coverage is trivial). Let *c* be the number of parallel instances of pattern evolution, which is a user specified parameter. We start generating classification rules as soon as $\lceil c/2 \rceil$ instances of pattern evolution have terminated and only use the representative patterns found by these $\lceil c/2 \rceil$ instances. The classification rule generation process is the same as described in Subsection 5.2 except that now we consider representative patterns found by $\lceil c/2 \rceil$ instances rather than only one instance.

6. EXPERIMENTS

The algorithms were implemented in C++ and compiled with g++. The experiments were performed on a 2.20 GHz dual core and 3.7 GB memory PC running Ubuntu Linux 9.10. We analyze the performance from two perspectives: runtime efficiency and normalized accuracy in protein classification. All experiments are performed with 5-fold cross-validation: each dataset is partitioned into 5 subsets with the ratio of the number of positives to the number of negatives being maintained. For each dataset, we run the algorithms 5 times and each time we use 1 subset as the test set and the rest as the training set. The classifiers are learned solely based on the test set and the average performance is reported.

The protein datasets consist of protein structures from RCSB Protein Data Bank (http://www.rcsb.org/pdb/) classified by SCOP (Structural Classification of Proteins: http://scop.mrc-lmb.cam.ac.uk/scop/). We use 16 protein datasets generated from all the large SCOP families with more than 25 members (listed in Table 1). In each dataset, protein structures in a selected family are taken as the positive set. Unless otherwise specified, we randomly select 256 other proteins (i.e., not members of the 16

families) as a common negative set used by all protein datasets. In order to remove redundancy and possible bias in graph sets, we only use proteins with pairwise sequence identity less than 90% from the culled PDB list created by Dunbrack Lab (http://dunbrack.fccc.edu/PISCES.php). To generate a protein graph, each graph node denotes an amino acid, whose location is represented by the location of its alpha carbon. We perform 3-D Almost Delaunay Tessellation (Bandyopadhyay and Snoeyink, 2004) on locations of all alpha carbons in the protein to generate the edges. Nodes are labeled with their amino acid types and edges are labeled with the distances between the alpha carbons (we discretized distances between alpha carbons into 7 bins and thereby there are 7 edge labels). We only consider edges shorter than 11.5 angstroms because amino acids have little long-distance interaction. On average, each protein graph has 250 nodes and 2700 edges.

6.1 Performance Analysis

We first compare the performance of COM (Jin et al., 2009), GAIA (Jin et al., 2010) and graphSig (Ranu and Singh, 2009). The implementation of graphSig can only process balanced datasets that have equal number of positives and negatives, so we generate balanced protein datasets, whose numbers of positives are listed in Table 1 and whose numbers of negatives are the same as the numbers of positives, for this comparison. The parameters used by the three algorithms are summarized in Table 2.

Table 1: list of selected SCOP families

SCOP ID	Family name	# of selected proteins
46463	Globins	51
47617	Glutathione S-transferase (GST)	36
48623	Vertebrate phospholipase A2	29
48942	C1 set domains	38
50514	Eukaryotic proteases	44
51012	alpha-Amylases, C-terminal beta-sheet domain	26
51487	beta-glycanases	32
51751	Tyrosine-dependent oxidoreductases	65
51800	Glyceraldehyde-3-phosphate dehydrogenase-like	34
52541	Nucleotide and nucleoside kinases	27
52592	G proteins	33
53851	Phosphate binding protein-like	32
56251	Proteasome subunits	35
56437	C-type lectin domains	38
88634	Picornaviridae-like VP	39
88854	Protein kinases, catalytic subunit	41

Table 2: summary of parameters for experiments

Algorithm	Parameters for protein datasets
GAIA	s=100, n=10, c=32
COM	t_p=30%, t_n=0%
graphSig	maxPvalue=0.1, minFreq=10%

Figure 2 shows the normalized accuracy comparison between GAIA, COM and graphSig for all 16 protein datasets. GAIA has higher accuracy than COM in 14 out of 16 datasets. On average, GAIA has a normalized accuracy 5.7% higher than that of COM. We performed a paired t-test to evaluate the statistical significance of the difference between the accuracies of GAIA and COM. The p-value for GAIA's improvement over COM in normalized accuracy is 0.008 and thereby the difference in accuracy is significant at the 0.01 level. GraphSig is not comparable in processing

protein datasets. Its average normalized accuracy is 52.5%.

Figure 3 compares the runtime performance of GAIA, COM and graphSig. On average, GAIA is ~1.2 times faster than COM and ~90 times faster than graphSig.

The poor performance of graphSig can be due to its failure to group similar graphs together in its first step. Once dissimilar graphs are grouped together, frequent subgraph patterns cannot be found to characterize the graphs in a group since they are so diverse. Even if graphSig manages to cluster similar graphs into small groups and finds subgraph patterns with relatively high frequency, the number of subgraph patterns may still be too large for support vector machine, which is recommended by the authors, to generate classifiers effectively. GraphSig has high performance in chemical compound datasets, but protein graphs are much more complex than chemical compound graphs. First of all, protein graphs have more nodes and edges than chemical compound graphs. The protein graphs used in this paper have 250 nodes and 2700 edges on average while the chemical compound graphs used in (Ranu and Singh, 2009) have ~55 nodes and ~57 edges. Secondly, protein graphs have more labels than chemical compound graphs. There are 20 node labels and 7 edge labels in protein graphs. In chemical compound graphs, although many different atom types appear in the datasets, the majority of them are C, H, N and O. In addition, most edges in chemical compound graphs are labeled as either "single bond" or "double bond" with very few exceptions. As a result, the number of subgraphs in protein graphs is much larger than the number of subgraphs in chemical compound graphs, which also leads to the much lower performance processing protein graphs than chemical compound graphs for graphSig.

Figure 2: normalized accuracy comparison for balanced protein datasets between GAIA, COM and graphSig

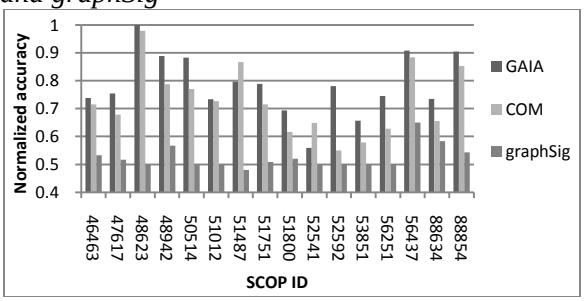
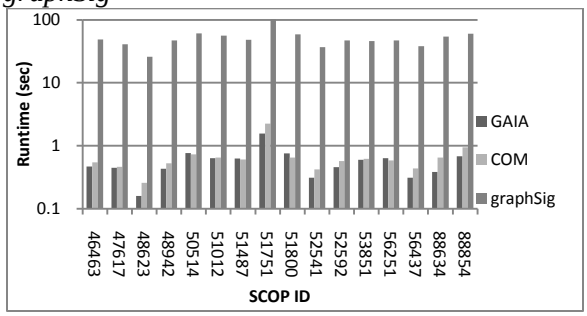


Figure 3: runtime comparison for balanced protein datasets between GAIA, COM and graphSig



We use the unbalanced protein datasets described at the beginning of this Section to provide further comparison between COM and GAIA. In most real applications there are a lot of negative graphs than positive graphs. Figure 4 and Figure 5 compare the normalized accuracy and runtime performance between GAIA and COM, respectively. Compared with using balanced protein datasets, using unbalanced protein datasets leads to 3.47% higher accuracy for GAIA and 5.47% higher accuracy for COM. The difference between accuracies of balanced datasets and unbalanced datasets is statistically significant at the 0.01 level. It demonstrates that having more negative graphs as the background simplifies the graph classification problem and leads to higher accuracy. For unbalanced protein datasets, the average normalized accuracy of GAIA is 2.7% higher than that of COM and GAIA is 1.5 times faster than COM. It can

be seen that the gap between the accuracy of GAIA and COM is closing as the graph classification problem becomes easier when more negative graphs are given.

Figure 4: normalized accuracy comparison for unbalanced protein datasets between GAIA and COM

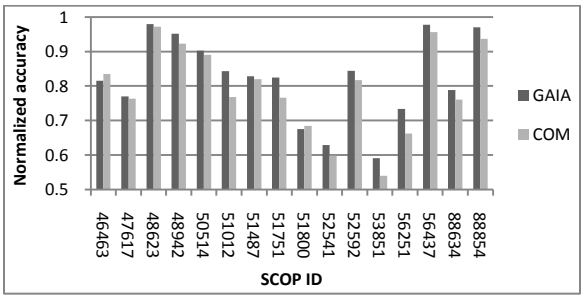
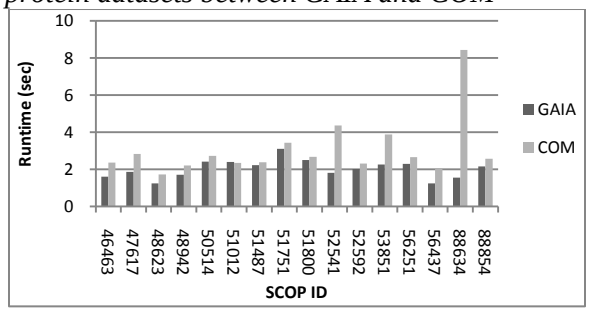


Figure 5: runtime comparison for unbalanced protein datasets between GAIA and COM



6.2 Discriminative Subgraph Patterns vs. Active Sites

In this subsection, we compare the positions of discriminative subgraph patterns and the positions of active sites. We collect the active site information from Catalytic Site Atlas (http://www.ebi.ac.uk/thornton-srv/databases/CSA/), whose active site entries are either literature-based or calculated with PSI-BLAST alignment to literature-based entries using an e value cut-off of 0.0005. We visualize the amino acids composing the discriminative subgraph patterns and active sites in protein structures by PyMOL (http://www.pymol.org/). Figures 6 through 9 show 4 examples. All protein structures are from RCSB Protein Data Bank (http://www.rcsb.org/pdb/). In these figures, the yellow spheres are the amino acids composing the discriminative subgraph patterns and the red sphere are the amino acids composing the active sites. The purple spheres are where discriminative subgraph patterns and active sites overlap.

It can be seen that the patterns found by discriminative subgraph mining algorithms without any information about active sites are close to active sites and sometimes even overlap with active sites, which demonstrates that the discriminative patterns are not only of use in protein classification but biologically meaningful as well. Discriminative patterns are not exactly the same as active sites because Catalytic Site Atlas only includes residues that are thought to be directly involved in some aspect of the reaction catalyzed by a protein while discriminative pattern mining algorithms search for residues that can discriminate certain proteins from others. In addition, GAIA and COM compute a set of patterns that can lead to an accurate classifier rather than enumerate all discriminative patterns, which causes them to miss some residues in the active sites. Another reason that discriminative patterns do not completely overlap with active sites is that some of the active sites are too small to be discriminative (e.g. the active sites in Figures 8 and Figure 9) and thereby not considered as discriminative patterns.

Figure 6: PDB 1a5iA (discriminative pattern: HIS-57, ILE-103, CYS-191, TRP-215; active site: HIS-57, ASP-102, GLY-193, SER-195, GLY-196)

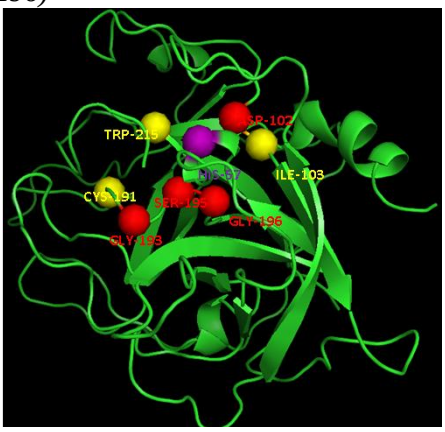


Figure 7: PDB 1aq0A (discriminative pattern: TRP-235, ASN-279, VAL-287, ASN-290; active site: GLU-232, GLU-280, LYS-283, GLU-288)

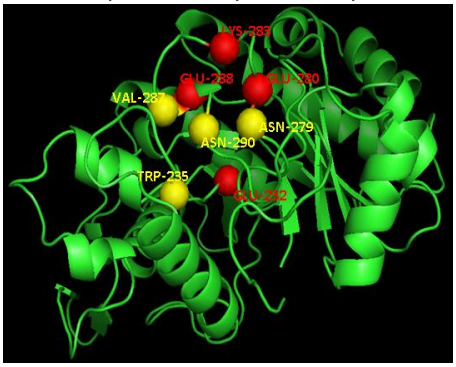


Figure 8: PDB 1ijlA (discriminative pattern: TYR-21, CYS-43, CYS-44, HIS-47; active site: GLY-29, HIS-47, ASP-89)

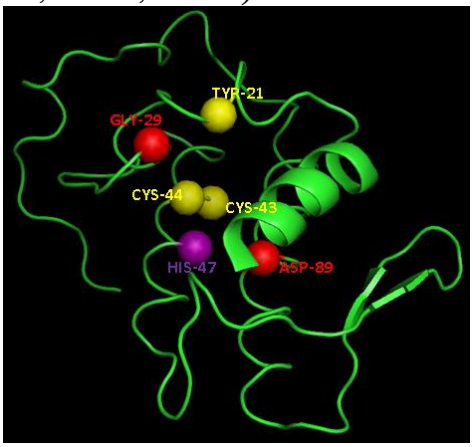
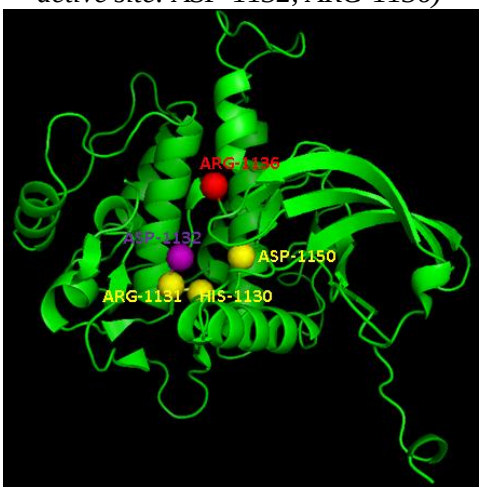


Figure 9: PDB 1ir3A (discriminative pattern: HIS-1130, ARG-1131, ASP-1132, ASP-1150; active site: ASP-1132, ARG-1136)



7. CONCLUSION

In this paper, we investigate the problem of graph classification for protein structures and introduce two discriminative subgraph mining algorithms, namely COM and GAIA, for efficient protein classification. COM uses a heuristic subgraph pattern exploration order to search the pattern space such that discriminative patterns are more likely to be detected early. In addition, it reduces the amount of pattern extension operations by approximating large subgraph patterns as co-occurrences of small patterns. GAIA utilizes evolutionary computation in discriminative subgraph pattern mining. During the pattern evolution performed by GAIA, candidate subgraph patterns compete for storage space in memory and CPU time for pattern extensions. Candidate subgraph patterns with high discrimination scores survive the competition and are used to generate graph classifiers. Experiments show that both COM and GAIA outperform graphSig, one of the state-of-the-art graph classification methods, in terms of classification accuracy and runtime

efficiency for protein classification. We also demonstrate that discriminative subgraph patterns are close to active sites in 3-D protein structures.

REFERENCES

D. Bandyopadhyay, J. Huan, J. Liu, J. Prins, J. Snoeyink, W.Wang, and A. Tropsha (2006). Structure-based function inference using protein family-specific fingerprints, *Protein Science*, vol. 15, pp. 1537-1543, 2006.

D. Bandyopadhyay and J. Snoeyink (2004). Almost Delaunay Simplices: Nearest Neighbor Relations for Imprecise Points. In *SODA 2004* (pp. 403-412).

H. Fei and J. Huan (2008). Structure feature selection for graph classification. In *CIKM* (pp. 991-1000).

H. Fei and J. Huan (2009). L2 Norm Regularized Feature Kernel Regression for Graph Data. In *CIKM* (pp. 593-600).

H. Fröhlich, J. K. Wegner, F. Sieker, A. Zell (2005). Optimal assignment kernels for attributed molecular graphs. In *ICML* (pp. 225-232).

T. Gärtner, P. A. Flach, S. Wrobel (2003). On graph kernels: hardness results and efficient alternatives. In *COLT* (pp. 129-143).

J. A. Gonzalez, L. B. Holder, D. J. Hook (2002). Graph-based relational concept learning. In *ICML* (pp. 219-226).

J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha (2004). Mining spatial motifs from protein structure graphs. In *RECOMB* (pp. 308-315).

N. Jin, C. Young and W. Wang (2009). Graph Classification Based on Pattern Co-occurrence. In *CIKM* (pp. 573-582).

N. Jin, C. Young and W. Wang (2010). GAIA: graph classification using evolutionary computation. In *SIGMOD* (pp. 879-890).

H. Kashima, T. Koyanagi (2002). SVM kernels for semi-structured data. In *ICML* (pp. 291-298).

H. Kashima, K. Tsuda, A. Inokuchi (2003). Marginalized kernels between labeled graphs. In *ICML* (pp. 321-328).

T. Kudo, E. maeda, Y Matsumoto (2004). An application of boosting to graph classification. In *NIPS*.

A. G. Murzin, S. E. Brenner, T. Hubbard, C. Chothia (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.* 247, pp. 536-540.

S. Ranu and A. K. Singh (2009). GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases. In *ICDE* (pp. 844-855).

H. Saigo, N. Kraemer and K. Tsuda (2008). Partial Least Squares Regression for Graph Mining. In *SIGKDD* (pp. 578-586).

A. Smalter, J. Huan, G. Lushington (2008). A Graph Pattern Diffusion Kernel for Chemical Compound Classification. In *BIBE'08*.

M. Thoma, H. Cheng, A. Gretton, J. Han, H. Kriegel, A. Smola, L. Song, P. Yu, X. Yan, K. Borgwardt (2009). Near-optimal supervised feature selection among frequent subgraphs, In *SDM*.

G. Wang and R. L. Dunbrack, Jr. (2003). PISCES: a protein sequence culling server. *Bioinformatics*, 19:1589-1591, 2003.

X. Yan, H. Cheng, J. Han, and P. S. Yu (2008). Mining significant graph patterns by leap search. In *SIGMOD* (pp. 433–444).

X. Yan and J. Han (2002). gSpan: graph-based substructure pattern mining. In *ICDM* (pp. 721–724).