Mining in the Proximity of Subgraphs

Nikhil S. Ketkar
University of Texas at Arlington
ketkar@cse.uta.edu

Lawrence B. Holder
University of Texas at Arlington
holder@cse.uta.edu

Diane J. Cook
University of Texas at Arlington
cook@cse.uta.edu

ABSTRACT

Graphs are a natural way to represent multi-relational data and are extensively used to model a variety of application domains in diverse fields ranging from bioinformatics to homeland security. Often, in such graphs, certain subgraphs are known to possess some distinct properties and graph patterns in the proximity of these subgraphs can be an indicator of these properties. In this work we focus on the task of mining in the proximity of subgraphs, known to possess certain distinct properties and identify patterns which distinguish these subgraphs from other subgraphs without these properties. This task is novel and of considerable interest as it can facilitate the prediction of previously unknown subgraphs possessing the properties under consideration in the graph and can lead to a better understanding of the application domain. We characterize the task of mining in the proximity of subgraphs as a supervised learning problem and present a heuristic algorithm for the same. Experimental comparison with the ILP system CProgol on real world and artificial datasets provides a strong indication of the ability and viability of the approach in uncovering interesting patterns.

1. INTRODUCTION

Recently, graph-based data mining or the use of a graph based representation for multi-relational data and the mining of interesting graph patterns has received a considerable amount of attention by the data mining community. As a result, a number of approaches have been developed which have achieved promising results in uncovering interesting patterns in biological networks [24], social networks[15] and the world wide web [21]. A concise overview of the current work in in the field can be achieved by categorizing it according to the types of tasks, the types of graph data and the approaches in graph-based data mining. There are two fundamentally different groups of tasks in graph-based data mining, namely those involving supervised learning and those involving unsupervised knowledge discovery. While the tasks involving supervised learning are concerned with inducing classifiers from graph data, the tasks involving unsupervised knowledge discovery are concerned with finding regularities in graph data. Graph data can be in two major forms, namely in the form of a single graph or a set of separate graph transactions which are also called examples. Approaches to graph-based data mining can be classified in six major categories, namely the mathematical graph theory based approaches, the greedy search based approaches, the inductive logic programming approaches, the probabilistic or statistical approaches, the inductive database approaches and the kernel function based approaches. We now discuss each of these approaches, their application to supervised learning and unsupervised knowledge discovery and the type of graph data they deal with.

Mathematical graph theory based approaches have been extensively applied to unsupervised knowledge discovery from graph data. These approaches mainly deal with graph data in the form of transactions and mine a complete set of graph patterns, mainly using a support or frequency measure. The initial work in this area was the AGM[6] system which uses the Apriori level-wise approach. FSG[12] takes a similar approach and further optimizes the algorithm for improved running times. gFSG [11] is a variant of FSG which enumerates all geometric subgraphs from the database. gSpan[25] uses DFS codes for canonical labeling and is much more memory and computationally efficient than the previous approaches. Instead of mining all subgraphs, CloseGraph[26] only mines closed subgraphs. A graph G is closed in a dataset if there exists no supergraph of G that has the same support as G. Gaston [18] efficiently mines graph datasets by first considering frequent paths which are transformed to trees which are further transformed to graphs. FFSM [5] is a graph mining system which uses an algebraic graph framework to address the underlying problem of subgraph isomorphism.

In comparison to mathematical graph theory based approaches which are complete, greedy search based approaches use heuristics to evaluate the solution. The two pioneering works in the field are Subdue[1] and GBI[13]. Subdue uses MDL-based compression heuristics, and has been applied to learning predictive as well as descriptive models. While learning descriptive models, Subdue can deal with both the single graph as well as the graph transactions category. While learning predictive models, Subdue mainly deals with the graph transactions category. Recently, some work has been pursued for learning predictive models in the single graph category[22]. In contrast to Subdue which uses

LinkKDD'06, August 20, 2006, Philadelphia, Pennsylvania, USA.

MDL-based compression heuristics, GBI uses an empirical graph size-based heuristic. The empirical graph size definition depends on the size of the extracted patterns and the size of the compressed graph. GBI learns predictive models in the graph transactions category.

Another methodology in this field is that of inductive logic programming. Although ILP cannot be considered to be a mainstream approach in graph-based data mining, when graph data is expressed as a collection of facts represented in Horn clause logic, inductive logic programming systems like FOIL[19], Progol[14] and WARMR[2] can be applied to supervised learning as well as unsupervised knowledge discovery from graph data. As the graph data is represented as a set of Horn clauses, there is little difference between the single graph category and the graph transactions category. Thus this approach can be applied to both of the categories. This approach has the advantage of the extensive descriptive power of Horn clause logic and ILP systems are capable of learning any multi-relational concept which can be expressed in Horn clause logic by providing the necessary background knowledge. This approach, due to its extensive descriptive power also tends to have a significantly lower performance as compared to graph-based systems while learning multi-relational concepts which do not require the extensive descriptive power, as shown by a recent empirical study[9].

Probabilistic or statistical approaches in graph-based data mining such as probabilistic relational models [3], relational probability trees [16], and relational Bayesian classifiers [17], focus on supervised learning from both the single graph as well as the graph transactions category of data. These approaches address tasks [4] such as entity resolution, group detection, link prediction by applying statistical inference procedures to graph data in light of difficulties such as autocorrelation [7], and degree disparity [8].

Another promising direction in the field of graph-based data mining is that of inductive databases which are a new generation of databases that are not only capable of dealing with data but also with patterns or regularities within the data. Data mining in such a framework is an interactive querying process. This approach mainly performs supervised learning from the graph transactions category of data. The inductive database framework is especially interesting for bioinformatics because of the large and complex databases that exist in this domain and the lack of methods to gain scientific knowledge from them. The pioneer work in this field was the MolFea system [20], which is based on the level-wise version space algorithm. MolFea is the Molecular Feature miner that mines for linear fragments in chemical compounds.

Lastly, the kernel function based approaches have been used to a certain extent for mining graph datasets. These approaches focus on supervised learning from the graph transactions category of data. The kernel function defines a similarity between two graphs. When high dimensional data is represented in linear space, the function to learn is difficult in this space. We can map the linear data to nonlinear, space and the problem of learning in that high dimensional space becomes learning scalar products. Kernel functions make computation of such scalar products very efficient. The key is finding efficient mapping functions and good feature vectors. The pioneering approach that applied kernel functions to graph structures is the diffusion kernel [10].

2. TASK DESCRIPTION

Having presented a brief overview of the current work in graph-based data mining, we introduce the task of mining in the proximity of subgraphs. Consider an example from a money laundering domain which comprises data about individuals, institutions and the transfer of funds among them. This data is multi-relational in nature and a graph-based representation provides a natural way to model this domain. Figure 1 (a) shows the graph-based representation of the fraud detection domain. Here, vertices represent individuals and institutions while edges represent the transfer of funds among individuals and institutions.

Now, in such a graph assume that we know certain individuals, institutions and the transactions between them to be fraudulent in nature. We also know certain individuals, institutions and the transactions between them to be innocent in nature. These individuals and institutions along with the transfer of funds among them can be viewed as subgraphs in the graph representation of the multi-relational data from the money laundering domain. What is peculiar about these subgraphs is that they can be viewed to either possess or lack the property of being fraudulent in nature. We refer to these subgraphs as sites. Furthermore, the subgraphs known to possess some distinct properties are referred to as positive sites and the subgraphs known to lack some distinct properties are referred to as negative sites. Figure 1 (b) shows two subgraphs representing known fraudulent individuals, institutions and the transfer of funds between them (positive sites) on the graph. Two other subgraphs representing known innocent individuals, institutions and the transfer of funds between them (negative sites) are also denoted on the graph. Three important points must be noted. Firstly, the site, as a whole, is known to possess or lack a distinct property. The individual vertices and edges which form the site do not possess the property under consideration. Secondly, all the sites, positive or negative, may or may not be isomorphic to each other. Lastly, sites may consist of a connected or disconnected subgraphs, although our example only illustrates sites which consist of connected subgraphs for the sake of simplicity.

Now, given this graph containing positive and negative sites, we would like to identify the characteristics that distinguish the positive sites from the negative sites. This would serve two purposes. Firstly, the characteristics that distinguish the positive sites from the negative sites could help identify other, previously unknown positive sites in the graph. In our example this would be previously unknown, possibly fraudulent, individuals, institutions and the transactions among them. Secondly, the characteristics would help further our understanding of the domain. In our example this would be insights into the modus oprandi of individuals and institutions engaged in money laundering activities. It is intuitive that the characteristics which distinguish positive sites from negative sites might be certain graph patterns in the proximity of sites. In our example it is likely as the transfer of funds among fraudulent individuals and institutions have a pattern different from the transfer of funds between innocent individuals and institutions. Figure 1 (c) shows a graph pattern which distinguishes positive sites from negative sites in the graph. Figure 1 (d) shows a previously unknown positive site, that is, possibly fraudulent individuals and institutions and the transactions among them, identified using the pattern shown in Figure 1 (c).

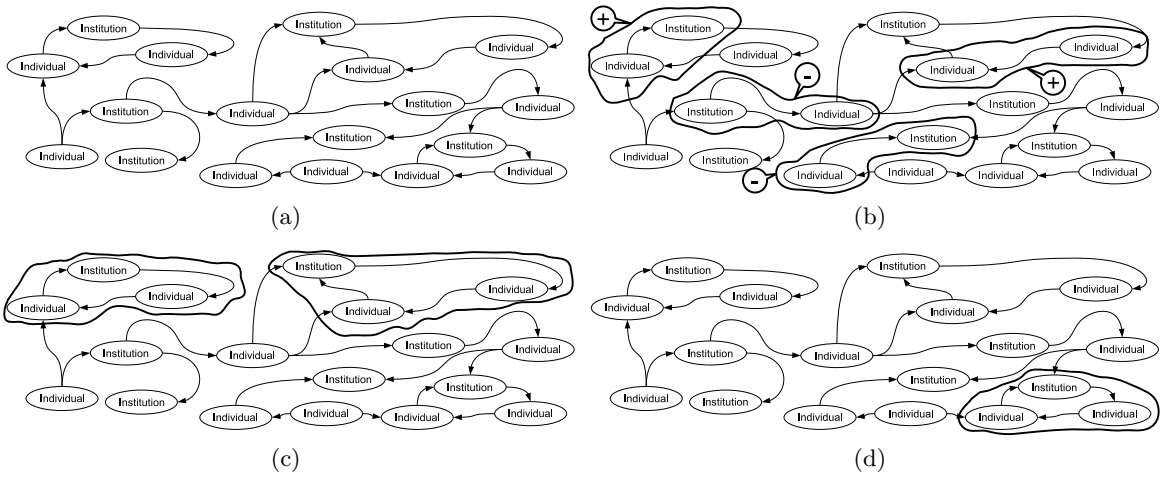


Figure 1: An example from the money laundering domain illustrating the task of mining in the proximity of subgraphs (a)Graph-based representation of the money laundering domain. Vertices represent individuals and institutions, edges represent the transfer of funds among individuals and institutions. (b)Positive and negative sites in the graph. Subgraphs representing known fraudulent individuals, institutions and the transfer of funds among them are positive sites. Subgraphs representing known innocent individuals, xs institutions and the transfer of funds among them are negative sites. (c)Distinguishing graph pattern in the proximity of the positive sites absent in the proximity of the negative sites. (d)Previously unknown, possibly fraudulent individuals, institutions and transfer of funds between them, identified by using the discovered pattern.

The example discussed above presents a quick overview of the task of mining in the proximity of subgraphs. Our survey indicates that this task could have potential applications in a variety of multi-relational domains. An example would be the application of the task in identifying previously unknown terrorist cells or threat groups using a communication graph and known terrorist cells. Another example would be the application of the task in identifying previously unknown functional modules from gene interaction graphs and known functional modules. Among the current approaches for mining graph-based data, only the inductive logic programming approach can be applied to this task to a certain extent. ILP systems can learn predicate definitions analogous to a site in the case where all sites are isomorphic to each other. Our experiments show that our approach tends to perform better than ILP while learning large concepts and while learning from large graphs. In the case where the sites consist of non-isomorphic subgraphs with varying number of vertices and edges, ILP performs poorly as this involves learning a predicate definition containing a list which can have a variable length, and is computationally expensive.

In this work, we characterize this task as a supervised learning problem and present a heuristic algorithm for the same. Experimental comparison with the ILP system CProgol on real world and artificial datasets evaluate the ability of the approach in uncovering interesting patterns. The rest of the paper is organized as follows. In Section 3, we formalize the task of mining in the proximity of subgraphs. In Section 4, we present a heuristic algorithm for the task and in Section 5, we present experimental results on a number of real world and artificially generated datasets. Conclusions and future work are presented in Section 6.

3. TASK FORMULATION

In this section we formalize the task of mining in the proximity of subgraphs.

Our graph-based representation of multi-relational data consists of a single graph where entities are represented as vertices and the relationships between entities are represented as edges. Note that this graph may be connected or disconnected. Most application domains have different types of entities and relations, hence in our graph-based representation, the vertices and edges have labels associated with them. Relations between entities can be directional or non-directional in nature. For example, the transfer of funds from one person to other is directional in nature while a person living in the same city block as some other person would be non-directional in nature. Hence, in our graph-based representation, edges can be directed or undirected. Note that it is possible to model non-directional relations using two directed edges, but we avoid this representation for the sake of simplicity. Lastly, our graph-based representation does not allow self loops and multiple edges. Such a graph can be formally defined as $G = (V, D, U, \Sigma_V, \Sigma_D, \Sigma_U, l_V, l_D, l_U)$ where,

1. V denotes the finite set of vertices,
2. D denotes the set of directed edges such that $D \subseteq V \times V$,
3. U denotes the set of undirected edges such that $U \subseteq V \times V$,
4. Σ_V denotes the finite alphabet of vertex labels,
5. Σ_D denotes the finite alphabet of directed edge labels,
6. Σ_U denotes the finite alphabet of undirected edge labels,

7. l_V denotes the finite total function $l_V : V \rightarrow \Sigma_V$,

8. l_D denotes the finite total function $l_D : D \rightarrow \Sigma_D$, and

9. l_U denotes the finite total function $l_U : U \rightarrow \Sigma_U$.

We now define subgraphs for our graph-based representation. Let $G = (V, D, U, \Sigma_V, \Sigma_D, \Sigma_U, l_V, l_D, l_U)$ and $G' = (V', D', U', \Sigma'_V, \Sigma'_D, \Sigma'_U, l'_V, l'_D, l'_U)$ be two graphs. G' is defined to be a subgraph of G iff the following hold,

1. $V' \subseteq V$,

2. $D' \subseteq D$ and

3. $U' \subseteq U$.

We now define graph isomorphism for our graph-based representation. Let $G = (V, D, U, \Sigma_V, \Sigma_D, \Sigma_U, l_V, l_D, l_U)$ and $G' = (V', D', U', \Sigma'_V, \Sigma'_D, \Sigma'_U, l'_V, l'_D, l'_U)$ be two graphs. G' is defined to be isomorphic to G iff there exists a bijection $m : V' \rightarrow V$ such that the following hold,

1. $\forall p \in V', (l'_V(p) = l_V(m(p)))$,

2. $\forall p, q \in V', ((p, q) \in D' \Leftrightarrow ((m(p), m(q)) \in D)$,

3. $\forall u \in D', (l'_D(u) = l_D(m(u)))$,

4. $\forall r, s \in V', ((r, s) \in U' \Leftrightarrow ((m(r), m(s)) \in U)$, and

5. $\forall v \in U', (l'_U(v) = l_U(m(v)))$.

Using these notions we can now define the task of mining in the proximity of subgraphs. Given a graph G and a set of subgraphs S_P, which we refer to as positive sites, and another set of subgraphs S_N, which we refer to as negative sites, find a subgraph c of graph G such that the following holds,

1. $\forall p \in S_P, \exists i$ such that the following hold,

   (a) p is a subgraph of i

   (b) i is a subgraph of G

   (c) i is isomorphic to c

2. $\forall n \in S_N, \nexists j$ such that the following hold,

   (a) n is a subgraph of j

   (b) j is a subgraph of G

   (c) j is isomorphic to c

We refer to the subgraph c as a concept. Intuitively, this task can be viewed as identifying a subgraph such that a subgraph isomorphic to this subgraph is present and is connected to every positive site and there does not exist a subgraph, isomorphic to this subgraph and connected to any of the negative sites.

4. ALGORITHM AND ANALYSIS

In this section we present MPS (Mining in the Proximity of Subgraphs), a computationally constrained, heuristic algorithm for the task of learning mining in the proximity of subgraphs. MPS accepts a graph and a set of positive and negative sites denoted on this graph as input. MPS generates a list of subgraphs to approximate the concept subgraph, as defined in Section 2. Intuitively, this can be seen as generating a set of subgraphs such that subgraphs isomorphic to these subgraphs are mostly present and connected to positive sites and subgraphs isomorphic to these subgraphs are rarely connected to any of the negative sites. MPS is presented in Algorithm 1. An illustration of the working of MPS, using a example from the money laundering domain is given in Figure 2.

MPS performs a beam search which begins by generating a list of the subgraph instances which form the positive and negative sites. These subgraph instances are extended by one vertex and one edge or one edge in all possible ways, as guided by the input graph, to generate the extended subgraph instance list. Subgraph instances in this list are then grouped to form a list of isomorphic subgraph instance groups. Every isomorphic subgraph instance group corresponds to a subgraph and this subgraph is then evaluated according to the following measure.

$$V(g) = \frac{|P_g| + |N_g|}{|S_P| + |S_N|}$$

where,

1. g is the subgraph,

2. $P_g = \{r \mid g$ is a subgraph of r and $\exists g \in S_P\}$,

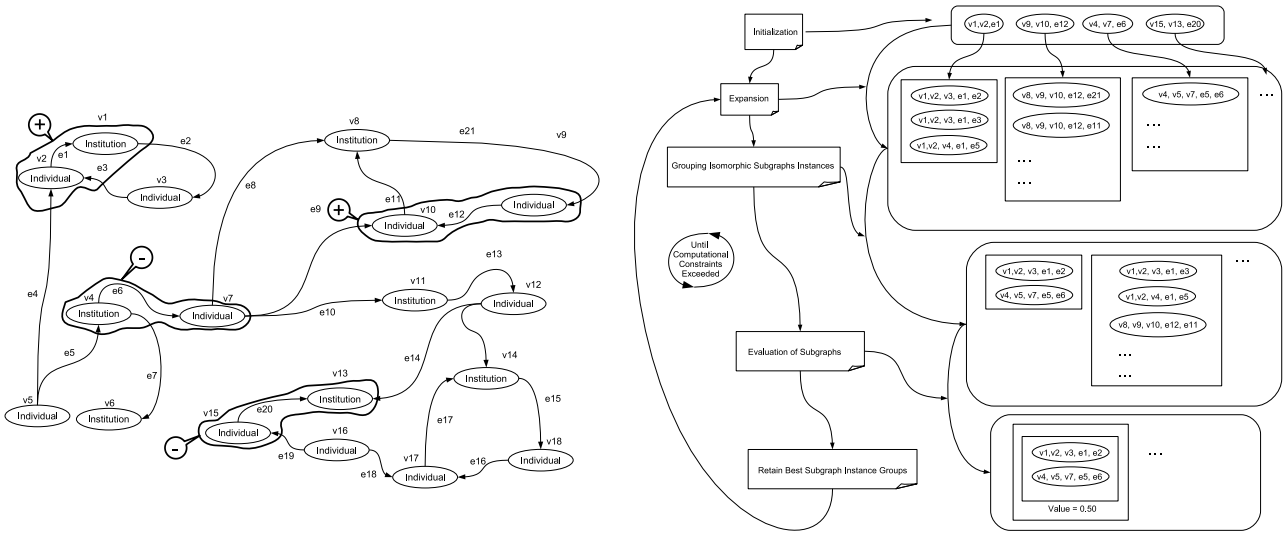
3. $N_g = \{s \mid g$ is a subgraph of s and $\nexists g \in S_N\}$,

Intuitively, this evaluation measure can be viewed as the number of positive sites to which an instance of the subgraph is connected added to the number of negative sites which do not have an instance of the subgraph connected to it, divided by the total number of positive and negative sites. After evaluating each subgraph corresponding to an isomorphic instance group, only the best groups are retained for further expansion. The parameter Beam, determines how many groups are retained for further expansion. The process of expansion, evaluation and retaining the best instance groups continues until the constraint enforced by the Limit parameter is exceeded. After this the subgraph corresponding to the best instance group is added to the list of best subgraphs and the positive sites which have an instance of the subgraph connected to it are removed from the list of positive sites. The procedure of expansion, evaluation and retaining the best instance groups then continues on the updated positive sites. This process continues until the constraint enforced by the Iterations parameter is exceeded or there are no more positive sites. Finally, the algorithm reports the list of best subgraphs.

The MPS algorithm is constrained to be polynomial by the Limit and the Beam parameters in the case where the graph isomorphism check is also polynomial. At present, graph isomorphism has not been proven to be either an NP-complete problem nor a P-problem although it is known to be polynomial in most cases except for some instances of unconstrained graphs[23].

5. EXPERIMENTAL RESULTS

We analyzed the performance of the MPS algorithm by comparing it with the ILP system CProgol on a number of real and artificially generated datasets. In this section we briefly discuss CProgol, our experiments and their results. The MPS algorithm was implemented in CMU Common Lisp 19c and our experiments were performed on 2.80 GHZ Intel Xeon server with 2GB main memory running Suse 9.1.



(a) Identifiers assigned to vertices and edges.

(b) Important steps in MPS algorithm.

Figure 2: Illustration of the MPS algorithm using an example from the money laundering domain. MPS begins with the initialization of the list of subgraphs from the sites. This is followed by the expansion of subgraphs, the grouping of isomorphic subgraphs, the evaluation of isomorphic subgraph groups and retention of the best subgraph groups. This process continues until the computational constraints are exceeded.

Algorithm 1 MPS

Input: $G, S_P, S_N, Iterations, Beam, Limit$
Output: L_c

$IterationCount \leftarrow 0$;
repeat
  $LimitCount \leftarrow 0$;
  $BestSubgraph \leftarrow \phi$;
  $BestSubgraphValue \leftarrow 0$
  $CurrInstanceList \leftarrow S_P \cup S_N$;
  repeat
    $ExtendedInstanceList \leftarrow$ EXTEND every instance in $CurrInstanceList$ in possible ways;
    repeat
      $CurrInstance \leftarrow$ FIRST element of $ExtendedInstanceList$;
      $CurrInstanceGroup \leftarrow$ REMOVE all subgraph instances in $ExtendedInstanceList$ isomorphic to $CurrInstance$;
      $CurrSubgraph \leftarrow$ EXTRACT subgraph from $CurrInstance$;
      $CurrSubgraphValue \leftarrow$ EVALUATE the value of $CurrSubgraph$;
      if $CurrSubgraphValue > BestSubgraphValue$ **then**
        $BestSubgraph \leftarrow CurrSubgraph$;
        $BestSubgraphValue \leftarrow CurrSubgraphValue$;
      end if
      INSERT $CurrInstanceGroup$ in $InstanceGroupList$, order by $CurrSubgraphValue$;
      INCREMENT $LimitCount$;
    until $(ExtendedInstanceList \neq \phi)$ and $(LimitCount \leq Limit)$
    $CurrInstanceList \leftarrow$ APPEND first $Beam$ instance groups in $InstanceGroupList$;
  until $(CurrinstanceList \neq \phi)$ and $(LimitCount \leq Limit)$
  INSERT $BestSubgraph$ in L_c;
  $S_P \leftarrow S_P -$ COVERED$(BestSubgraph)$;
  INCREMENT $IterationCount$;
until $(S_P \neq \phi)$ and $(IterationCount \leq Iterations)$
return L_c

5.1 CProgol

CProgol[14] is an ILP system, characterized by the use of mode-directed inverse entailment and a hybrid search mechanism. Inverse entailment is a procedure which generates a single, most specific clause that, together with the background knowledge, entails the observed data. The inverse entailment in CProgol is mode-directed, that is, it uses mode definitions. A mode declaration is a constraint which imposes restrictions on the atoms and their arguments appearing in a hypothesis clause by,

1. Determining which atoms can occur in the head and the body of hypotheses.
2. Determining which arguments can be input variables, output variables or constants.
3. Determining the number of alternative solutions for instantiating the atom.

The user-defined mode declarations aid the generation of the most specic clause. CProgol first computes the most specific clause which covers the seed example and belongs to the hypothesis language. The most specific clause can be used to bound the search from below. The search is now bounded between the empty clause and the most specific clause. The search proceeds within the bounded search space in a general-to-specific manner. The search is a hybrid search, because it is a general-to-specific search bounded from below with respect to the most specific clause. The search strategy is an A* algorithm which is guided by a weighted compression and accuracy measure. The A* search returns a clause which covers the most positive examples and maximally compresses the data. Any arbitrary Prolog program can serve as background knowledge for CProgol. The mode denitions and the background knowledge together define a hypothesis language. The hypothesis space explored by CProgol consists of every hypothesis defined by the hypothesis language.

5.2 Real Datasets

Our experimentation with real datasets involved seven tasks on two datasets extracted from the Internet Movie Database (www.imdb.com). The first dataset consists of data about movie sequels. A sequel is a movie that follows another movie and contains elements of the previous movie like same characters and settings. For example the movie Shrek 2 is a sequel of the movie Shrek. We identified 107 movies and their sequels and then extracted the actors, directors and writers of each of these movies form the IMDb. This data was then represented as a graph, where movies, actors, directors and writers were represented as vertices and a person acting, directing or writing a particular movie was represented by an edge between a person and the movie. Figure 3 illustrates this representation of movie data using a graph.

The second dataset consists of data about movie trilogies. A trilogy is a set of three movies involving some of the same characters and a continued story line. An example would be the movies Matrix, Matrix Reloaded and Matrix Revolutions which form a trilogy. We identified 53 movie trilogies and extracted the actors directors and writers of each of the movies from the IMDb. This data was represented as a graph in a manner similar to the movie sequel dataset, as shown in Figure 3.

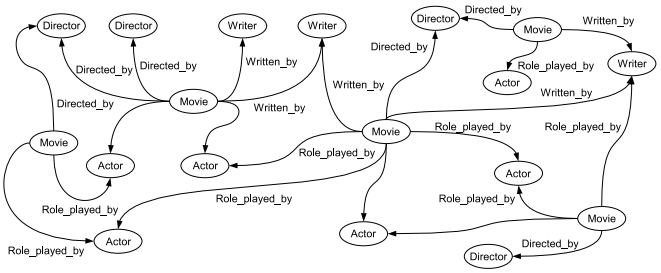


Figure 3: Graph based representation of the IMDB domain.

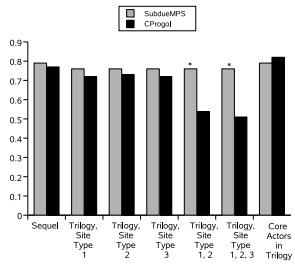
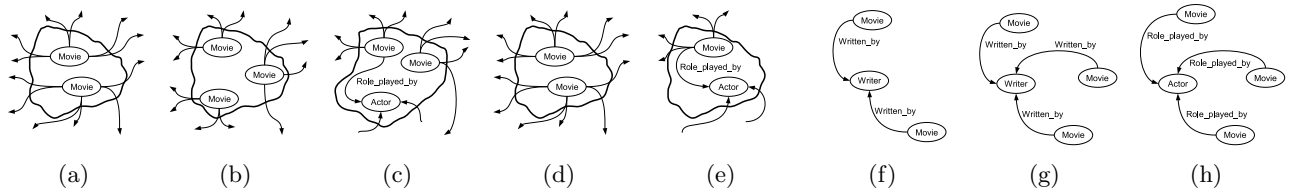


Figure 4: Results on tasks from IMDB domain. An asterisk indicates significantly different performance.

Our first task involved learning the concept of a sequel from the movie sequel dataset given two movies which comprise a sequel as positive sites and two movies which do not form a sequel as negative sites. This is illustrated in Figure 5 (a). Our second task involved learning the concept of a trilogy from the movie trilogy dataset given three movies which comprise a trilogy as positive sites and three movies which do not form a trilogy as negative sites. This is illustrated in Figure 5 (b). The third task involved learning the concept of a trilogy from the movie trilogy dataset given two movies and an actor in a trilogy as positive sites and two movies and an actor which do not form a trilogy as negative examples. This is illustrated in Figure 5 (c). The next task involved learning the concept of trilogy given only two movies in the trilogy as positive sites and two movies which do not fall in a trilogy as negative examples. This is illustrated in Figure 5 (d). The fifth task involved learning the concept of trilogy given two different kinds of sites. The first site consisted of three movies as in Figure 5 (b) and the second site consisted of two movies and a actor as in Figure 5 (c). The sixth task involved learning the concept of trilogy given three different kinds of sites. The first site consisted of three movies as in Figure 5 (b), the second site consisted of two movies and a actor as in Figure 5 (c) and the third site consisted of two movies as in Figure 5 (d). The final task involved learning core characters in a trilogy given an actor and a movie. This is illustrated in Figure 5 (e). Core characters are characters which have an important part in the story line of the trilogy. For example, the characters of Neo, Trinity and Morpheus are core characters in the Matrix trilogy. It must be noted that in all the tasks mentioned above, the negative sites can be two movies, three movies or an actor and two movies which do not form a sequel or



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 5: Sites and concepts learned on the tasks from IMDB domain. (a)Site for learning Sequel. (b)Type 1, two movies. (c)Type 2, Two movies and an actor. (e)Type 3, Three movies. (d)Site for learing core actors in a trilogy. (e)Concept learnt for Sequel. (f)Concept learnt for trilogy. (g)Concept learnt for core actors in a trilogy.

a trilogy, depending on the task. Due to this there can be a very large number of negative sites (all combinations of actors and movies). For each task we randomly selected a subset of all these negative sites, keeping the number of negative sites selected equal to the number of positive sites for the task.

We evaluated the MPS on each of these tasks by performing a 3 fold cross validation on each of the task mentioned above. As a baseline, we ran CProgol on each of the tasks. For tasks 1,2,3,4 and 7 CProgol was provided with mode definitions to learn predicates $sequel(movie, movie)$, $trilogy(movie, movie, movie)$, $trilogy(actor, movie, movie)$, $trilogy(movie, movie)$ and $core(actor, movie)$ respectively. For tasks 5 and 6, since the sites are not isomorphic it is not possible to formulate the problem in this way. This is because a site representing a trilogy can either be three movies or an actor and two movies in task 5 and three movies or an actor and two movies or just two movies in task 6. We formulate this problem as learning the predicate $trilogy(A)$ where A is a list of entities where entities can be actors or movies and the number of entities can vary in length. The appropriate mode definitions to formulate the task in this manner were provided to CProgol for tasks 5 and 6.

The results are presented in Figure 4. The performance of MPS was found to be comparable to CProgol for tasks 1,2,3,4 and 7. For tasks 5 and 6 MPS performed significantly better than CProgol. The concepts learned by MPS are presented in Figure 5 (f to h).

It can be seen that in each task the concept learned by MPS not only achieves high accuracy but also uncovers an interesting property about the domain. CProgol has a lower performance in tasks 5 and 6 where it has to learn a predicate containing a list which can be computationally more expensive than learning a predicate containing a fixed number of atoms. These results indicate that the performance of MPS is comparable to CProgol while learning from sites isomorphic to each other. However in the general case where the sites can be subgraphs of arbitary size, MPS can outperform CProgol.

5.3 Artificial Datasets

We systematically analyzed the MPS algorithm by comparing its performance with CProgol on a number of artificially generated datasets. We identified three factors which would have a major effect on the performance of the algorithm, namely, the size of the graph, the number of sites and the size of the site. For simplicity, we measure graph size and site size as the number of edges. Our artificial graph generator accepts three parameters namely the size of the graph, the size of the site and the number of sites and generates a graph with the given number of sites, half of which are positive and half negative. The number of vertices, the vertex labels and the edge labels are chosen by the generator so as to satisfy user defined parameters. The artificial graph generator also generates an equivalent logic representation of the graph dataset consisting of predicates such as $vertex(id, label)$, $edge(id, id, label)$ and $site(id, id)$ which we use for our experimentation with CProgol. Note that for these experiments all the generated sites were isomorphic and the learning problem could be formulated as learning a single predicate. Also note that we use half of the sites for training and half for testing and every result is an average over three runs. In each of our experiments we generate datasets by holding two parameters constant and vary the third parameter. The performance of MPS and CProgol is then measured on the datasets. Since MPS is implemented in Common Lisp and CProgol in C, we measure the number of hypotheses explored by each algorithm instead of the runtime. The number of hypotheses explored by each system can be easily found by counting the number of times the evaluation function is called in each algorithm.

Figures 6 (a) and 6 (b) show the accuracy and the number of hypotheses explored by MPS and CProgol with increasing size of graph with the number of sites and the size of the site kept constant. Figures 6 (c) and 6 (d) show the accuracy and the number of hypotheses explored by MPS and CProgol with increasing number of sites with the graph size and site size kept constant. Figures 5 (e) and 5 (f) show the accuracy and the number of hypotheses explored by MPS and CProgol with increasing size of site with the graph size and number of sites kept constant. We observe that MPS achieves higher accuracy than CProgol as the size of the concept (number of edges or relations in the concept) grows. CProgol has to explore an increasingly larger number of hypotheses than Subdue as the size of the concept grows. MPS achieves an increased accuracy which is comparable to CProgol as the number of sites are increased. CProgol has to explore a larger number of hypotheses than MPS as the number of sites are increased. MPS achieves an increased accuracy which is comparable to CProgol as the size of the sites (number of edges or relations in the site) increases. CProgol has to explore an increasingly larger number of hypotheses than MPS as the as the size of the sites (number of edges or relations in the site) increases. The results indicate that MPS can outperform CProgol while learning large concepts (measured as the number of relations) and while learning from large relational datasets (measured as the number of relations).

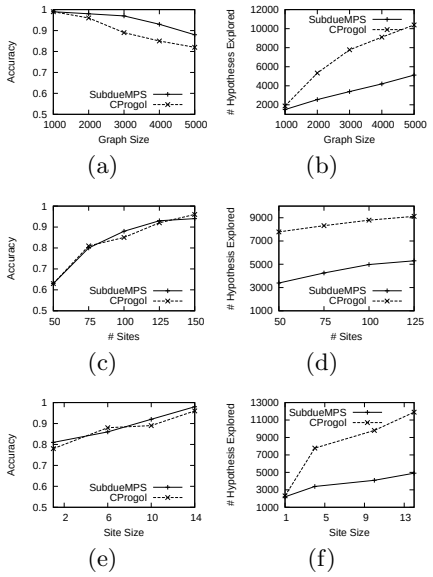


Figure 6: Results on the artificial datasets.

6. CONCLUSIONS AND FUTURE WORK

We identified the novel task of mining in the proximity of subgraphs, known to possess certain distinct properties and identify patterns which distinguish these subgraphs from other subgraphs without these properties. We presented MPS, a heuristic algorithm to address the task. Experimental comparison with the ILP system CProgol on real world and artificial datasets indicate the ability of the approach in uncovering interesting patterns.

MPS is currently limited to discovering exact graph patterns in the proximity of subgraphs. Discovering inexact graph patterns which are approximately present in the proximity of given subgraphs would allow MPS to function more robustly in most real world domains. As a part of our future work we plan on extending the algorithm for discovering inexact graph patterns.

7. REFERENCES

[1] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *J. Artif. Intell. Res. (JAIR)*, 1:231–255, 1994.

[2] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Min. Knowl. Discov.*, 3(1):7–36, 1999.

[3] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

[4] L. Getoor. Link mining: a new data mining challenge. *SIGKDD Explorations*, 5(1):84–89, 2003.

[5] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*, pages 549–552, 2003.

[6] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23, 2000.

[7] D. Jensen and J. Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In *ICML*, pages 259–266, 2002.

[8] D. Jensen, J. Neville, and M. Hay. Avoiding bias when aggregating relational data with degree disparity. In *ICML*, pages 274–281, 2003.

[9] N. S. Ketkar, L. B. Holder, and D. J. Cook. Comparison of graph-based and logic-based multirelational data mining. *SIGKDD Explorations*, 7(2), 2005.

[10] R. I. Kondor and J. D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *ICML*, pages 315–322, 2002.

[11] M. Kuramochi and G. Karypis. Discovering frequent geometric subgraphs. In *ICDM*, pages 258–265, 2002.

[12] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1038–1051, 2004.

[13] T. Matsuda, T. Horiuchi, H. Motoda, and T. Washio. Extension of graph-based induction for general graph structured data. In *PAKDD*, pages 420–431, 2000.

[14] S. Muggleton. Inverse entailment and progol. *New Generation Comput.*, 13(3&4):245–286, 1995.

[15] M. Mukherjee and L. B. Holder. Graph-based data mining for social network analysis. In *Proceedings of the ACM KDD Workshop on Link Analysis and Group Detection*, 2004.

[16] J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. In *KDD*, pages 625–630, 2003.

[17] J. Neville, D. Jensen, and B. Gallagher. Simple estimators for relational bayesian classifiers. In *ICDM*, pages 609–612, 2003.

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

[19] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[20] L. D. Raedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In *IJCAI*, pages 853–862, 2001.

[21] A. Rakhshan, L. B. Holder, and D. J. Cook. Structural web search engine. In *FLAIRS Conference*, pages 319–324, 2003.

[22] I. Russell and Z. Markov, editors. *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference, Clearwater Beach, Florida, USA*. AAAI Press, 2005.

[23] S. Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Addison-Wesley, 1990.

[24] S. Su, D. J. Cook, and L. B. Holder. Application of knowledge discovery to molecular biology: Identifying structural regularities in proteins. In *Pacific Symposium on Biocomputing*, pages 190–201, 1999.

[25] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

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