Graph-based Relational Learning with Application to Security

Lawrence Holder, Diane Cook, Jeff Coble and Maitrayee Mukherjee
Department of Computer Science and Engineering
University of Texas at Arlington
Box 19015, Arlington, TX 76019

Abstract. We describe an approach to learning patterns in relational data represented as a graph. The approach, implemented in the Subdue system, searches for patterns that maximally compress the input graph. Subdue can be used for supervised learning, as well as unsupervised pattern discovery and clustering. We apply Subdue in domains related to homeland security and social network analysis.

1 Introduction

The £eld of relational data mining, of which graph-based relational learning is a part, is a new area investigating approaches to mining relational information by £nding associations involving multiple tables in a relational database. Two main approaches have been developed for mining relational information: logic-based approaches and graph-based approaches. Logic-based approaches fall under the area of inductive logic programming (ILP) [17]. ILP embodies a number of techniques for inducing a logical theory to describe the data, and many techniques have been adapted to relational data mining [7]. Graph-based approaches differ from logic-based approaches to relational mining in several ways, the most obvious of which is the syntax of the representation. Furthermore, logic-based approaches to supervised relational learning rely on the prior identi£cation of the predicate or predicates to be learned, while graph-based approaches are more data-driven, identifying any portion of the graph that has discriminatory power. However, logic-based approaches allow the expression of more complicated patterns involving, e.g., recursion, variables, and constraints among variables. These representational limitations of graphs can be overcome, but at a computational cost.

The ability to mine relational data has become a crucial challenge in many security-related domains. For example, the U.S. House and Senate Intelligence Committees' report on their inquiry into the activities of the intelligence community before and after the September 11, 2001 terrorist attacks revealed the necessity for "connecting the dots" [21], that is, focusing on the relationships between entities in the data, rather than merely on an entity's attributes. A natural representation for this information is a graph, and the ability to discover previously-unknown patterns in such information could lead to signi£cant improvement in our ability to identify potential threats.

In this article we review techniques for relational learning and focus on a method for graph-based relational learning implemented in the Subdue system. We then look at the application of Subdue to the security-related problem of identifying patterns in a graph representation of relational data describing a simulated terrorist threat scenario.

2 Related Work

In this section we review representative approaches to both logic-based and graph-based relational learning. Logic-based relational learning methods come under the class of methods known as Inductive Logic Programming (ILP). Graph-based relational learning methods are less proli£c. We £rst review related work in graph-based data mining, discuss the difference between graph-based data mining and graph-based relational learning, and then describe two approaches to graph-based relational learning.

2.1 Logic-based Relational Learning

One approach to relational learning is Inductive Logic Programming (ILP), which represents data using First Order Predicate Calculus (FOPC) in the form of Prolog logic programs. ILP systems have been successful in a number of domains (e.g., biological domains [19]). We describe here two prominent ILP systems: FOIL and PROGOL. We also describe the ILP system WARMR, whose approach is similar to that of some graph-based data mining methods.

The FOIL system [2] is a top-down approach for learning relational concepts (theories) represented as an ordered sequence of function-free de£nite clauses. Given extensional background knowledge relations and examples of the target concept relation, FOIL begins with the most general theory and follows a set-covering approach, which repeatedly adds a clause to the theory that covers some of the positive examples and few negative examples. Individual clauses are generated by adding literals one at a time. The next literal added is that which maximizes an information gain heuristic based on examples correctly classi£ed before and after addition of the literal. Literals within some percentage of the maximum can be retained for possible later backtracking. FOIL exerts some preference toward literals that introduce new variables, but are determinate. In order to avoid overly-complex clauses, FOIL uses a result from the minimum description length principle that the description length of the clause should not exceed the description length of the examples covered by the clause. FOIL has been applied to a number of relational domains, including learning patterns in hypertext [22].

The Progol system [18] uses inverse entailment guided by mode declarations to learn relational concepts represented by de£nite clauses. Progol is an incremental learner that processes examples one at a time. Clauses learned from the current example are added to the background knowledge, and the covered positive examples are removed from consideration (i.e., a set-covering approach). Inverse entailment identi£es the most-speci£c clause consistent with the background knowledge, the current example and the mode declarations. Progol's mode declarations constrain how variables are bound within literals, as in FOIL, but also limits the number of instantiations of these literals within a theory. Progol performs an A*-like search of a lattice ordered by subsumption and bounded by the most-speci£c clause. The search is guided by maximizing theory compression and by a re£nement operator that keeps the search within the lattice and avoids redundancy. Progol has also been successful in several relational domains, e.g., mutagenesis [23] and protein structure [24].

The WARMR system [5] uses an Apriori-like method to £nd all substructures that occur more than a given frequency threshold, or minimum support, in the input examples. WARMR represents examples and learned substructures using de£nite clause logic and searches the space of substructures using specialization ordered by Θ-subsumption and constrained by

mode declarations. As with other Apriori-based approaches, WARMR gains ef£ciency by pruning all specializations of substructures whose frequency falls below the given threshold.

2.2 Graph-based Relational Learning

Graph-based data mining (GDM) is the task of £nding novel, useful, and understandable graph-theoretic patterns in a graph representation of data. Several approaches to GDM exist based on the task of identifying frequently occurring subgraphs in graph transactions, i.e., those subgraphs meeting a minimum level of support. Kuramochi and Karypis [16] developed the FSG system for £nding all frequent subgraphs in large graph databases. FSG starts by £nding all frequent single and double edge subgraphs. Then, in each iteration, it generates candidate subgraphs by expanding the subgraphs found in the previous iteration by one edge. In each iteration the algorithm checks how many times the candidate subgraph occurs within an entire graph. The candidates, whose frequency is below a user-de£ned level, are pruned. The algorithm returns all subgraphs occurring more frequently than the given level.

Yan and Han [26] introduced gSpan, which combines depth-£rst search and lexicographic ordering to £nd frequent subgraphs. Their algorithm starts from all frequent one-edge graphs. The labels on these edges together with labels on incident vertices de£ne a code for every such graph. Expansion of these one-edge graphs maps them to longer codes. The codes are stored in a tree structure such that if $\alpha = (a_0, a_1, ..., a_m)$ and $\beta = (a_0, a_1, ..., a_m, b)$, the β code is a child of the α code. Since every graph can map to many codes, the codes in the tree structure are not unique. If there are two codes in the code tree that map to the same graph and one is smaller than the other, the branch with the smaller code is pruned during the depth-£rst search traversal of the code tree. Only the minimum code uniquely de£nes the graph. Code ordering and pruning reduces the cost of matching frequent subgraphs in gSpan.

Inokuchi et al. [12] developed the Apriori-based Graph Mining (AGM) system, which uses an approach similar to Agrawal and Srikant's [1] apriori algorithm for discovering frequent itemsets. AGM searches the space of frequent subgraphs in a bottom-up fashion, beginning with a single vertex, and then continually expanding by a single vertex and one or more edges. AGM also employs a canonical coding of graphs in order to support fast subgraph matching. AGM returns association rules satisfying user-speci£ed levels of support and con£dence.

We distinguish graph-based relational learning (GBRL) from graph-based data mining in that GBRL focuses on identifying novel, but not necessarily most frequent, patterns in a graph representation of data [11]. Only a few GBRL approaches have been developed to date. Two speci£c approaches, Subdue [4] and GBI [27], take a greedy approach to £nding subgraphs maximizing an information theoretic measure. Subdue searches the space of subgraphs by extending candidate subgraphs by one edge. Each candidate is evaluated using a minimum description length metric [20], which measures how well the subgraph compresses the input graph if each instance of the subgraph were replaced by a single vertex. GBI continually compresses the input graph by identifying frequent triples of vertices, some of which may represent previously-compressed portions of the input graph. Candidate triples are evaluated using a measure similar to information gain. Kernel-based methods have also been used for supervised GBRL [15].

3 Graph-based Relational Learning in Subdue

The Subdue graph-based relational learning system[1] [4, 3] encompasses several approaches to graph-based learning, including discovery, clustering and supervised learning, which will be described in this section. Subdue uses a labeled graph $G = (V, E, L)$ as both input and output, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of vertices, $E = \{(v_i, v_j)|v_i, v_j \in V\}$ is a set of edges, and L is a set of labels that can appear on vertices and edges. The graph G can contain directed edges, undirected edges, self-edges (i.e., $(v_i, v_i) \in E$), and multi-edges (i.e., more than one edge between vertices v_i and v_j). The input graph need not be connected, but the learned patterns must be connected subgraphs (called substructures) of the input graph. The input to Subdue can consist of one large graph or several individual graph transactions, and in the case of supervised learning, the individual graphs are classi£ed as positive or negative examples.

3.1 Substructure Discovery

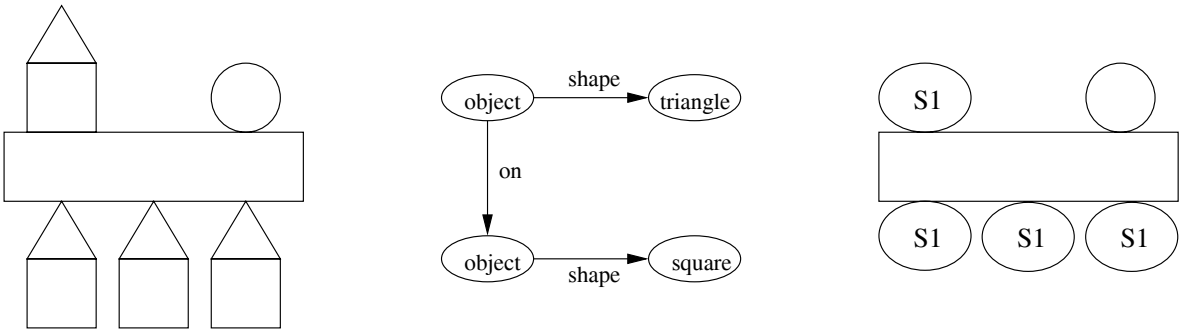
Subdue searches for a substructure that best compresses the input graph. Subdue uses a variant of beam search for its main search algorithm. A substructure in Subdue consists of a subgraph de£nition and all its occurrences throughout the graph. The initial state of the search is the set of substructures consisting of all uniquely labeled vertices. The only operator of the search is the *ExtendSubstructure* operator. As its name suggests, it extends a substructure in all possible ways by a single edge and a vertex, or by only a single edge if both vertices are already in the subgraph.

The search progresses by applying the *ExtendSubstructure* operator to each substructure in the current state. The resulting state, however, does not contain all the substructures generated by the *ExtendSubstructure* operator. The substructures are kept on a queue and are ordered based on their description length (or sometimes referred to as value) as calculated using the MDL principle described below.

The search terminates upon reaching a user-speci£ed limit on the number of substructures extended, or upon exhaustion of the search space. Once the search terminates and Subdue returns the list of best substructures found, the graph can be compressed using the best substructure. The compression procedure replaces all instances of the substructure in the input graph by single vertices, which represent the substructure de£nition. Incoming and outgoing edges to and from the replaced instances will point to, or originate in the new vertex that represents the instance. The Subdue algorithm can be invoked again on this compressed graph. This procedure can be repeated a user-speci£ed number of times, and is referred to as an iteration.

Subdue's search is guided by the minimum description length (MDL) [20] principle, which seeks to minimize the description length of the entire data set. The evaluation heuristic based on the MDL principle assumes that the best substructure is the one that minimizes the description length of the input graph when compressed by the substructure [4]. The description length of the substructure S given the input graph G is calculated as $DL(G, S) = DL(S)+DL(G|S)$, where $DL(S)$ is the description length of the substructure, and $DL(G|S)$ is the description length of the input graph compressed by the substructure. Description length

[1] Subdue source code, sample datasets and publications are available at http://ailab.uta.edu/subdue.



(a) Input (b) Substructure (c) Compressed

Figure 1: Example of Subdue's substructure discovery capability.

$DL()$ is calculated as the number of bits in a minimal encoding of the graph. Subdue seeks a substructure S that minimizes $DL(G, S)$.

As an example, Figure 1a shows a collection of geometric objects described by their shapes and their "ontop" relationship to one another. Figure 1b shows the graph representation of a portion ("triangle on square") of the input graph for this example and also represents the substructure minimizing the description length of the compressed graph. Figure 1c shows the input example after being compressed by the substructure.

3.2 Graph-based Clustering

Given the ability to £nd a prevalent subgraph pattern in a larger graph and then compress the graph with this pattern, iterating over this process until the graph can no longer be compressed will produce a hierarchical, conceptual clustering of the input data. On the i^{th} iteration, the best subgraph S_i is used to compress the input graph, introducing new vertices labeled S_i in the graph input to the next iteration. Therefore, any subsequently-discovered subgraph S_j can be de£ned in terms of one or more S_i, where $i < j$. The result is a lattice, where each cluster can be de£ned in terms of more than one parent subgraph. For example, Figure 2 shows such a clustering done on a DNA molecule. Subdue £nds substructures S_1 (C–N) and S_2 (C–C) during earlier iterations, which are used in later iterations to discover substructures S_3 (S_1–S_2) and S_4 (S_2–O). Subdue can then discover the bottom-left substructure S_5 (O–S_2–S_3–O) in terms of previously discovered substructures. See [13] for more information on graph-based clustering.

3.3 Supervised Learning

Extending a graph-based discovery approach to perform supervised learning involves, of course, the need to handle negative examples (focusing on the two-class scenario). In the case of a graph the negative information can come in two forms. First, the data may be in the form of numerous small graphs, or graph transactions, each labeled either positive or negative. Second, data may be composed of two large graphs: one positive and one negative.

The £rst scenario is closest to the standard supervised learning problem in that we have a set of clearly de£ned examples. Figure 3a depicts a simple set of positive and negative

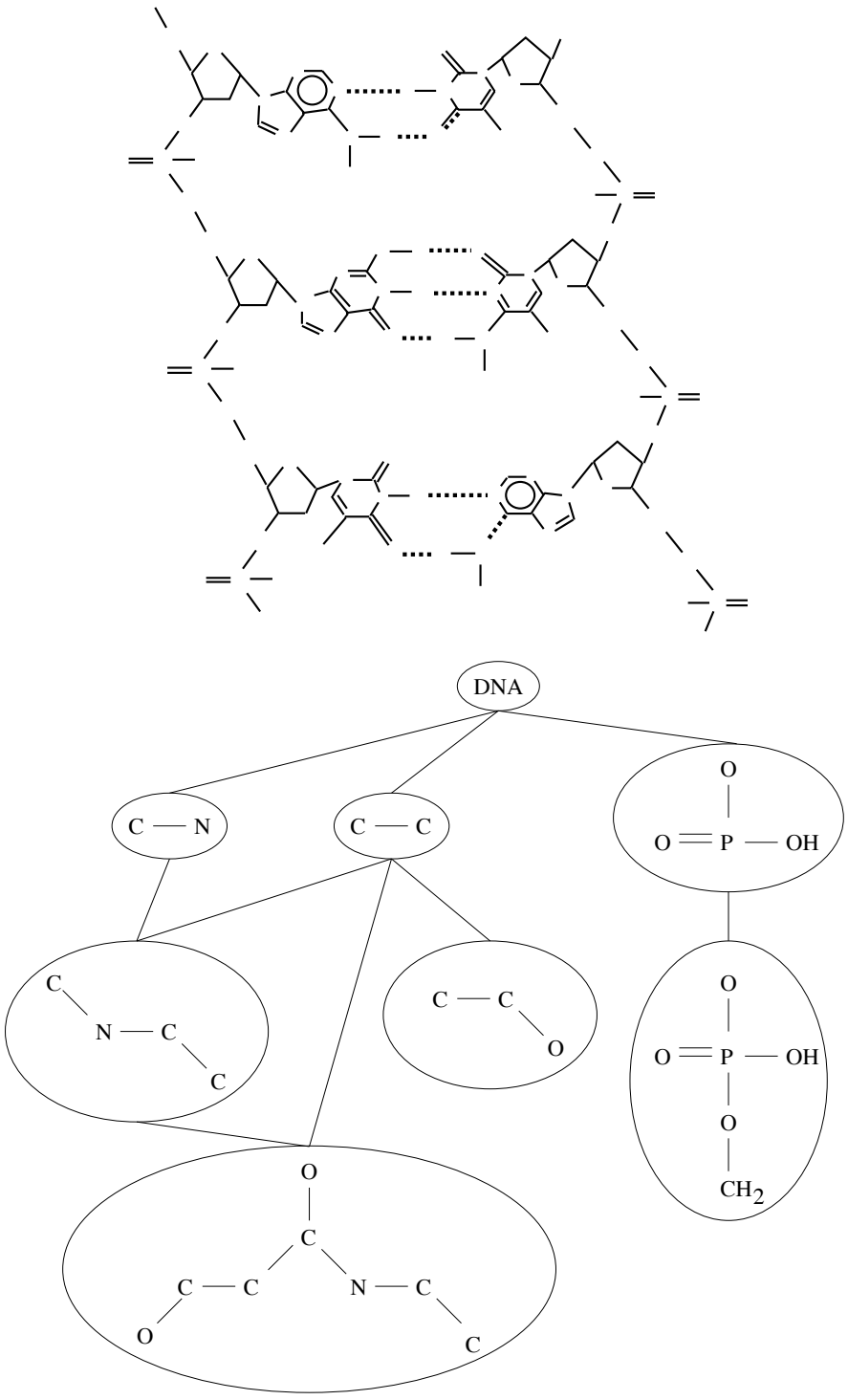


Figure 2: Example of Subdue's clustering (bottom) on a portion of DNA (top).

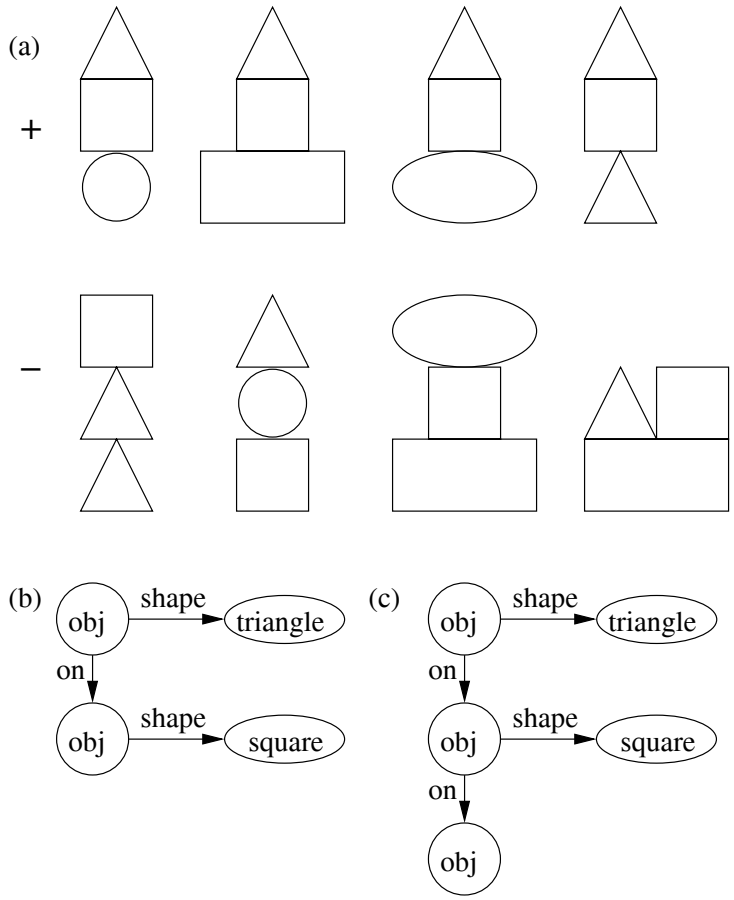


Figure 3: Graph-based supervised learning example with (a) four positive and four negative examples, (b) one possible graph concept, and (c) another possible graph concept.

examples. Let G^+ represent the set of positive graphs, and G^- represent the set of negative graphs. Then, one approach to supervised learning is to £nd a subgraph that appears often in the positive graphs, but not in the negative graphs. This amounts to replacing the information-theoretic measure with simply an error-based measure. For example, we would £nd a subgraph S that minimizes

$$\frac{|\{g \in G^+ | S \not\subseteq g\}| + |\{g \in G^- | S \subseteq g\}|}{|G^+| + |G^-|},$$

where $S \subseteq g$ means S is isomorphic to a subgraph of g. The £rst term of the numerator is the number of false negatives, and the second term is the number of false positives.

This approach will lead the search toward a small subgraph that discriminates well, e.g., the subgraph in Figure 3b. However, such a subgraph does not necessarily compress well, nor represent a characteristic description of the target concept. We can bias the search toward a more characteristic description by using the information-theoretic measure to look for a subgraph that compresses the positive examples, but not the negative examples. If $DL(G)$ represents the description length (in bits) of the graph G, and $DL(G|S)$ represents the description length of graph G compressed by subgraph S, then we can look for an S that minimizes $DL(G^+|S) + DL(S) + DL(G^-) - DL(G^-|S)$, where the last two terms represent the portion of the negative graph incorrectly compressed by the subgraph. While $DL(G^-)$ can be ignored in minimizing over S, its value need be computed only once, and allows the values reported for each substructure to make more sense. This approach will lead the search toward a larger subgraph that characterizes the positive examples, but not the negative examples, e.g., the subgraph in Figure 3c.

Finally, this process can be iterated in a set-covering approach to learn a disjunctive hypothesis. If using the error measure, then any positive example containing the learned subgraph would be removed from subsequent iterations. If using the information-theoretic measure, then instances of the learned subgraph in both the positive and negative examples (even multiple instances per example) are compressed to a single vertex. See [10] for more information on graph-based supervised learning.

3.4 Graph Grammar Learning

As mentioned earlier, two of the advantages of an ILP approach to relational learning are the ability to learn recursive hypotheses and constraints among variables. Graph grammars offer the ability to represent recursive graphical hypotheses [8]. Graph grammars are similar to string grammars except that terminals can be arbitrary graphs rather than symbols from an alphabet. Graph grammars can be divided into two types: node-replacement grammars and hyperedge-replacement grammars. Node-replacement grammars allow non-terminals on vertices, and hyperedge-replacement grammars allow non-terminals on edges. Figure 4b shows an example of a context-free, node-replacement graph grammar. Recent research has begun to develop techniques for learning graph grammars [14, 6].

A graph-based learning approach can be extended to consider graph grammar productions by analyzing the instances of a substructure to see how they relate to each other. If two or more instances are connected to each other by an edge, then a recursive production rule generating an in£nite sequence of such connected subgraphs can be constructed. A slight modi£cation to the information-theoretic measure taking into account the extra information

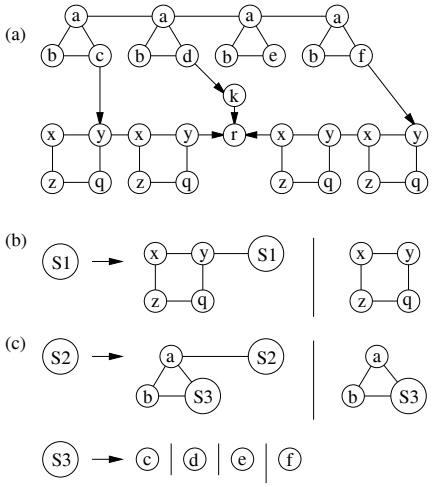


Figure 4: Graph grammar learning example with (a) the input graph, (b) the £rst grammar rule learned, and (c) the second and third grammar rules learned.

needed to describe the recursive component of the production is all that is needed to allow such a hypothesis to compete along side simple subgraphs (i.e., terminal productions) for maximizing compression. The above constraint that the subgraphs be connected by a single edge limits the grammar to be context free. More than one connection between subgraph instances can be considered, and would allow learning context-sensitive grammars, but the algorithm is exponential in the number of connections.

Figure 4b shows an example of a recursive, node-replacement graph grammar production rule learned from the graph in Figure 4a. These productions can be disjunctive, as in Figure 4c, which represents the £nal production learned from Figure 4a using this approach. The disjunctive rule is learned by looking for similar, but not identical, extensions to the instances of a subgraph. A new rule is constructed that captures the variability of the extensions, and is included in the pool of production rules competing based on their ability to compress the input graph. With a proper encoding of this disjunction information, the MDL criterion will tradeoff the complexity of the rule with the amount of compression it affords in the input graph.

An alternative to de£ning these disjunctive non-terminals is to construct a variable whose range consists of the different values of the production. In this way we can introduce constraints among variables contained in a subgraph by adding a constraint edge to the subgraph. For example, if the four instances of the triangle structure in Figure 4a each had another edge to a *c*, *d*, *e* and *f* vertex respectively, then we could propose a new subgraph, where these two vertices are represented by variables, and an equality constraint is introduced between them. If the range of the variable is numeric, then we can also consider inequality constraints between variables and other vertices or variables in the subgraph pattern.

This section has described and demonstrated a number of techniques stemming from an approach to graph-based relational learning and implemented in the Subdue system. The next section applies GBRL, speci£cally the supervised learning technique, to a security-related domain.

Table 1: EAGLE threat event simulator dataset parameters.

Dataset	Parameters		Number of Cases		Number of Groups	
	Observability	Noise	Vulnerability	Productivity	Threat	Non-Threat
1	Low	High	101	106	10	43
2	High	Medium	92	158	12	57
3	Medium	Low	101	106	14	57
4	High	Low	77	133	18	78

4 Application of Subdue to Detecting Security Threats

As part of the U.S. Air Force program on Evidence Assessment, Grouping, Linking and Evaluation (EAGLE), a domain has been built to simulate the evidence available about terrorist groups and their plans prior to their execution. This domain is motivated from an understanding of the real problem of intelligence data analysis. The EAGLE domain consists of a number of concepts, including threat and non-threat actors, threat and non-threat groups, targets, exploitation modes (vulnerability modes are exploited by threat groups, productivity modes are exploited by threat and non-threat groups), capabilities, resources, communications, visits to targets, and transfer of resources between actors, groups and targets. The domain follows a general plan of starting a group, recruiting members with needed capabilities, acquiring needed resources, visiting a target, and then exploiting the target. These events involve various forms of communication and transfer of assets. The EAGLE simulator generates various threat and non-threat groups, and then executes various vulnerability and productivity exploitations. The simulator generates evidence related to all these events, and this evidence is passed through £lters varying the degree of observability and noise in the £nal evidence.

This £nal evidence is the data from which we are to learn. We address two different relational learning problems in this domain. First, we attempt to learn patterns distinguishing vulnerability exploitation cases (terrorist attacks) from productivity exploitation cases (legitimate uses). Second, we attempt to learn patterns distinguishing threat groups from non-threat groups. We address this second problem in two forms: with all evidence and with only communication evidence between actors. We generated a number of datasets using the EAGLE simulator for testing, and four representative datasets are described in Table 1. Each of the four datasets vary in the amount of observability and noise in the data, and the speci£c number of cases and groups in the data.

Before presenting the results of the experimentation, we should address the applicability of a relational learning approach to the problem of £nding patterns that discriminate threat from non-threat events and groups. First, given the fortunate reality that few terrorist events have taken place in the United States, there may not be many positive examples from which to learn. In addition to executed terrorist events, there are a number of terrorist plots and groups which are preempted before completing their mission. Non-U.S. terrorist information may also provide a further source of positive examples. Finally, intelligence analysts can provide arti£cial examples based on their knowledge of the domain. Since we do not have access to much of the real data described above, we rely on the intent of the simulator designers to capture the relevant dimensions of the real problem, which was part of their mandate. The ability of relational learning systems to discover known patterns in the simulator data will increase analysts' con£dence in the patterns discovered in the real data. The simulator has undergone numerous re£nements based on input from intelligence analysts.

Second, while the learning problem in this work is to £nd discriminators between threat

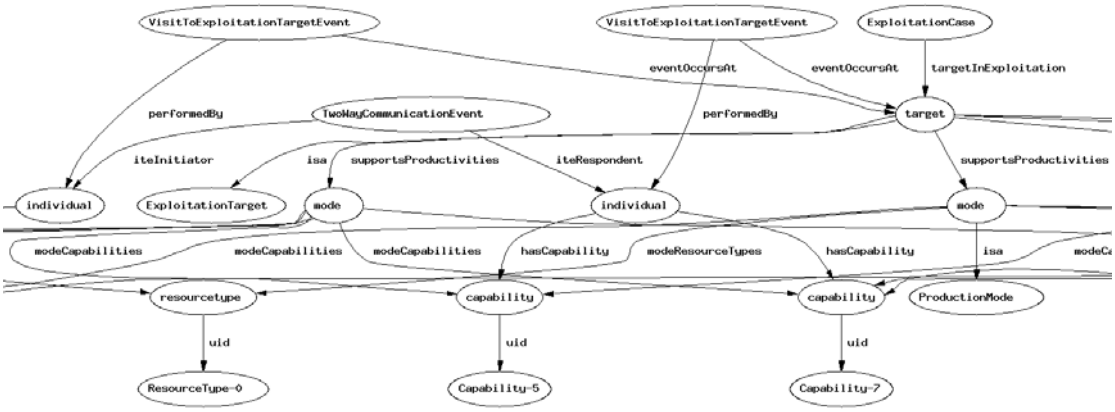


Figure 5: Portion of a vulnerability exploitation case graph drawn using GraphViz [9].

Table 2: Size and performance statistics for learning to distinguish vulnerability from productivity cases.

Dataset	Vertices	Edges	3-fold CV Accuracy	Average Learning Time Per Fold
1	10,617	22,406	58%	51 sec.
2	24,251	61,535	63%	2,912 sec.
3	17,837	55,986	62%	134 sec.
4	47,704	209,556	68%	2,412 sec.

and non-threat events and groups, the primary task in "connecting the dots" is identifying general patterns in terrorist networks and their activities and looking for these patterns in current intelligence data. This task generally falls under the area of link discovery, which includes the task of £nding instances of known patterns in the intelligence data. However, this task is essentially a subgraph isomorphism task, and can be greatly in¤uenced by where in the large graph the search begins. Discriminators, like the ones learned in this work, can help direct the search. Still, some discriminators can represent novel, previously unknown (albeit small) patterns that are indicative of terrorists (e.g., nationality, VISA type, attended ¤ight school). Discriminatory patterns can also be helpful to the analyst who is con£dent in their classi£cation of the event or group, but has not identi£ed an explicit pattern supporting their intuition. Therefore, relational learning can have a bene£cial impact on the task of identifying terrorist events and groups.

4.1 Learning Patterns in Threat Events

In lieu of an exact speci£cation of the EAGLE simulator evidence and our graph representation, we show a portion of a vulnerability case graph in Figure 5. The upper-right "ExploitationCase" vertex is the root vertex of the case. From there, we see the target and its modes, individuals and their capabilities, and visit and communication events. Columns two and three of Table 2 show the number of vertices and edges in all the cases for each dataset.

We used Subdue in supervised learning mode to perform a 3-fold cross-validation experiment on each dataset, where the vulnerability case graphs comprised the positive examples, and the productivity case graphs comprised the negative examples. Figure 6 shows one of the

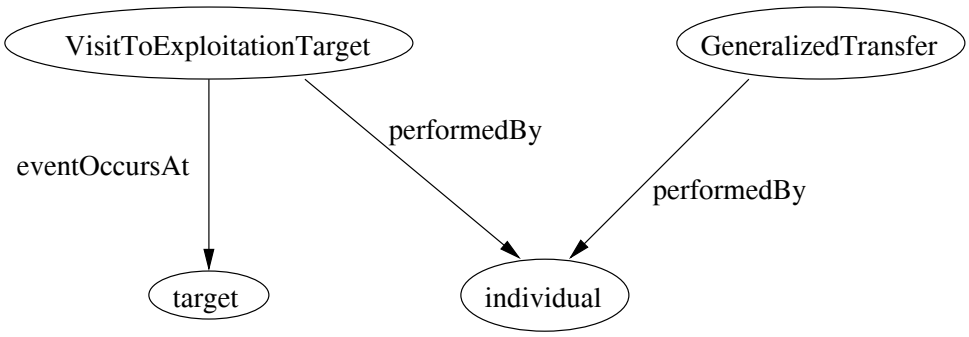


Figure 6: Sample pattern distinguishing vulnerability from productivity cases.

patterns learned for distinguishing vulnerability from productivity cases. This pattern indicates that the same individual both visited the target and was involved in a generalized transfer of a resource. The last two columns of Table 2 show the cross-validation accuracy and the average learning time per fold for each dataset. Note that the testing phase of the cross-validation experiment involves a subgraph isomorphism to determine if the learned pattern is in the test examples. While subgraph isomorphism is known to be NP-Complete, the pattern sizes were small enough to allow testing to £nish in a reasonable amount of time (usually a few days). This time is not included in the timings reported in Table 2.

Results show that Subdue was able to achieve 58-68% accuracy discriminating vulnerability from productivity cases. Not surprisingly, the accuracy increases with higher observability and lower noise. Subdue's learning time was less than 50 minutes even on graphs of size 250,000 vertices and edges. The majority of errors consist of false negatives, which unfortunately, would indicate a vulnerability case being incorrectly classi£ed as a productivity event. One explanation for the majority of false negatives is that these experiments attempt to £nd one, connected graph pattern that can discriminate vulnerability from productivity cases. A better approach would be to learn a disjunction of patterns for the discrimination task, which would correctly classify more of the positive examples. Subdue has the capability of running with multiple iterations in a set-covering mode to learn disjunctive hypotheses. We are investigating this approach to see if such hypotheses can reduce the number of false negatives. Preliminary results indicate that this is indeed the case.

4.2 Learning Patterns in Threat Groups

Figure 7 shows a portion of a threat group graph, which contains persons, their capabilities and resources, and their communications with others in the group. Columns two and three of Table 3 show the number of vertices and edges in all the groups for each dataset. The graph sizes vary considerably both because of the number of groups in the dataset, and because the amount of communication between group members varied considerably due to arbitrary amounts of legitimate communication added beyond that involved in speci£c cases.

We used Subdue in supervised learning mode to perform a 3-fold cross-validation experiment on each dataset, where the threat group graphs comprised the positive examples, and the non-threat group graphs comprised the negative examples. Figure 8 shows one of the patterns learned for distinguishing threat from non-threat groups. This pattern indicates that a group possessing capabilities 6 and 7, along with resource 2, is a threat group. The "mode" vertex

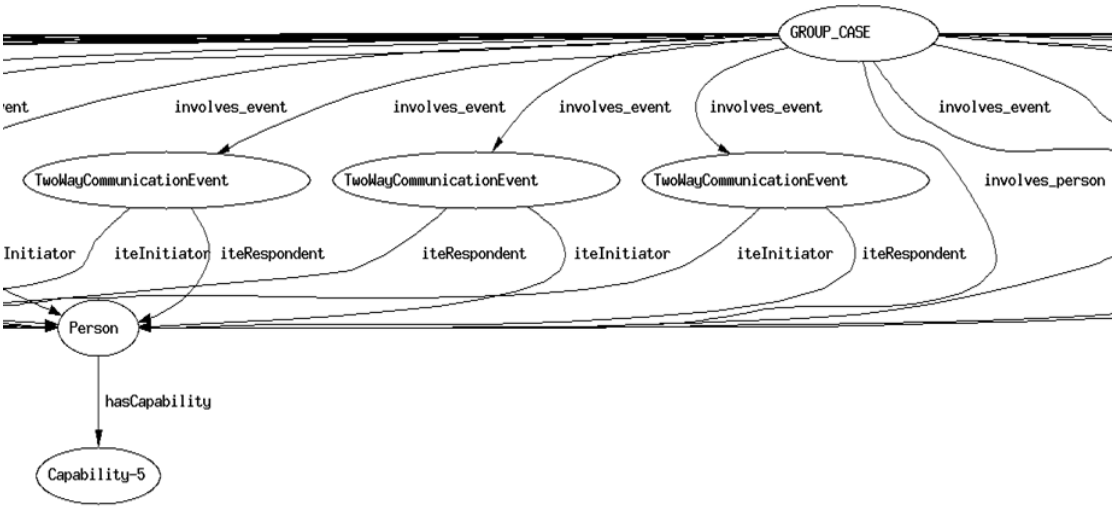


Figure 7: Portion of a threat group graph drawn using GraphViz [9].

Table 3: Size and performance statistics for learning to distinguish threat from non-threat groups.

Dataset	Vertices	Edges	3-fold CV Accuracy	Average Learning Time Per Fold
1	5,813	7,814	78%	2,216 sec.
2	42,962	111,472	81%	1,333 sec.
3	48,381	57,985	93%	11,578 sec.
4	223,427	392,562	85%	9,547 sec.

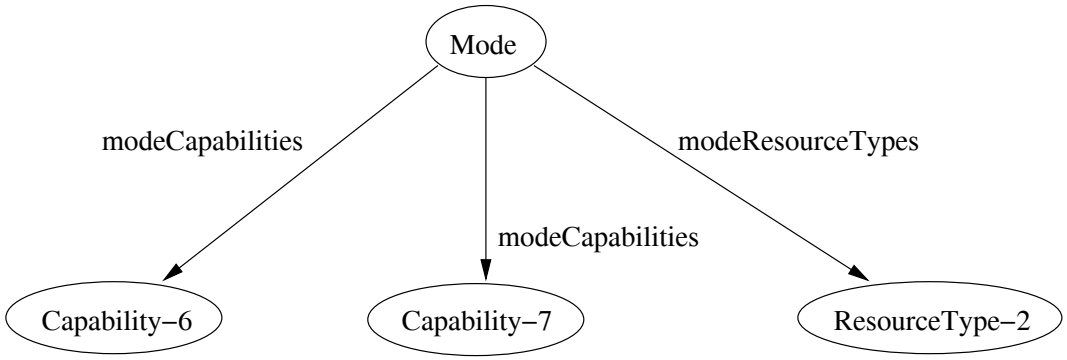


Figure 8: Sample pattern distinguishing threat from non-threat groups.

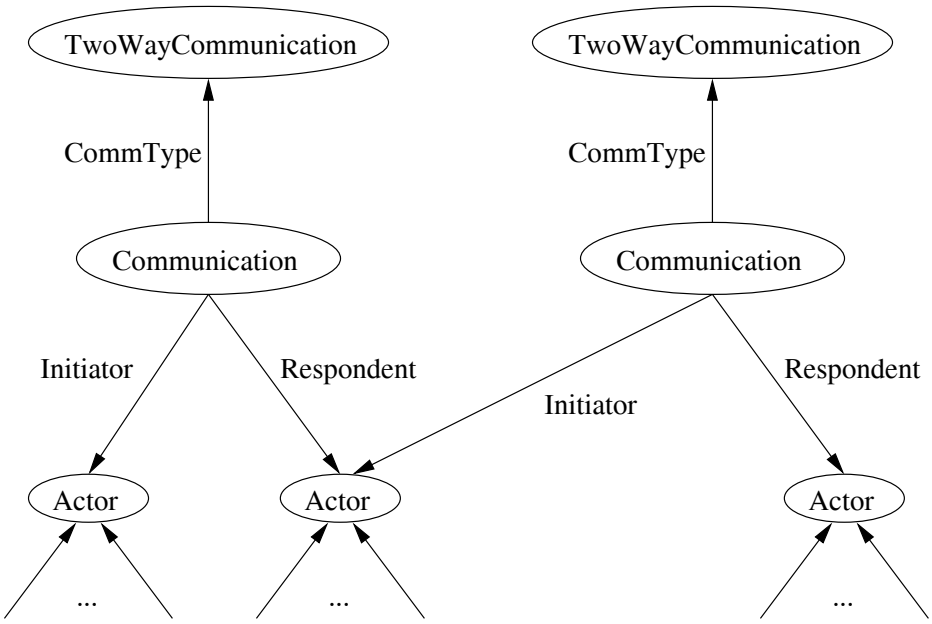


Figure 9: Portion of a group graph using only communication events to connect actors.

is used to collect the capabilities and resources of a group in order to match them to the exploitation modes of targets in the evidence. The EAGLE simulator does not assign meaning to the capabilities and resources, but an example capability might be driving a truck, and an example resource might be explosives.

The last two columns of Table 3 show the cross-validation accuracy and the average learning time per fold for each dataset. Results show that Subdue was able to achieve 78-93% accuracy discriminating threat from non-threat groups. Here, lower noise had the most signi£cant effect on performance, while higher observability had less of an effect. Subdue's learning time was less than 200 minutes even on graphs of size 600,000 vertices and edges. As with the previous experiments on cases, a majority of the errors are false negatives. And as discussed in the last section, learning a disjunction of substructures using a set-covering approach will reduce these types of errors.

4.3 Learning Communication-Only Patterns in Threat Groups

The last experiment investigates Subdue's ability to learn patterns in threat groups using only communications relationships between the actors of a group. Figure 9 shows a portion of such a graph, where each actor participates as the initiator or respondent in a communications event. This simpli£ed graph representation is more akin to typical social network representations [25].

The £rst four columns of Table 4 show the parameters of three datasets used for this experiment. These datasets were generated using the same EAGLE simulator, but we used different simulator parameters to control the number of threat and non-threat groups. We used Subdue in supervised learning mode, where the threat group graphs comprised the positive examples, and the non-threat group graphs comprised the negative examples. Figure 10 shows one of the patterns learned for distinguishing threat from non-threat groups This pattern shows a

Table 4: Size and performance statistics for learning to distinguish threat from non-threat groups using only communication.

Threat Groups	Non-Threat Groups	Total Vertices	Total Edges	Training Accuracy	Learning Time
5	20	3,723	4,152	100%	17 sec.
15	85	10,447	12,351	96%	147 sec.
20	85	7,411	8,889	92%	35 sec.

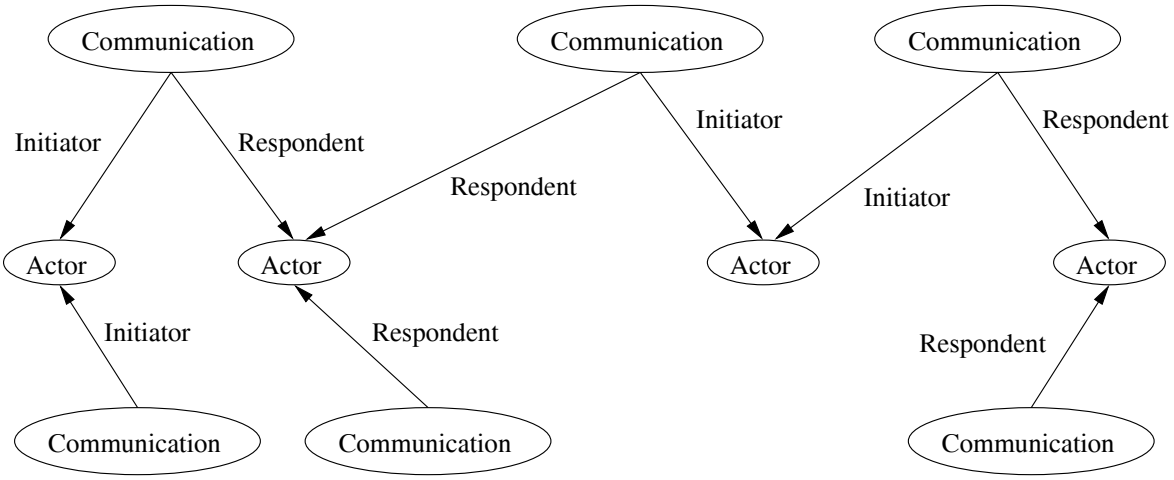


Figure 10: Sample pattern distinguishing threat from non-threat groups using only communication.

set of actors involved in a linear-like communication chain. Such chains are typical of threat groups who attempt to hide their identity by minimizing the amount of communication among members.

The last two columns of Table 4 show the training-set accuracy and the learning time for each dataset. We do not show the results for 3-fold cross-validation, because the patterns learned in the experiment were signi£cantly larger than in previous experiments, and the cost of the subgraph isomorphism testing needed for cross-validation became prohibitive. The results show that Subdue was able to achieve 92-100% training-set accuracy discriminating threat from non-threat groups using only communication evidence. Subdue's learning time was less than 3 minutes even on graphs of size 22,000 vertices and edges. While the accuracy values are dif£cult to generalize, the chain-like communication patterns discovered by Subdue, which are known properties of terrorist communication, indicate that Subdue is able to learn relevant patterns in this domain.

5 Conclusions

The necessity to "connect the dots" [21] in order to combat threats to homeland security requires explicit representation of relational information and the ability to discover patterns in relational data. Graphs are a natural representation for many relational domains. Our approach to graph-based relational learning, and its implementation in the Subdue system, offers the ability to discover patterns in data represented as a graph. These patterns can take the form of prevalent subgraphs, a hierarchical, conceptual clustering of subgraphs, or a subgraph that can

distinguish positive graphs from negative graphs. Results on simulated threat data indicate that Subdue is effective and ef£cient in learning patterns to distinguish threats from non-threats, especially when focusing on groups and communications between group members. These results show the potential of graph-based relational learning for helping intelligence analysts better identify and assess possible security threats.

There are several future directions for our graph-based relational learning research that will improve our ability to handle security-related data. First, such data typically streams in rapidly, preventing our running Subdue from scratch each time new data is available. We need to develop incremental methods for augmenting the data graph and re-evaluating candidate patterns in the light of new data. Second, the need to clearly separate positive and negative examples can be dif£cult or impossible. For example, we may not be certain of an individuals membership in a known threat group. We need to develop methods for learning in what we call *supervised graphs*, where each vertex and edge of the input graph can be annotated with a degree of participation in the various categories of interest (e.g., membership in a threat group). Finally, improved scalability of graph operations is necessary to learn patterns, evaluate their accuracy on test cases, and ultimately to use the patterns to £nd matches in future intelligence data. The graph and subgraph isomorphism operations are a signi£cant bottleneck to these capabilities. We need to develop faster and approximate versions of these operations to improve the scalability of graph-based relational learning.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the Twentieth Conference on Very Large Databases*, pages 487–499, 1994.

[2] R. M. Cameron-Jones and J. R. Quinlan. Ef£cient top-down induction of logic programs. *SIGART Bulletin*, 5(1):33–42, 1994.

[3] D. Cook and L. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[4] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Arti£cial Intelligence Research*, 1:231–255, 1994.

[5] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 30–36, 1998.

[6] S. Doshi, F. Huang, and T. Oates. Inferring the structure of graph grammars from data. In *Proceedings of the International Conference on Knowledge-Based Computer Systems*, 2002.

[7] S. Dzeroski and N. Lavrac, editors. *Relational Data Mining*. Springer, 2001.

[8] H. Ehrig, G. Engels, H. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools*. World Scienti£c, 1999.

[9] E. Gansner, E. Koutso£os, and S. North. *Drawing Graphs with dot*. AT&T Bell Labs, February 2002.

[10] J. Gonzalez, L. Holder, and D. Cook. Graph-based relational concept learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[11] L. Holder and D. Cook. Graph-based relational learning: Current and future directions. *ACM SIGKDD Explorations*, 5(1):90–93, 2003.

[12] A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.

[13] I. Jonyer, D. Cook, and L. Holder. Graph-based hierarchical conceptual clustering. *Journal of Machine Learning Research*, 2:19–43, 2001.

[14] I. Jonyer, L. Holder, and D. Cook. Concept formation using graph grammars. In *Proceedings of the KDD Workshop on Multi-Relational Data Mining*, 2002.

[15] H. Kashima and Akihiro Inokuchi. Kernels for graph classi£cation. In *Proceedings of the International Workshop on Active Mining*, 2002.

[16] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the First IEEE Conference on Data Mining*, 2001.

[17] S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

[18] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[19] D. Page and M. Craven. Biological applications of multi-relational data mining. *ACM SIGKDD Explorations*, 5(1):69–79, 2003.

[20] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scienti£c, 1989.

[21] U.S. Senate and House Committees on Intelligence. Joint inquiry into intelligence community activities before and after the terrorist attacks of september 11, 2001, December 2002.

[22] S. Slattery and M. Craven. Combining statistical and relational methods for learning in hypertext domains. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 38–52, 1998.

[23] A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proceedings of the Fourth International Conference on Inductive Logic Programming*, 1994.

[24] M. Turcotte, S. H. Muggleton, and M. J. E. Sternberg. Application of inductive logic programming to discover rules governing the three-dimensional topology of protein structure. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 53–64, 1998.

[25] S. Wasserman and K. Faust. *Social Network Analysis*. Cambridge University Press, 1994.

[26] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the International Conference on Data Mining*, 2002.

[27] K. Yoshida, H. Motoda, and N. Indurkhya. Graph-based induction as a uni£ed learning framework. *Journal of Applied Intelligence*, 4:297–328, 1994.