CHAPTER 7

UNSUPERVISED AND SUPERVISED PATTERN LEARNING IN GRAPH DATA

Diane J. Cook, Lawrence B. Holder, and Nikhil Ketkar [1]

University of Texas at Arlington

{cook,holder,ketkar}@cse.uta.edu

7.1 INTRODUCTION

The success of machine learning and data mining for business and scientific purposes has fueled the expansion of its scope to new representations and techniques. Much collected data is structural in nature, containing entities as well as relationships between these entities. Compelling data in bioinformatics [32], network intrusion detection [15], web analysis [2, 8], and social network analysis [7, 27] has become available that requires effective handling of structural data. The ability to learn

[1]This work is partially supported by the National Science Foundation grants IIS-0505819 and IIS-0097517.

ISBN 0-000-00000-0

concepts from relational data has also 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" [28], that is, focusing on the relationships between entities in the data, rather than merely on an entity's attributes.

In this chapter we describe an approach to mining concepts from graph-based data. We provide an algorithmic overview of the approach implemented in the Subdue system and describe its uses for unsupervised discovery and supervised concept learning. This book details several different approaches for performing these same tasks. To assist in understanding the relationships between these approaches, we also perform experimental comparisons between Subdue and frequent subgraph miners and ILP learning methods. We observe that each type of approach offers unique advantages for certain classes of applications and all face challenges that will drive future research.

7.2 MINING GRAPH DATA USING SUBDUE

The Subdue graph-based relational learning system[2] [3, 4] 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 $G = (V, E, \mu, \nu)$ 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 μ and ν represent node and edge labeling functions, as described in the earlier chapter by Bunke and Neuhaus. 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

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

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 classified as positive or negative examples.

7.2.1 Substructure Discovery

As an unsupervised discovery algorithm, Subdue searches for a substructure, or subgraph of the input graph, that best compresses the input graph. Subdue uses a variant of beam search for its main search algorithm, as summarized in Figure 7.1.. A substructure in Subdue consists of a subgraph definition and all its instances 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 compression (or sometimes referred to as value) as calculated using the MDL principle described below. Only the top *beam* substructures remain on the queue for expansion during the next pass through the main discovery loop.

The search terminates upon reaching a limit on the number of substructures extended (this defaults to half the number of edges in the graph), or upon exhaustion of the search space. Once the search terminates and Subdue returns the list of best substructures, 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 definition. Incoming and outgoing edges to and from the replaced instances will point to or originate from the new vertex that represents the instance. The Subdue algorithm can be invoked again on this compressed graph. This procedure can be repeated multiple times, and is referred to as an iteration.

Subdue's search is guided by the Minimum Description Length (MDL) [26] principle given in Equation 7.1, where $DL(S)$ is the description length of the substructure being evaluated, $DL(G|S)$ is the description length of the graph as compressed by the substructure, and $DL(G)$ is the description length of the original graph. The best substructure is the one that minimizes this compression ratio.

$$Compression = \frac{DL(S) + DL(G|S)}{DL(G)} \tag{7.1}$$

As an example, Figure 7.2. shows the four instances that Subdue discovers of a pattern S_1 in the example input graph and the resulting compressed graph, as well as the pattern S_2 found in this new graph and the resulting compressed graph. To allow slight variations between instances of a discovered pattern (as is the case in Figure 7.2.), Subdue applies an error-tolerant graph match between the substructure definition and potential instances (see the chapter by Bunke and Neuhaus for more details on this topic). The discovery algorithm can be biased by incorporating prior knowledge in the form of predefined substructures or preference weights on desirable (undesirable) edge and vertex labels. As shown by Rajappa, Subdue's run time is polynomial in the size of the input graph [25]. Substructure discovery using Subdue has yielded expert-evaluated significant results in domains including predictive toxicology, network intrusion detection, earthquake analysis, web structure mining, and protein data analysis [16, 23, 30].

```
SUBDUE(Graph,BeamWidth,MaxBest,MaxSubSize, Limit)
    ParentList = Null;
    ChildList = Null;
    BestList = Null;
    ProcessedSubs = 0;
    Create a substructure from each unique vertex label
        and its single-vertex instances;
    Insert the resulting substructures in ParentList;
    while ProcessedSubs ≤ Limit
        and ParentList not empty
    do
        while ParentList is not empty
        do
            Parent = RemoveHead(ParentList);
            Extend each instance of Parent in all possible ways;
            Group the extended instances into Child substructures;
        for each Child
        do
            if SizeOf(Child) less than MaxSubSize
            then
                Evaluate Child;
                Insert Child in ChildList in order by value;
                if BeamWidth < Length(ChildList)
                then
                    Destroy substructure at end of ChildList;
        Increment ProcessedSubs;
        Insert Parent in BestList in order by value;
        if MaxBest < Length(BestList)
        then
            Destroy substructure at end of BestList;
        Switch ParentList and ChildList;
    return BestList;
```

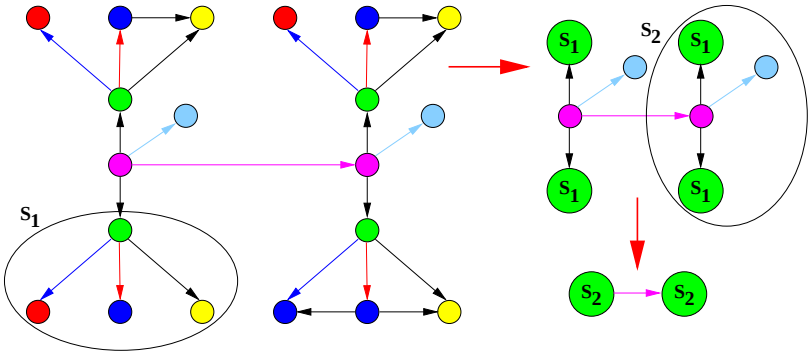

Figure 7.2. Example of Subdue's substructure discovery capability.

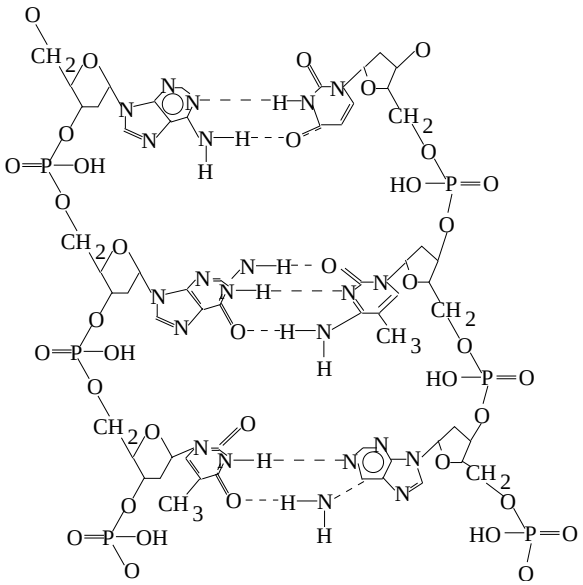

Figure 7.3. Visualization of a portion of DNA.

7.2.2 Graph-based Clustering

Given the ability to find 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.

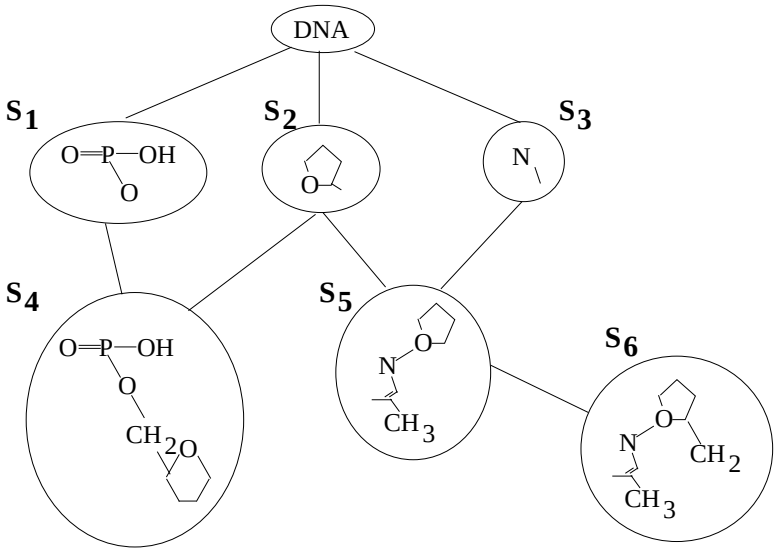

Figure 7.4. Subdue's hierarchical cluster generated from DNA data.

Therefore, any subsequently-discovered subgraph S_j can be defined in terms of one or more S_i, where $i < j$. The result is a lattice, where each cluster can be defined in terms of more than one parent subgraph. For example, Figure 7.4. shows such a clustering extracted from the graph representation of a DNA molecule (visualized in Figure 7.3.) [10]. Note that in this cluster hierarchy, for example, subgraph S_4 is composed of an S_1 and and S_2 together with a CH_2.

7.2.3 Supervised Learning

Extending a graph-based discovery approach to perform supervised learning introduces 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.

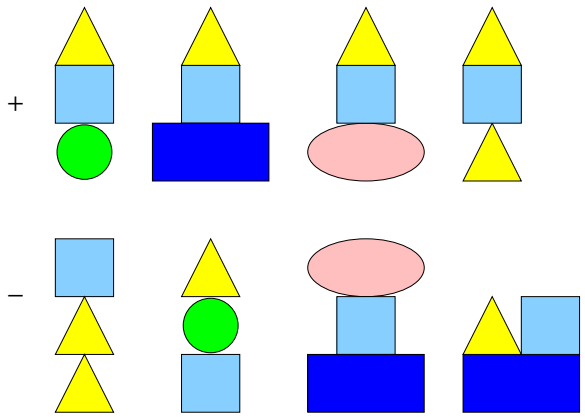

Figure 7.5. Visualization of graph-based data with four positive and four negative examples.

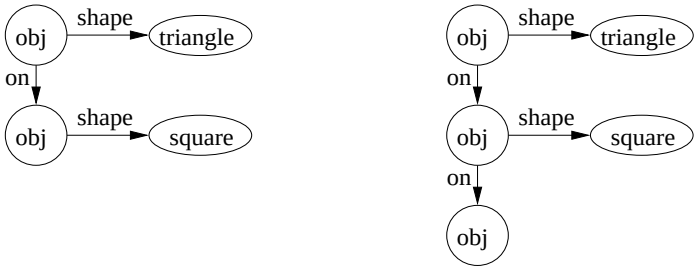

Figure 7.6. Two possible graph concepts learned from example data.

The first scenario is closest to the standard supervised learning problem in that we have a set of clearly-defined examples. Figure 7.5. depicts a simple set of positive (G^+) and negative (G^-) examples. One approach to supervised learning is to find a subgraph that appears in many positive graphs, but in few negative graphs. This amounts to replacing the compression-based measure with an error-based measure. For example, we would find a subgraph S that minimizes

$$\frac{|\{g \in G^+ | S \not\subseteq g\}| + |g \in G^- | S \subseteq g\}|}{|G^+| + |G^-|} = \frac{FN + FP}{P + N}, \tag{7.2}$$

where $S \subseteq g$ means S is isomorphic to a subgraph of g (although we do not actually perform a subgraph isomorphism test during learning). The first 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 on the left in Figure 7.5.. 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 compression-based 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. This approach will lead the search toward a larger subgraph that characterizes the positive examples, but not the negative examples, e.g., the subgraph on the right in Figure 7.5..

Finally, this process can be iterated in a set-covering approach to learn a disjunctive hypothesis. Using the error measure, any positive example containing the learned subgraph would be removed from subsequent iterations. Using the

compression-based measure, instances of the learned subgraph in both the positive and negative examples (even multiple instances per example) are compressed to a single vertex.

7.3 COMPARISON TO OTHER GRAPH-BASED MINING ALGORITHMS

Because graph-based data mining has demonstrated success for a variety of tasks in structural domains (see Chapters 14 through 17 of this book for examples), a number of varied techniques and methodologies have arisen for mining interesting subgraph patterns from graph datasets. These include mathematical graph theory-based approaches like FSG ([14] and Chapter 6 of this book), gSpan ([33] and Chapter 5 of this book), greedy search-based approaches like Subdue or GBI [17], inductive logic programming (ILP) approaches such as used by WARMR [6], and kernel function-based approaches ([11] and Chapter 11 of this book). In this chapter we contrast the methodologies employed by Subdue with frequent substructure and ILP approaches, and attempt to highlight differences in discoveries that can be expected from the alternative techniques.

7.4 COMPARISON TO FREQUENT SUBSTRUCTURE MINING APPROACHES

Mathematical graph theory-based approaches mine a complete set of subgraphs mainly using a support or frequency measure. The initial work in this area was the AGM [9] system which uses the Apriori level-wise approach. FSG takes a similar approach and further optimizes the algorithm for improved running times. gFSG [12] is a variant of FSG which enumerates all geometric subgraphs from the database. gSpan uses DFS codes for canonical labeling and is much more memory and computationally efficient than previous approaches. Instead of mining all subgraphs, CloseGraph [34] 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. 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 and GBI. Subdue uses MDL-based compression heuristics, and 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.

Methodologies that focus on complete, frequent subgraph discovery, such as FSG and gSpan, are guaranteed to find all subgraphs that satisfy the user-specified constraints. Although completeness is a fundamental and desirable property, a side effect of the complete approaches is that these systems typically generate a large number of substructures, which by themselves provide relatively less insight about the domain. As a result, interesting substructures have to be identified from the large set of substructures either by domain experts or by other automated methods in order to achieve insights into this domain. In contrast, Subdue typically produces a smaller number of substructures which best compress the graph dataset, and which can provide important insights about the domain.

A thorough comparison of Subdue with these approaches is difficult because most frequent subgraph discovery approaches are designed to look for patterns that are frequent across a disjoint set of graph transactions, rather than to find patterns that are frequent or interesting within a single graph (see [31, 13] for some representation-limited exceptions). Another key distinction of Subdue from many other graph-based methods is that Subdue can accommodate a free-form graph representation, where others often have a specific representation tailored to transactional data, which is a restricted form of relational data. Table 7.4 lists for Subdue, FSG, and gSpan, whether these systems can process directed graphs, multi-graphs, graphs with self-edges, as well as restrictions on labels or graph size.

	Types of graphs				
	Directed	Labels	Multigraph	Self-edges	#vertices/edges
Subdue	√	Alpha-numeric	√	√	Unlimited
FSG	√	Alpha-numeric	×	×	Unlimited
gSpan	×	Numeric	×	×	Max 254/254

Table 7.1. Acceptable input graphs.

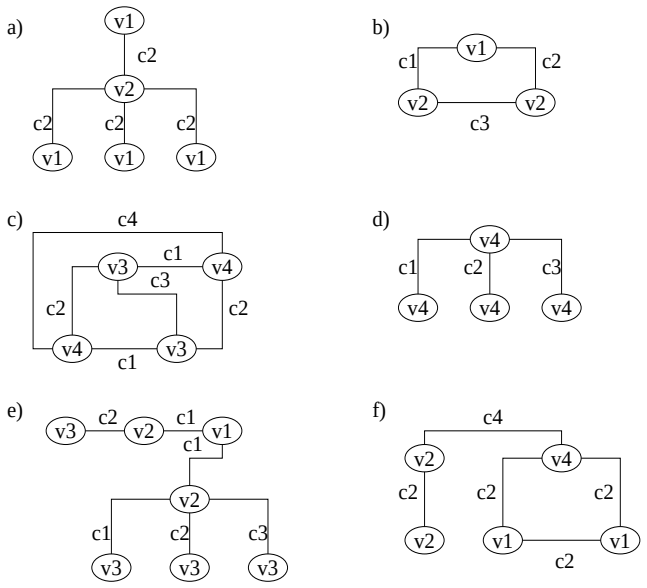

Figure 7.7. Embedded substructures.

These restrictions are based on the original algorithm and could be superseded by modifications.

7.4.1 Experiments Using Synthetic Datasets

We demonstrate the advantage of Subdue's compression-based methodology by performing an experimental comparison of Subdue with the graph-based data mining

systems FSG and gSpan on a number of artificial datasets. We generate graph datasets with 500, 1000, 1500 and 2000 transactions by embedding one of the six substructures shown in Figure 7.7.. Each of the generated twenty-four datasets (six different embedded substructures, each with 500, 1000, 1500 and 2000 transactions) have the following properties:

1. 60% of the transactions have the embedded substructure: the rest of the transactions are generated randomly.

2. For all the transactions that contain the embedded substructure, 60% of the transaction is the embedded substructure, that is, coverage of the embedded substructure is 60%.

Since each of the twenty-four datasets have both properties listed above, it is clear that the embedded substructure is the most interesting substructure in each of the datasets. Using these datasets we now compare the performance of Subdue, FSG and gSpan. For each of the twenty-four datasets, Subdue was run with the default parameters and FSG and gSpan were run at a 10% support level. We want to determine if Subdue can find the best substructure despite its greedy search approach. We also want to compare quality of the reported substructure based on an interestingness factor, for which we use compression. This is motivated by work in information theory that uses this measure to identify the pattern which best describes the information in the data. We use a slightly different calculation of compression in the experiments than the one employed by the Subdue algorithm in order to remove a level of bias in the results. Lastly, we want to compare the run times of the three algorithms on a variety of datsets.

Table 7.4.1 summarizes the results of the experiments and Table 7.4.1 summarizes the runtimes of Subdue, FSG and gSpan. The results indicate that Subdue discovers the embedded substructure and reports it as the best substructure about

#Transactions	% Cases where Subdue reported embedded substructure as best	#Substructures generated by FSG/gSpan
500	66%	233495
1000	83%	224174
1500	83%	217477
2000	78%	217479
Average	79%	223156

Table 7.2. Results (average) on six artificial datasets.

#Transactions	FSG	gSpan	Subdue
500	734	61	51
1000	815	107	169
1500	882	182	139
2000	1112	249	696
Average	885	150	328

Table 7.3. Run times (seconds) on six artificial datasets.

80% of the time. Both FSG and gSpan generate approximately 200,000 substructures among which there exists the embedded substructure. The runtime of Subdue is intermediate between FSG and gSpan. Subdue clearly discovers and reports fewer but more interesting substructures. Although it could be argued that setting a higher support value for FSG and gSpan can lead to fewer generated substructures, it should be noted that this can cause FSG and gSpan to miss the interesting pattern. We observed that the increase in run time for Subdue is nonlinear when we increase the size of the dataset. Increase for FSG and gSpan was observed to be linear (largely because of various optimizations). The primary reason for this behavior is the less efficient implementation of graph isomorphism in Subdue than in FSG and gSpan. A more efficient approach for graph isomorphism, with the use of canonical labeling, needs to be developed for Subdue.

7.4.2 Experiments Using Real Datasets

In addition, we performed an experimental comparison of Subdue with gSpan and FSG on the chemical toxicity and the chemical compounds datasets that are provided with gSpan. The chemical toxicity dataset has a total of 344 transactions. There are 66 different edge and vertex labels in the dataset. FSG and gSpan results were recorded based on a 5% support threshold. From our experiments we determined the support of any best compressing substructure should be greater than 10% for the dataset of size greater than 50. Therefore, we selected 5% support. If support is set to a lesser value, large numbers of random and insignificant patterns are generated.

Table 7.4.2 summarizes the results of the experiment, and Figure 7.8. shows the best compressing substructure discovered by Subdue. Because gSpan supports only numeric labels, atom and bond labels have been replaced by unique identifying numbers in this database. As the results indicate, Subdue discovered frequent

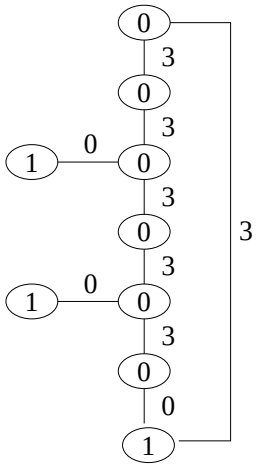

Figure 7.8. Best compressing substructure discovered by Subdue on the chemical toxicity dataset.

Compression from Subdue's best substructure	16%
Best compression from any FSG/gSpan substructure	8%
Number of substructures reported by FSG/gSpan	844
Runtime Subdue (seconds)	115
Runtime FSG (seconds)	8
Runtime gSpan (seconds)	7

Table 7.4. Results from the chemical toxicity dataset.

patterns missed by FSG and gSpan that resulted in greater compression. However, the runtime of Subdue is much larger than that of FSG and gSpan.

The chemical compounds dataset has a total of 422 transactions. There are 21 different edge and vertex labels in the dataset. The results for FSG and gSpan were recorded based on a 10% support threshold. We selected 10% support because if support is set to a lesser value, large numbers of random and insignificant patterns

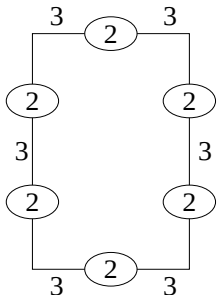

Figure 7.9. Best compressing substructure discovered by Subdue on the chemical compound dataset.

Compression from Subdue's best substructure	19%
Best compression from any FSG/gSpan substructure	7%
Number of substructures reported by FSG/gSpan	15966
Runtime Subdue (seconds)	142
Runtime FSG (seconds)	21
Runtime gSpan (seconds)	4

Table 7.5. Results from the chemical compounds dataset.

are generated. Table 7.4.2 summarizes the results of the experiment, and Figure 7.9. shows the best compressing substructure discovered by Subdue. Again, Subdue found better-compressing substructures but required a greater runtime.

7.5 COMPARISON TO ILP APPROACHES

Logic-based mining, popularly known as Inductive Logic Programming (ILP) [18], is characterized by the use of logic for the representation of structural data. ILP systems represent examples, background knowledge, hypotheses and target concepts in Horn clause logic. The core of ILP is the use of logic for representation and the

search for syntactically legal hypotheses constructed from predicates provided by the background knowledge. ILP systems such as FOIL [24], CProgol [20], Golem [22] and WARMR [6] have been extensively applied to supervised learning and to a certain extent to unsupervised learning.

In contrast, graph-based approaches are characterized by representation of structural data in the form of graphs. Graph-based approaches represent examples, background knowledge, hypotheses and target concepts as graphs. Here we perform a qualitative comparison of Subdue and logic-based mining approaches.

By performing a comparison of the graph-based and logic-based approaches, we intended to analyze the ability of the approaches to efficiently discover complex structural concepts and to effectively utilize background knowledge. To do so, we must establish some notions on the complexity of a structural concept and identify the types of background knowledge generally available for a mining task.

The complexity of a multi-relational, or structural, concept is a direct consequence of the number of relations in the concept. For example, learning the concept of arene (a chemical compound with a six-member ring as in benzene), which comprises learning six relations, involves the exploration of a larger portion of the hypothesis space than learning the concept of hydroxyl (oxygen connected to hydrogen as in methanol), which comprises learning one relation. The concept of arene is thus more complex than that of hydroxyl.

Although the number of relations in the concept is a key factor in the complexity of the concept, there are also other factors such as the number of relations in the examples from which the concept is to be learned. For example, learning the concept of hydroxyl from a set of phenols (hydroxy group attached to an arene) involves the exploration of a larger hypothesis space than learning the same hydroxyl concept from a set of alcohols (hydroxy group attached to an alkyl). The concept of a

hydroxyl group is thus more complex to learn from phenols than it is from a set of alcohols. We identify this complexity as *structural complexity*.

In order to learn a particular concept, it is essential that the representation used by a data mining system be able to express that particular concept. For a representation to express a particular concept, it is beneficial to have both the syntax which expresses the concept and the semantics which associates meaning to the syntax. A relational concept can be said to have a *greater complexity* than some other relational concept if it requires a more expressive representation. To learn numerical ranges, for example, it is essential to have the syntax and the semantics to represent notions like "less than", "greater than" and "equal to". We identify this complexity as *semantic complexity*.

A relational learner can be provided background knowledge which condenses the hypothesis space. For example, if the concept to be learned is "compounds with three arene rings" and the concept of an arene ring is provided as a part of the background knowledge, then the arene rings in examples could be condensed to a single entity. This would cause a massive reduction in the hypothesis space required to be explored to learn the concept, and the mining algorithm would perform more efficiently than without the background knowledge. We identify such background knowledge as background knowledge intended to condense the hypothesis space.

A mining algorithm can also be provided background knowledge which augments the hypothesis space. For example, consider that the algorithm is provided with background knowledge which allows it to learn concepts like "less than", "greater than" and "equal to". In this case, the algorithm would explore a hypothesis space larger than what it would explore without the background knowledge. Thus, introducing background knowledge has augmented the hypothesis space and has facilitated the learning of concepts which would not be learned without the background

knowledge. We identify such background knowledge as background knowledge intended to augment the hypothesis space.

Using these notions, we now identify the factors on the basis of which the the graph-based and logic-based approaches can be compared. They are:

1. Ability to learn structurally large relational concepts.

2. Ability to learn semantically complex relational concepts or the ability to effectively use background knowledge that augments the hypothesis space to learn semantically complex relational concepts.

3. Ability to effectively use background knowledge that condenses the hypothesis space.

7.5.1 CProgol

The representative for ILP systems that we will use for our experiments is CProgol [19], which is 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, i.e., uses mode declarations. A mode declaration is a constraint which imposes restrictions on the predicates and their arguments appearing in a hypotheses clause. These constraints specify which predicates can occur in the head and the body of hypotheses. They also specify which arguments can be input variables, output variables or constants, as well as the number of alternative solutions for instantiating the predicate.

In CProgol, user-defined mode declarations aid the generation of the most specific 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-subsumption lattice in a general-to-specific manner, bounded from below with respect to the most specific clause. The search strategy is A* 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.

7.5.2 Experiments Using Structurally Large Concepts from the Mutagenesis Dataset

The Mutagenesis dataset [29] has been collected to identify mutagenic activity in a compound based on its molecular structure and is considered to be a benchmark dataset for multi-relational data mining. The Mutagenesis dataset consists of the molecular structure of 230 compounds, of which 138 are labeled as mutagenic and 92 as non-mutagenic. The mutagenicity of the compounds has been determined by the Ames Test. The task is to distinguish mutagenic compounds from non-mutagenic ones based on their molecular structure. The Mutagenesis dataset basically consists of atoms, bonds, atom types, bond types and partial charges on atoms. The dataset also consists of the hydrophobicity of the compound (logP), the energy level of the compounds lowest unoccupied molecular orbital (LUMO), a boolean attribute identifying compounds with 3 or more benzyl rings (I1), and a boolean attribute identifying compounds which are acenthryles (Ia). Ia, I1, logP and LUMO are relevant properties in determining mutagenicity. When run on the entire Mutagenesis dataset, as described above, using 10-fold cross validation, Subdue achieved 63% accuracy with an average learning time per fold of 94 minutes, while CProgol achieved 60% accuracy with an average learning time per fold of 24 minutes. The difference in accuracy was not statistically significant.

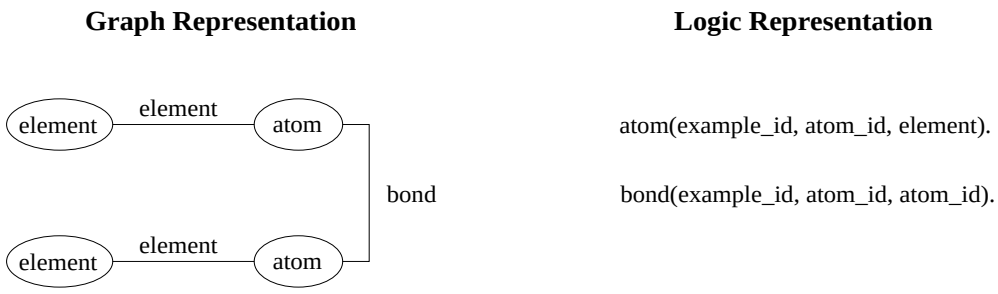

Figure 7.10. Representation for structurally large concepts.

In order to compare the performance of the approaches while learning structurally large concepts we ran Subdue and CProgol on a variant of the Mutagenesis dataset. Since we intended to compare the ability of the approaches to learn large structural concepts, both the mining algorithms were provided only with the basic information of the atoms, the elements and the bonds without any other information or background knowledge. This is shown in Figure 7.10.. The algorithms are not provided with any additional information or any form of background knowledge, because we intended to compare the ability to learn large structural concepts. The introduction of any additional information or background knowledge would prevent this from happening. If systems were provided with the partial charge on the atoms and background knowledge to learn ranges, the systems would learn ranges on partial charges which would contribute to the accuracy. This would make it difficult to analyze how the approaches compared while learning structurally large concepts. Hence the partial charge information and the background knowledge to learn ranges was not given to either system.

The atom type and bond type information was also not provided to either system. The reasoning behind doing so is that we view the atom type and bond type information as a propositional representation of relational data. Such information allows the relational learners to learn propositional representations of relational concepts rather than the true relational concept. Consider for example the rule

found by CProgol on the Mutagenesis dataset [29], atom(A,B,c,195,C). This rule denotes that compounds with a carbon atom of type 195 are mutagenic. The atom type 195 occurs in the third adjacent pair of 3 fused 6 member rings. Therefore all compounds with 3 fused 6 member rings are labeled active. Thus, a rule involving 15 relations (3 fused 6 member rings) has been discovered by learning a single relation. Discovering such a rule has allowed CProgol to learn a propositional representation of a relational concept rather that the true relational concept. Providing atom type and bond type information would allow both systems to learn propositional representations of structurally large relational concepts rather than the true relational concepts. We do not consider the learning of such concepts equivalent to the learning of structurally large relational concepts, and therefore do not provide either system with the atom type and bond type information.

The results of the experiment are shown in Table 7.5.2. For the training set, the accuracy for one run on the entire dataset and the learning time are shown. For the 10-fold cross validation (CV), average learning time over 10 folds is shown. The results show that Subdue performs significantly better than CProgol. Subdue learns 17 graphs representing 17 rules. One of the rules discovered by Subdue is shown in Figure 7.11.. This rule has an accuracy of 76.72% and a coverage of 81.15%. The hypotheses learned by CProgol mostly consisted of a single atom or bond predicate. These results give a strong indication that a graph-based approach can perform better than a logic-based approach when learning structurally large concepts.

7.5.3 Experiments Using Structurally Large Concepts from Artificial Data

We performed additional experiments using artificially generated Bongard problems [1] to reinforce the insights from the experiments on the Mutagenesis dataset. Bongard problems were introduced as an artificial domain in the field of pattern

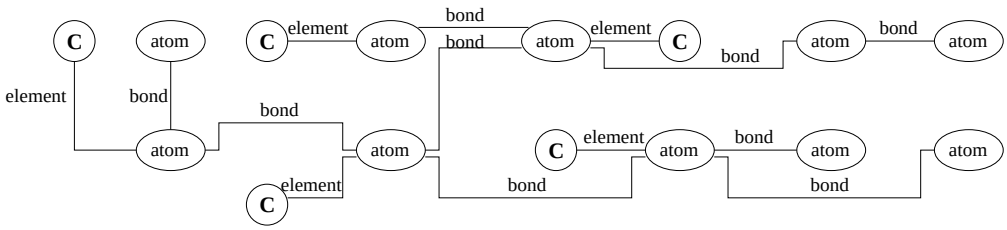

Figure 7.11. Rule discovered by Subdue on the mutagenesis dataset while learning structurally large concepts.

CProgol training set accuracy	60.00%
Subdue training set accuracy	86.00%
CProgol training set runtime (seconds)	2010
Subdue training set runtime (seconds)	1876
CProgol 10-fold CV accuracy	61.74%
Subdue 10-fold CV accuracy	81.58%
CProgol 10-fold CV runtime (average, seconds)	1940
Subdue 10-fold CV runtime (average, seconds)	2100
CProgol - Subdue, Δerror$+/-\delta$	20.84% +/− 12.78%
CProgol - Subdue, confidence	99.94%

Table 7.6. Results on mutagenesis structurally large concepts.

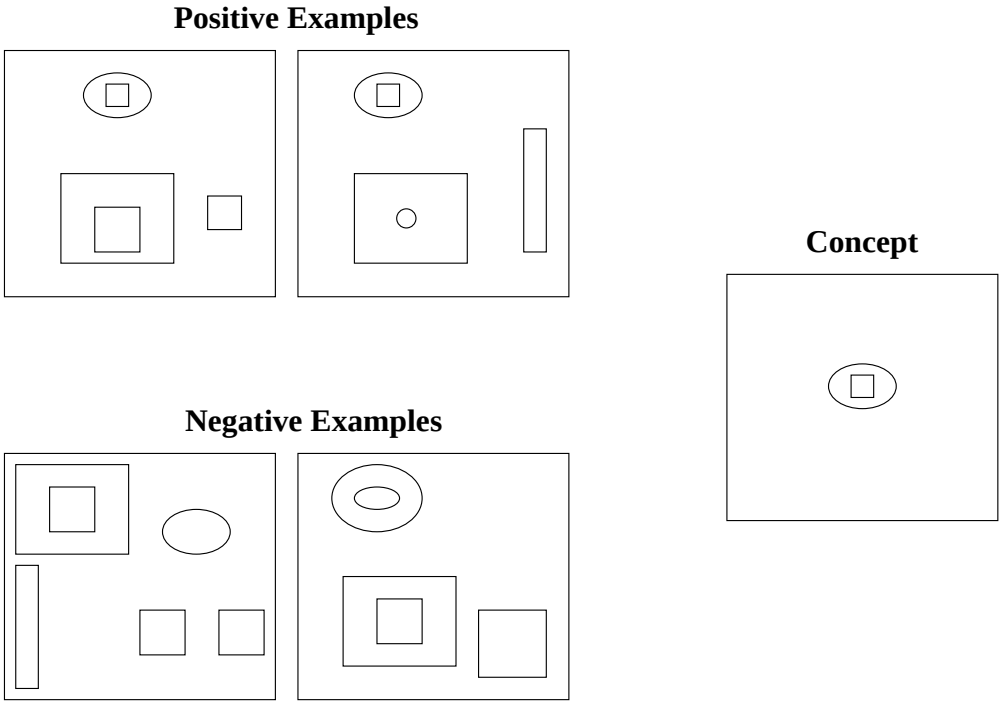

Figure 7.12. A Bongard problem.

recognition. A simplified form of Bongard problems has been used as an artificial domain in the field of ILP [5]. We use a similar form of Bongard problems for our artificial domain experiments. We use a Bongard problem generator to generate datasets as shown in Figure 7.12.. Each dataset consists of a set of positive and negative examples. Each example consists of a number of simple geometrical objects placed inside one another. The task is to determine the particular set of objects, their shapes and their placement which can correctly distinguish the positive examples from the negative ones.

Figure 7.13. shows the representations used for Subdue and CProgol. We systematically analyzed the performance of Subdue and CProgol on artificially-generated Bongard problems with increasing numbers of objects in the concept and increasing numbers of objects in the examples. In the first experiment, the number of objects in the Bongard concept was varied from 5 to 35. The number of additional objects

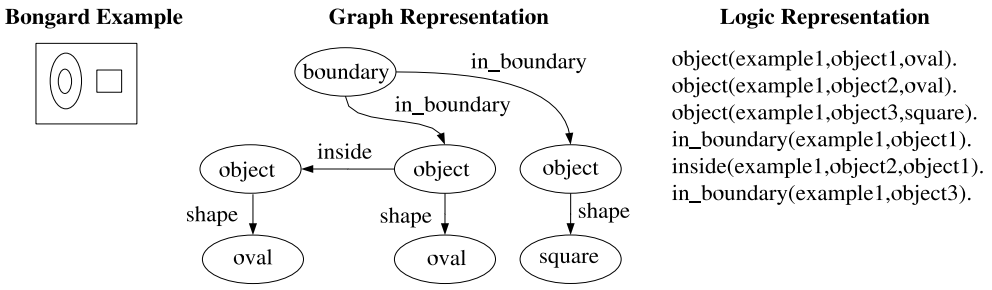

Figure 7.13. Representation for Bongard problems.

in each example (objects which are not part of the concept) were kept constant at 5. For every concept size from 5 to 35, 10 different concepts were generated. For each of the 10 concepts a training set and a test set of 100 positive and 100 negative examples were generated. CProgol and Subdue were run on the training sets and were tested on the test sets.

Figure 7.14. (a) shows the average accuracy achieved by CProgol and Subdue on 10 datasets for every concept size ranging from 5 to 35. It is observed that Subdue clearly outperforms CProgol. In order to further analyze the performance of the systems we reran the same experiment but in this case the systems were iteratively given increased resources (this was achieved by varying the 'nodes' parameter in CProgol and the 'limit' parameter in Subdue) so that we could determine the number of hypotheses each system explored before it learned the concept (a cutoff accuracy of 80% was decided). Figure 7.14. (b) shows the number of hypotheses explored by each system to achieve an accuracy of 80% (this experiment was only performed for concept sizes varying from 5 to 18 as a significantly large amount of time was required). A snapshot of the experiment for concept size 10 is shown in Figure 7.14. (c). The results show that CProgol explores a larger number of hypotheses than Subdue.

An analysis of CProgol indicates that it first generates a most-specific clause from a randomly selected example using the mode definitions. Mode definitions

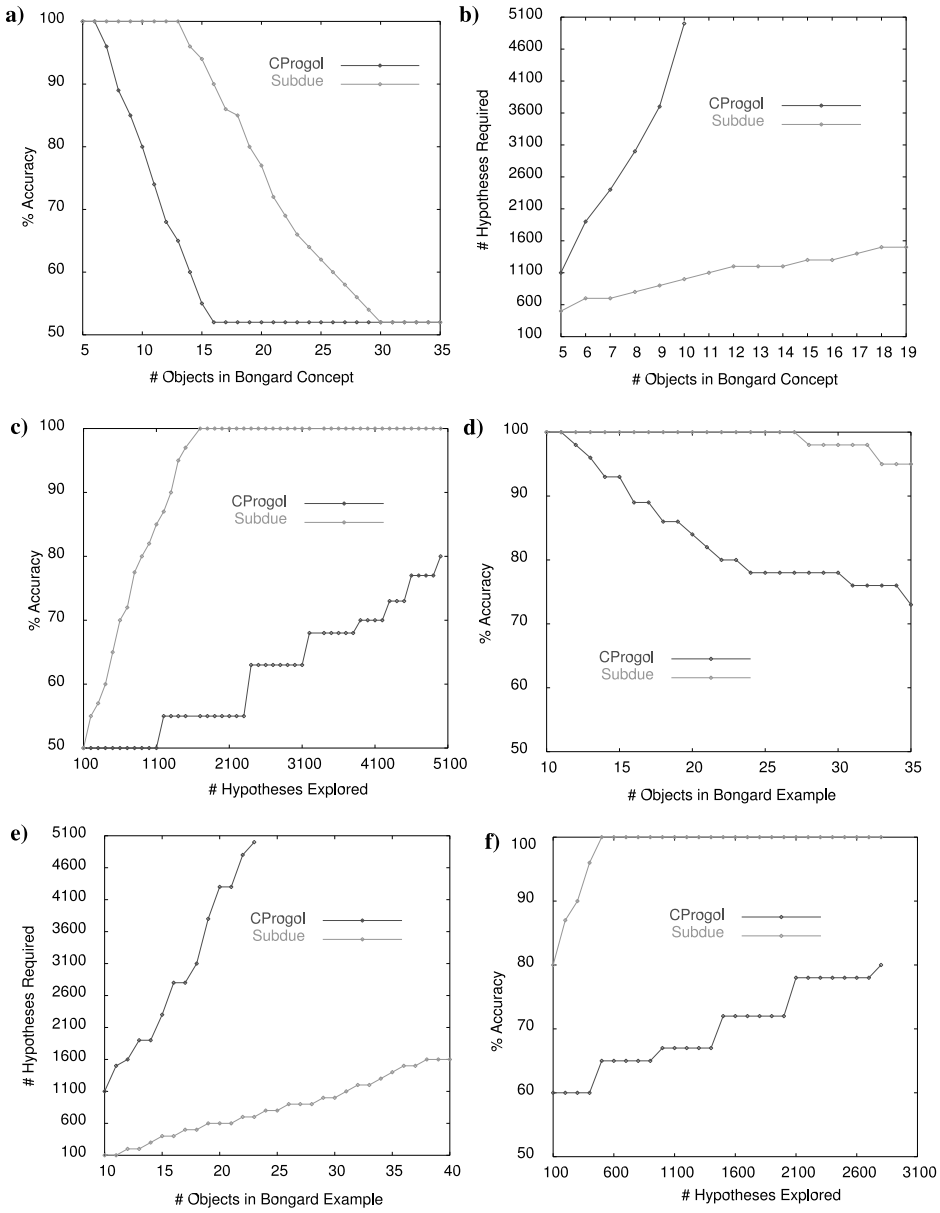

Figure 7.14. Results for structurally large Bongard problems.

together with the background knowledge form a user-defined model for generation of candidate hypotheses. After generation of the most-specific clause, CProgol performs a general-to-specific search in the bounded-subsumption lattice guided by the mode definitions. The most general hypothesis is the empty clause and the most specific hypothesis is the clause generated in the previous step. The process of hypothesis generation is affected more by the mode definitions and the background knowledge than the examples, firstly because a single example is used to construct the most specific clause, and secondly because the mode definitions have a major effect on the process of hypothesis generation. Thus, CProgol makes more use of the mode definitions and background knowledge and less use of the examples.

This observation about CProgol can be partially generalized to other logic-based approaches like top-down search of refinement graphs [24], inverse resolution[21] and relative least general generalization [22]. An analysis of Subdue indicates that hypotheses are generated only on the basis of the examples. The candidate hypotheses are generated by extending the subgraph by an edge and a vertex or just an edge in all possible ways as in the examples. As Subdue generates the hypotheses only on the basis of the examples, it is more example driven. This observation about Subdue can be partially generalized to other graph-based systems like FSG, AGM,and gSpan, because there is more use of the examples and less use of the model. Graph-based approaches tend to explore the hypothesis space more efficiently because they use only the examples to generate candidate hypotheses, and thus can search a larger portion of the smaller hypothesis space with a given amount of resources, which is essential in learning structurally large relational concepts.

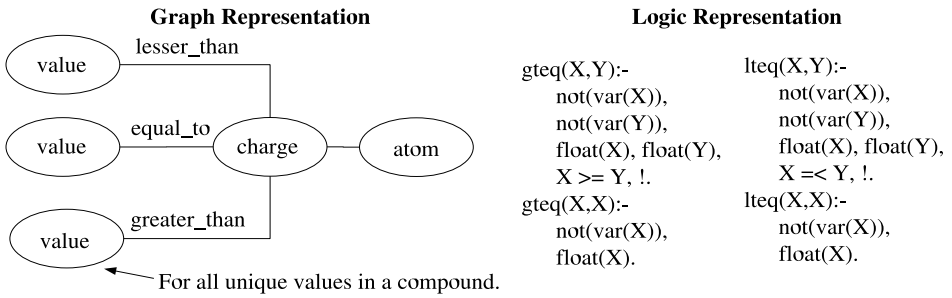

Figure 7.15. Representation for learning semantically complex concepts.

7.5.4 Experiments Using Semantically Complex Concepts from the Mutagenesis Dataset

In order to compare the performance of the approaches for learning semantically complex concepts, we ran Subdue and CProgol on the Mutagenesis dataset. Each system was provided with background knowledge so that numerical ranges could be learned. For CProgol this was achieved by introducing Prolog based background knowledge. For Subdue this was achieved by explicitly instantiating the background knowledge, i.e., additional structure was added to the training examples. This is shown in Figure 7.15..

The results of this experiment are shown in Table 7.5.4. The results indicate that CProgol uses the background knowledge and shows an improved performance while Subdue has achieved a lower accuracy than what it achieved without the background knowledge. These results give a strong indication that a logic-based approach performs better than a graph-based approach when learning semantically complex concepts.

In general, graph-based mining algorithms explore only those hypotheses which are explicitly present in the examples. For hypotheses to be explicitly present in the examples, it is essential that the targeted semantically complex concepts be explicitly instantiated in the examples. The drawbacks of the data-driven approach

CProgol training set accuracy	82.00%
Subdue training set accuracy	80.00%
CProgol training set runtime (seconds)	960
Subdue training set runtime (seconds)	848
CProgol 10-fold CV accuracy	78.91%
Subdue 10-fold CV accuracy	77.39%
CProgol 10-fold CV runtime (average, seconds)	810
Subdue 10-fold CV runtime (average, seconds)	878
CProgol - Subdue, Δerror$+/-\delta$	1.52% +/− 11.54%
CProgol - Subdue, confidence	31.38%

Table 7.7. Results on mutagenesis semantically complex concepts.

are that explicit instantiation is cumbersome in most cases and also that explicit instantiation is not a generalized methodology to learn complex semantic concepts. For example, suppose a domain expert were to suggest that the ratio of the number of carbon atoms to the number of hydrogen atoms in a molecule has an effect on the mutagenicity. CProgol with some added background knowledge could use this information to classify the molecules. Subdue, on the other hand, would require making changes to the representation such that the pattern would be found in terms of a graph. Logic-based approaches allow the exploration of hypotheses through implicitly defined background knowledge rather than explicit instantiation in the examples. This is essential in learning semantically complex multi-relational concepts.

7.6 CONCLUSIONS

The comparisons that were performed in this project highlight features of alternative approaches to graph mining and point to directions for continued research. We have observed that Subdue is preferred over FSG or gSpan when data is presented in one large graph or when a pattern is dominantly present in small or medium-size datasets. However, for large databases and those which exhibit a high degree of randomness, FSG or gSpan will likely be better choices because Subdue may not find the best pattern. Subdue can discovery concepts which are less frequent but potentially of greater interest. However, Subdue needs to make use of techniques such as canonical labeling and approximate algorithms for graph isomorphism in order to scale to larger datasets.

In addition, we can conclude that graph-based mining algorithms tend to explore the concept hypothesis space more efficiently than logic-based algorithms, which is essential for mining structurally large concepts from databases. However, logic-based systems make more efficient use of background knowledge and are better

at learning semantically complex concepts. These experiments point out the need for graph mining algorithms to effectively use background knowledge. Additional features such as generalizing numeric ranges from graph data will further improve these algorithms.

In addition to these comparisons, we are currently in the process of comparing Subdue to graph kernel learning algorithms. By performing these comparisons we hope to define a unifying methodology for mining graph-based data.

REFERENCES

1. M Bongard. *Pattern Recognition*. Spartan Books, 1970.

2. S Chakrabarti, M van den Berg, and B Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *Proceedings of the International World Wide Web Conference*, 1999.

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 Artificial Intelligence Research*, 1:231–255, 1994.

5. Luc de Raedt and W V Laer. Inductive constraint logic. In *Proceeedings of the Workshop on Algorithmic Learning Theory*, pages 80–94, 1995.

6. L Dehaspe and H Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

7. P Domingos and M Richardson. Mining the network value of customers. In *Proceedings of the international conference on Knowledge discovery and data mining*, page 5766, 2001.

8. G W Flake, S Lawrence, C L Giles, and F Coetzee. Self-organization of the web and identification of communities. *IEEE Computer*, 35(3):66–71, 2000.

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

10. Istvan Jonyer, Lawrence B Holder, and Diane J Cook. Hierarchical conceptual structural clustering. *International Journal on Artificial Intelligence Tools*, 10(1–2):107–136, 2001.

11. R I Kondor and J D Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the International Conference on Machine Learning*, pages 315–322, 2002.

12. M Kuramochi and G Karypis. Discovering frequent geometric subgraphs. In *Proceedings of the International Conference on Data Mining*, pages 258–265, 2002.

13. M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. In *Proceedings of the SIAM Data Mining Conference*, 2003.

14. M Kuramochi and G Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1038–1051, 2004.

15. W Lee and S Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the Seventh USENIX Security Symposium*, 1998.

16. Nitish Manocha, Diane J Cook, and Lawrence B Holder. Structural web search using a graph-based discovery system. *Intelligence Magazine*, 12(1), 2001.

17. T Matsuda, H Motoda, T Yoshida, and T Washio. Mining patterns from structured data by beam-wise graph-based induction. In *Proceedings of the Fifth International Conference on Discovery Science*, pages 422–429, 2002.

18. S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

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

20. S. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.

21. S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352, 1988.

22. S Muggleton and C Feng. Efficient induction of logic programs. In *Proceeedings of the Workshop on Algorithmic Learning Theory*, pages 368–381, 1990.

23. C. Noble and D. Cook. Graph-based anomaly detection. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2003.

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

25. Sriram Rajappa. Data mining in nonuniform distributed databases. Master's thesis, The University of Texas, 2003.

26. J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, 1989.

27. M F Schwartz and D C M Wood. Discovering shared interests using graph analysis. *Communications of the ACM*, 36(8):7889, 1993.

28. 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.

29. A Srinivasan, S Muggleton, M J E Sternberg, and R D King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

30. Shaobing Su, Diane J Cook, and Lawrence B Holder. Knowledge discovery in molecular biology: Identifying structural regularities in proteins. *Intelligent Data Analysis*, 3:413–436, 1999.

31. N. Vanetik, E. Gudes, and S. Shimony. Computing frequent graph patterns from semi-structured data. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2002.

32. Jason T L Wang, Mohammed J Zaki, Hannu T T Toivonen, and Dennis Shasha. *Data Mining in Bioinformatics*. Springer, 2004.

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

34. X Yan and J Han. CloseGraph: mining closed frequent graph patterns. In *Proceedings of the Conference on Knowledge Discovery and Data Mining*, pages 286–295, 2003.