Graph-Based Relational Concept Learning

Jesus A. Gonzalez JAGONZALEZ@INAOEP.MX
Instituto Nacional de Astrofísica, Optica y Electronica, (INAOE), Puebla, Mexico

Lawrence B. Holder HOLDER@CSE.UTA.EDU
Diane J. Cook COOK@CSE.UTA.EDU
Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019 USA

Abstract

We introduce a graph-based relational concept learning approach and its implementation in the SubdueCL system. We start by describing the Subdue discovery system that uses a graph representation and was extended to perform the concept learning task. Then, we present SubdueCL, our graph-based concept learner. Empirical results comparing SubdueCL with the Inductive Logic Programming systems FOIL and Progol show that SubdueCL is competitive with logic-based learners using structural data. Results in an artificial domain show that SubdueCL learns successfully with training example graphs of various size, density and noise.

1. Introduction

This research is intended to show the effectiveness of graph-based relational concept learning. To show this, we perform an empirical analysis where we test the SubdueCL system (our new graph-based relational concept learner) in two types of domains. The first is an artificial domain, where we evaluate SubdueCL's ability to learn known concepts in arbitrary graphs under various conditions of graph size, graph density, and noise. Second, we test the system with real and more complex domains to evaluate effectiveness in real-world domains. We also compare SubdueCL to the Inductive Logic Programming (ILP) concept learning systems FOIL (Cameron-Jones and Quinlan 1994) and Progol (Muggleton 1995). We perform this comparison, because logic-based systems have dominated the area of relational concept learning, especially ILP systems (Muggleton and Feng 1992). However, first-order logic can also be represented as a graph, and in fact, first-order logic is a subset of what can be represented using graphs (Sowa 1984). Therefore, learning systems using graphical representations have the potential to learn richer concepts if they can handle the increased size of the hypothesis space.

2. Related Work

In this section we present the work related to the area of learning using graphs. We start with a brief description of conceptual graphs in section 2.1, then we continue with a description of graph-based discovery in section 2.2.

2.1 Conceptual Graphs

Conceptual graphs are important for our empirical analysis. There is a standard translation from conceptual graphs into logic and vice versa. We want to use conceptual graphs for the comparison with ILP systems so that we do not introduce any bias while converting graphs into the logic representation used by ILP systems. In section 4 we present some experiments that indicate the utility of using conceptual graphs for the comparison with ILP systems.

Conceptual Graphs (CGs) are a logic-based knowledge representation derived from semantic networks (Lehmann 1992) and Peirce existential graphs (Sowa 1984). CGs are being used in different areas such as natural language processing, information retrieval and expert systems.

Formally, a conceptual graph (Sowa 1984) $G = (R, C, U, lab)$ is a bipartite, connected, and finite graph. R and C are the relation and concept vertices respectively. U is the set of edges, where the edges of a particular relation from R are totally ordered. The vertex labels are defined by the mapping lab where: if $c \in C$, then $lab(c) \in T_c \times M \cup \{*\}$. If $r \in R$, then $lab(r) \in T_r$.

Figure 1 shows an example of a conceptual graph expressing "a cat is on a mat." In figure 1, x:y denotes that y is a marker of type x.

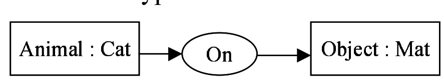

Figure 1. Conceptual Graph Example.

Concept vertices are used to represent entities, attributes, states and events. Relation vertices are used to interconnect concept vertices.

2.2 Subdue

Subdue (Cook and Holder 1994, 2000) is a relational learning system used to find substructures (subgraphs) that appear repetitively in the graph representation of databases. Subdue starts by looking for the substructure that best compresses the graph using the Minimum Description Length (MDL) principle (Rissanen 1989), which states that the best description of a data set is the one that minimizes the description length of the entire data set. In relation to Subdue, the best description of the data set is the one that minimizes:

$$\mathrm{I}(S) + \mathrm{I}(G|S)$$

where S is the substructure used to describe the input graph G, $\mathrm{I}(S)$ is the length (number of bits) required to encode S, and $\mathrm{I}(G|S)$ is the length of the encoding of graph G after being compressed using substructure S.

After finding the first substructure, Subdue compresses the graph and can iterate to repeat the same process. Subdue is able to perform an inexact match that allows the discovery of substructures whose instances have slight variations. Another important characteristic of Subdue is that it allows the use of background knowledge in the form of predefined substructures.

The model representation used by Subdue is a labeled graph. Objects are represented by vertices, while relations are represented by edges. Labels are used to describe the meaning of edges and vertices. When we work with relational databases, each row can be considered as an event. Events may also be linked to other events through edges. The event attributes are described by a set of vertices and edges, where the edges identify the specific attributes and the vertices specify the value of that attribute for the event.

Subdue uses a computationally-constrained beam search to find substructures. A substructure is a subgraph contained in the input graph. The algorithm starts with a single vertex as the initial substructure and at each iteration expands the instances of that substructure by adding an edge in every possible way, generating new substructures that might be considered for expansion. Section 3 provides more detail into the discovery algorithm and its use in graph-based concept learning.

3. Graph-Based Concept Learning

The main challenge in adding concept-learning capabilities to Subdue is the inclusion of "negative" examples into the process. Substructures that describe the positive examples, but not negative examples, are likely to represent the target concept. Therefore, the Subdue concept learner (which we will refer to as SubdueCL) accepts positive and negative examples in graph format.

Since SubdueCL is an extension to Subdue, it uses Subdue's core functions to perform graph operations, but the learning process is different. SubdueCL works as a supervised learner by differentiating positive and negative examples using a set-covering approach instead of graph compression. The hypothesis found by SubdueCL consists of a disjunction of substructures. SubdueCL forms one of these substructure disjuncts in each iteration. Positive example graphs that are described by the substructure found in a previous iteration are removed for subsequent iterations.

3.1 Substructure Evaluation

The way in which SubdueCL decides if a substructure will be part of the concept is different from Subdue. SubdueCL uses an evaluation formula to give a value to all the generated substructures. This formula assigns a value to a substructure according to how well it describes the positive examples (or a subset of the positive examples) without describing the negative examples. A substructure covers an example if the substructure matches a subgraph of the example. This graph match can be inexact, as controlled by a user-defined match threshold, and is constrained to run in time polynomial in the size of the graphs. Positive examples covered by the substructure increase the substructure value, while negative examples decrease its value. Positive examples that are not covered and the negative examples covered by the substructure are considered errors, because the ideal substructure would be one covering all the positive examples without covering any negative example. The substructure value is calculated as follows:

$$value = 1 - Error$$

where the error is calculated with respect to the positive and negative examples covered by the substructure using the following formula:

$$Error = \frac{\#PosEgsNotCovered + \#NegEgsCovered}{\#PosEgs + \#NegEgs} \qquad 3.1$$

#PosEgsNotCovered is the number of positive examples not covered by the substructure, and #NegEgsCovered is the number of negative examples covered by the substructure. #PosEgs is the number of positive examples remaining in the training set (the positive examples that have already been covered in a previous iteration were removed from the training set), and #NegEgs is the total number of negative examples, which does not change, because negative examples are not removed from the training set.

The only problem found with the use of formula 3.1 is that when two substructures have the same error, we would like to prefer the substructure covering more positive examples. For instance, suppose we have 10 positive examples and 10 negative examples. Substructure *S1* covers 5 positive examples and 0 negative examples, and substructure *S2* covers 10 positive examples and 5 negative examples. In this case both substructures have an error of 1/4 according to formula 3.1, but we prefer SubdueCL to select *S1*, because it does not cover any negative examples. In order to do this we need to assign a negative weight (say *k*) to the negative errors. After some algebraic manipulation we express the adjusted error with formula 3.2, where *k* is the negative weight. The default value for k is 3, but any value greater than or equal to 2 will effect the desired preference.

$$AdjustedError = \frac{\#PosEgsNotCovered + (k * \#NegEgsCovered - \#NegEgs * (1-k))}{\#PosEgs + \#NegEgs} \quad 3.2$$

Now the adjusted error of *S1* and *S2* according to formula 3.2 (and with the default value of k = 3) is 5/4 and 7/4 respectively. Using this formula, SubdueCL will prefer *S1* over *S2*.

3.2 SubdueCL Algorithm

The SubdueCL algorithm is shown in figures 2 and 3. The main function takes as parameters the positive examples G_p, the negative examples G_n, the *Beam* length (since SubdueCL's search algorithm is a beam search), and a *Limit* on the number of substructures to include in its search. The main function makes calls to the SubdueCL function in order to form the hypothesis *H* that describes the positive examples.

```
Main(Gp, Gn, Limit, Beam)
  H = {}
  repeat
    repeat
      BestSub = SubdueCL(Gp, Gn, Limit, Beam)
      if BestSub = {}
        then Beam = Beam * 1.1
    until ( BestSub ≠ {})
    Gp = Gp – { p ∈ Gp | BestSub covers p }
    H = H + BestSub
  until Gp = {}
  return H
end
```

Figure 2. SubdueCL's Main Algorithm.

A substructure is added to *H* after each call to the SubdueCL function. In the case that SubdueCL returns NULL, the *Beam* is increased by a 10% so that a larger search space can be explored during SubdueCL's search. We chose to increase the beam by 10%, because that value was enough to find a substructure in the next iteration in most experiments. Also, after SubdueCL finds a substructure, the positive examples covered by it are removed from the positive graph.

Figure 3 shows the SubdueCL function, which first builds a *ParentList* containing a substructure for each vertex in the graph with a different label, but keeping only as many substructures as the length of the *Beam*. The "*mod* Beam" qualifier means that the lists keep only as many substructures as the *Beam* size. Each of those substructures in the *ParentList* is then expanded by one edge or one vertex and an edge in all possible ways and evaluated according to equation 3.2 presented earlier. Those substructures that cover at least one positive example are kept in the *BestList*, which is limited to the *Beam* size. A *ChildList* keeps all the substructures that were obtained from the expansion of the substructures in the *ParentList* and is also limited by the *Beam* size.

The *Limit* parameter is used to constrain the number of expanded substructures, but if the *BestList* is empty after expanding *Limit* substructures from the *ParentList*, the limit is increased by 20% until one is found. We chose to increase this limit by 20%, because it was usually enough to find a positive substructure in the next trial when working with our experiments. Finally, the SubdueCL function returns the best of *BestList*. It is important to mention that all the lists are ordered according to the substructures' values.

```
SubdueCL(Gp, Gn, Limit, Beam)
  ParentList = (All substructures of one vertex in Gp) mod Beam
  repeat
    BestList = {}
    Exhausted = TRUE
    i = Limit
    while ( (i > 0 ) and (ParentList ≠ {}) )
      ChildList = {}
      foreach substucture S in ParentList
        foreach expanded substructure C in Expand(S)
          Evaluate(C, Gp, Gn)
          if CoversOnePos(C,Gp)
            then BestList = BestList ∪ {C}
          ChildList = ( ChildList ∪ C ) mod Beam
          i = i – 1
        endfor
      endfor
      ParentList = ChildList
    endwhile
    if BestList = {} and ParentList ≠ {}
      then Exhausted = FALSE
           Limit = Limit * 1.2
  until ( Exhausted = TRUE )
  return first(BestList)
end
```

Figure 3. SubdueCL Algorithm.

The version of SubdueCL just presented may return hypotheses inconsistent with the training examples. The only variation to produce a consistent version of the SubdueCL algorithm is to only consider substructures covering no negative examples in the *BestList*.

4. Experiments

In this section we present experimental results to evaluate SubdueCL's performance with different types of graphs. First, we use an artificial domain to see if SubdueCL learns known concepts. Then, we compare SubdueCL with ILP systems. In this comparison with ILP systems, we test SubdueCL with flat and relational domains.

4.1 Artificial Domain

The purpose of this experiment is to evaluate the effectiveness of SubdueCL in learning known relational concepts. An artificial domain is ideal for this, because we can control the experiment's environment. The basic idea of this experiment is that we embed a concept in a set of graphs and run SubdueCL to find that concept. In this experiment we used the inconsistent version of SubdueCL. We consider two types of graphs: normal graphs and conceptual graphs. *Normal graphs* consist of labeled vertices linked with labeled edges, where vertices represent concepts and edges represent relations among them. *Conceptual graphs*, as presented in section 2.1, are composed of labeled vertices of two types, concept vertices and relation vertices, that are linked with edges (unlabeled for the case of only binary relations).

As we mentioned in section 2.1, there is a standard translation of conceptual graphs into logic and vice versa. We can use this property when we compare SubdueCL with ILP systems; in this way we use the same procedure to translate our graphs into logic or the ILP rules to graphs while minimizing bias during the knowledge representation process. This is the motivation for showing that SubdueCL produces similar results using normal and conceptual graphs.

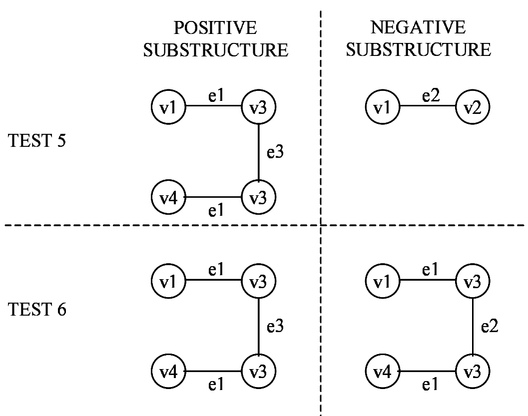

Figure 4. Positive and Negative Substructures Embedded in the Graphs for Tests 5 and 6.

In order to make this evaluation with SubdueCL, some experiments were conducted using an artificial domain. The artificial domain consists of a set of graphs. Some of those graphs represent positive examples and others negative examples. Positive example graphs are distinguished from the negative examples, because a known substructure is embedded into them, but not into the negative examples.

We performed a total of 13 tests varying the similarity between the positive and negative substructures and the graph's sizes. As an example of the graphs used for this experiment, figure 4 shows the positive and negative substructures that were embedded in the positive and negative examples of tests 5 and 6. Figure 5 shows the substructure that SubdueCL learned from those examples using normal and conceptual graphs.

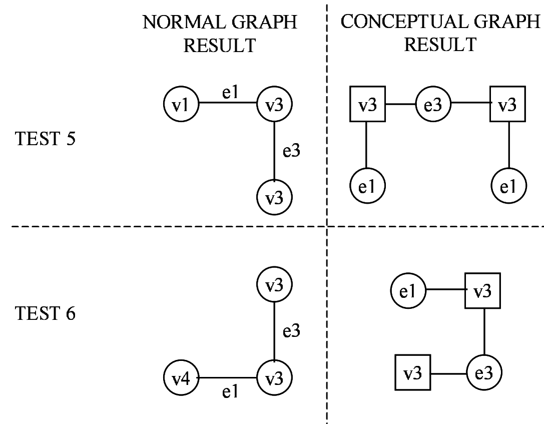

Figure 5. Result for Tests 5 and 6 with Normal and Conceptual Graphs.

From the results of this experiment, we could see that SubdueCL finds the positive embedded substructure (or a subgraph of it) and produces similar results using both normal and conceptual graphs (as we can see in figure 5 for test number 6). The conceptual graph shown in the result for test number 6 corresponds to a subgraph of the result for normal graphs. For test number 6, the only difference between the graphs is that the conceptual graph result does not contain the vertex "v4". For the result of test number 5, the conceptual graph is not a subgraph of the normal graph but they are very similar.

From the similarity of the results using conceptual and normal graphs for this experiment, we can conclude that SubdueCL discovers almost the same substructures. The difference between the results with normal and conceptual graphs in most of the cases is that the conceptual graph result is a subgraph of the normal graph result. This means that we can use a conceptual graphs representation with SubdueCL for the comparison with ILP systems.

From other experiments conducted in the artificial domain we found that SubdueCL produces more substructures to represent a concept as the graph density (number of edges divided by the number of vertices) increases (which also introduces noise). We also found from a 10-fold cross validation that SubdueCL obtains better accuracy using normal graphs than conceptual graphs. This happens because the search space increases (in the number of vertices) when we convert normal graphs to conceptual graphs. Finally we studied the learning curve produced by SubdueCL and found that it is able to learn a domain with a low number of training examples if the positive and negative examples are very different (Gonzalez 2001).

4.2 Comparison of SubdueCL with ILP Systems

Logic-based systems have dominated the area of relational concept learning, especially Inductive Logic Programming (ILP) systems (Muggleton and Feng 1992).

However, first-order logic can also be represented as a graph, and in fact, first-order logic is a subset of what can be represented using graphs (Sowa 1984). Therefore, learning systems using graphical representations have the potential to learn richer concepts if they can handle the increased size of the hypothesis space. In this section we compare SubdueCL to two ILP systems: FOIL (Cameron-Jones and Quinlan 1994) and Progol (Muggleton 1995). Quantitative comparisons indicate that the graph-based learner is competitive with the logic-based learners. We conducted experiments in structural (relational) and non-structural domains in order to capture the advantages and disadvantages of SubdueCL in these types of domains. For these experiments we used the conceptual graph representation (as presented in section 2.1).

4.2.1 Non-Relational Domains

Five non-relational domains were used to compare FOIL, Progol and SubdueCL: golf, vote, diabetes, credit and tic-tac-toe. The golf domain is a trivial domain used to demonstrate machine learning concepts, typically decision-tree induction (Quinlan 1986). The domain consists of fourteen examples (10 positive or play golf, 4 negative) and four attributes, two discrete and two continuous, with no missing values. The vote domain is the Congressional Voting Records Database available from the UCI machine learning repository (Keogh, Blake and Merz 1998). The diabetes domain is the Pima Indians Diabetes Database, and the credit domain is the German Credit Dataset from the Statlog Project Databases, also available from the UCI repository. For the Tic-Tac-Toe domain, 958 examples were exhaustively generated. Positive examples are those board configurations where "X" wins the game, and negative examples are those board configurations where "X" loses or the game is tied.

The accuracy values shown in Table 1 are the results of averaging the individual accuracies found using a 10-fold cross validation. The only domain where a 3-fold cross validation was used is the golf domain, because it consisted of only fourteen examples. In the case of the diabetes domain FOIL produced the best result with an accuracy of 70.66%, then SubdueCL with an accuracy of 65.79% and finally Progol with 63.68% accuracy. The low accuracy values are due to the continuous values of the attributes of the domain. SubdueCL does not yet learn ranges of values. From these results we can see that FOIL can better deal with continuous values. For the other domains, where we have either only discrete attributes or a combination of discrete and continuous values (Golf, Vote, Credit, and Tic-Tac-Toe domains), SubdueCL is competitive (better in some cases) in terms of accuracy with FOIL and Progol. A disadvantage of SubdueCL is that it does not have an effective way to deal with continuous values; that is, it cannot represent or output a substructure that refers to a range of values. This effect is most pronounced in the Diabetes domain, where most of the fields are continuous, and in the credit domain that consists of a combination of discrete and continuous values.

These results show that for non-relational domains SubdueCL is competitive and even better than ILP systems if the domain does not contain continuous values. The differences between the accuracies of the systems in the diabetes domain are significant. In the case of the credit domain, the differences between the accuracies of SubdueCL versus FOIL and FOIL versus Progol are significant. In order to be more competitive with ILP systems in this type of domain, we need to add the capability to deal with ranges of numbers.

Table 1. Percent Accuracy Results on Non-Relational Domains: Golf, Vote, Diabetes, Credit, and Tic-Tac-Toe

	Golf	Vote	Diabetes	Credit	T-T-T
FOIL	66.67	93.02	70.66	68.60	100.0
Progol	66.67	94.19	63.68	63.20	100.0
Subdue	66.67	94.65	65.79	63.50	100.0

4.2.2 Relational Domains

The three relational domains used in this study are illegal chess endgames, carcinogenesis and website hyperlink structure. Representations and results for the three domains are discussed in the following sections.

Chess Domain

The chess domain consists of 20,000 examples of row-column positions for a white king, white rook and black king such that the black king is in check (positive) or not (negative). Therefore, if white's turn is next, then the positive examples are illegal configurations. The relational information in this domain consists of adjacency relations between chess-board positions and less-than or equal relations between row and column numbers (0-7). Both FOIL and Progol extensionally define the three relations.

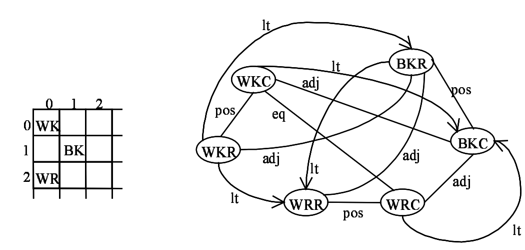

Figure 7. An Example from the Chess Domain. (a) Board configuration and (b) SubdueCL's graphical representation of the example.

Figure 7b shows the graph representation used for the chess domain example in figure 7a. Each piece is represented by two vertices corresponding to the row and column of the piece, connected by a position relation (e.g., WKC stands for white king column).

Due to computational constraints only a subset (5,000 examples) of the entire database was used for the 10-fold cross validation. The accuracy results are 99.74% for Progol, 99.34% for FOIL, and 99.74% for SubdueCL. The difference in error between SubdueCL and FOIL (which is the same as that between Progol and FOIL) is of 0.4% (+/- 0.7242%) with a confidence of 94.27%, and we are 89.27% certain (by ANOVA results) that the confidence value of the difference is correct. In terms of number of rules, Progol learned 6 rules, FOIL learned 11 rules, and SubdueCL learned 7 rules (substructures). The three systems perform comparably in this domain.

Carcinogenesis Domain

The PTC (Predictive Toxicology Challenge) carcinogenesis databases contain information about chemical compounds and the results of laboratory tests made on rodents in order to determine if the chemical induces cancer to them. The data in this database comes from the predictive toxicology evaluation project conducted by the National Institute of Environmental Health Sciences (NIEHS). We applied SubdueCL to the first two databases PTC1 and PTC2 but the most recent result comes from "The Predictive Toxicology Challenge 2000 – 2001" or PTC3 (see http://www.informatik.uni-freiburg.de/~ml/ptc/ for more information about the challenge).

Each example is composed of all the information related to the compound: the atoms that it contains, the element, type and charge of each atom, the bonds between atoms, the groups that are formed by the atoms and counters of groups of the same type (i.e., number of amine groups).

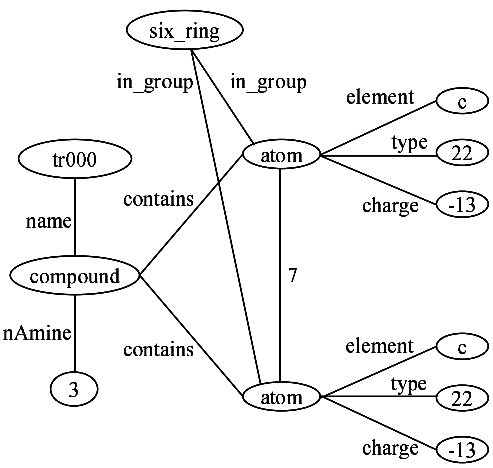

Figure 8. Graph representation of a chemical compound from the Cancer Domain.

The graph representation used for the cancer domain is shown in figure 8 (actual examples are much larger than this example). In the figure we can see that compounds are linked to their atoms with edges labeled "contains." Each atom vertex is connected to three vertices that describe the atom's element, type, and charge, linked with the edges "element," "type," and "charge" respectively. Two atoms can be connected with a bond edge. Bond edges are labeled with the type of bond, which is an integer number (1 = Single, 2 = Double, 3 = Triple, 4 = Aromatic, 5 = Single or Double, 6 = Single or Aromatic, 7 = Double or Aromatic, and 8 = Any). Groups are represented with vertices and connected to the vertices of all the atoms participating in the group with edges labeled as "in_group." We also included counters of the number of groups of the same type contained by the compound. Counters were represented with a vertex labeled with the number of groups (of the same type, e.g., amine) and linked with an edge with the name of the group preceded by an "n" (e.g., nAmine) to the compound. We did not include the short-term assays or predictive tests as used in the previous experiments, because they were not available for the new dataset.

We created a program to read all the information related to the compounds from an input file and generate an output graph file. The output graph file consists of positive and negative examples identified by their category (male rats MR, female rats FR, male mice MM, or female mice FM).

We created four training files and four testing files (i.e., a training file for male rats with its corresponding testing file). The results of the challenge were released in August 2001, and SubdueCL achieved maximum performance for the male rats category among other challenge submissions for the same category. This is an encouraging result that shows how SubdueCL is successful in this type of structural domain. Figure 9 shows one of the substructures found by SubdueCL in this domain (Gonzalez, Holder and Cook 2001). As we can see from the results, SubdueCL was able to find substructures of chemical compounds that are correlated with their tendency to cause cancer. These substructures are now under evaluation by toxicological experts to determine their relevancy.

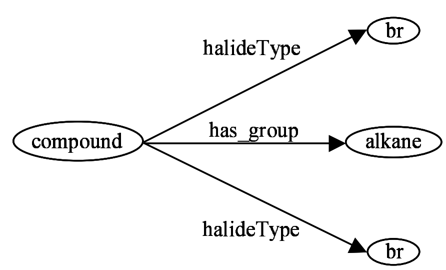

Figure 9. A compound that contains two "halide" groups of type "br" and that also has an "alkane" group may cause cancer.

We also performed a 10-fold cross validation with the training data with the following results: 64% accuracy for male rats, 72.14% accuracy for female rats, 68.8% accuracy for male mice, and 65.47% accuracy for female mice. The accuracy results are low due to the difficulty of the task.

Web Domain

The Web domain consists of graphs created from web sites. At the moment we have three options for the information that these graphs contain:

- Hyperlink structure
- Hyperlink structure + Page's title
- Hyperlink structure + Page's content

Figure 10 shows an example of a graph with the hyperlink structure option. Figure 11 shows an example of a graph created with the hyperlink structure plus the page's content.

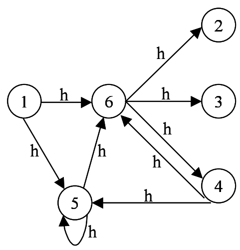

Figure 10. A Graph form the Web Domain created with the Hyperlink structure option

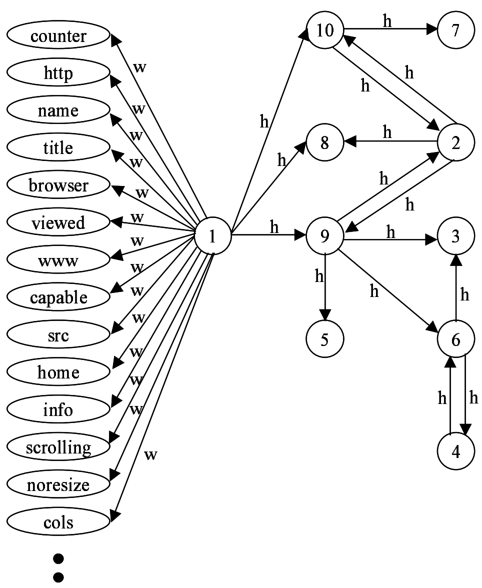

Figure 11. A Graph form the Web Domain created with the Hyperlink Structure + Page's Content Option

We are using a Perl program to extract this information and convert it into its graph representation. The program extracts the pages' hyperlink structure and converts it to a graph. In the case of the structure option, vertices have the number of the page and edges are labeled as "hyperlink" or "h" as in figure 10. If the page's content or title is considered, each word is represented as a vertex and is linked to the page number to which it belongs with an edge labeled as "word" or "w" as in figure 11.

The first experiment that we made for the Web domain consisted in differentiating the Web pages of professors and students. We considered the web sites of the professors and students of the Computer Science Department of the University of Texas at Arlington. We created graphs for all the professors and students and selected those graphs having at least five vertices. We made the professors' web pages our positive examples and the students' web pages our negative examples. For this initial experiment we chose the web page structure + page's content option, because we wanted to give as much information as possible to SubdueCL and observe its behavior. SubdueCL was able to find a very small substructure that could differentiate the positive examples from the negative examples. This substructure is shown in figure 12 and says that every professor has in their web page the word "box", but students do not have this word in their web pages. This is true because the word "box" is part of the address field of the professors' web pages.

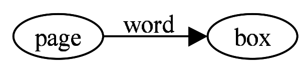

Figure 12. A Substructure Found in the Web Domain (Professors-Students).

We also performed a 3-fold cross validation for this experiment. We used 24 positive examples (professors web pages) and 23 negative examples (students web pages). In this test SubdueCL achieved 84.5% accuracy. This means that the predictive accuracy of a hypothesis produced by SubdueCL in the web domain is not very high (only 84.5%) when using the hyperlink structure + page's content. We performed the same 3-fold cross validation experiment with Progol obtaining an accuracy of 63.64% which is even lower than SubdueCL's accuracy.

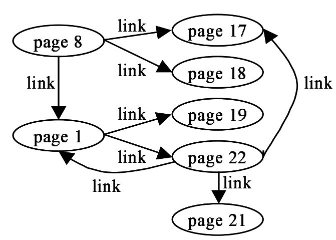

Figure 13. A Substructure Found in the Web Domain (Computer Stores - Professors).

For the second experiment in the web domain we used the hyperlink structure only. We chose this option, because we wanted to learn only structure without considering the web pages' content so that SubdueCL did not learn a substructure with a word (or set of words) describing the domain as in the previous experiment. We chose the structure graphs of web sites of computer stores as our positive examples and the structure graphs of web sites of professors as our negative examples. We searched the web for computer stores and created graphs for their web sites. As in the previous experiment, we chose only those graphs having at least five vertices. Figure 13 shows a substructure found in this domain. This substructure covered 24 of a total of 29 positive examples without covering any of the negative examples. Although not all the examples covered by this substructure had the same interpretation, for most of them the following explanation applies: Node 8 represents a category of products linked to three subcategories of products represented by nodes 1, 17, and 18. Nodes 19 and 22 represent either more specific subcategories derived from the subcategory represented by node 1 or specific products. Node 21 represents a specialization of the subcategory represented by node 22.

We also performed a 3-fold cross validation for this test. We used 31 positive examples (computer stores' pages) and 25 negative examples (professors' pages). SubdueCL achieved a predictive accuracy of 75.93% using the hyperlink structure method. This means that the predictive accuracy of SubdueCL using hyperlink structure is even lower than the predictive accuracy achieved using the hyperlink structure + pages' content option. We performed the same experiment with Progol and achieved an accuracy of 78.43% which is higher than the accuracy obtained with SubdueCL. This means that Progol performs slightly better for the experiments containing only hyperlink structure.

The results in the web domain tell us that SubdueCL is able to learn how to differentiate between the graph structure of different classes of web sites and is still competitive with Progol on some aspects of this task. Note that in the case that one class subsumes the other (as is the case with graphs of the structure of professors' web sites as the positive examples and graphs of the structure of computer stores web sites as the negative examples), we need to make the super class to be the positive examples and the subclass to be the negative examples.

5. Conclusion

In this research we introduced our graph-based concept learning approach and its implementation in the SubdueCL system. The experiments made with an artificial domain show that SubdueCL is able to successfully learn a concept from examples. In this empirical analysis we also compared SubdueCL to the ILP systems FOIL and Progol in relational and non-relational domains. The results of this comparison for non-relational domains show that SubdueCL either outperforms or obtains the same performance as FOIL and Progol when the domain does not contain only continuous values. For the case of relational domains SubdueCL outperformed FOIL and obtained the same accuracy as Progol. This shows that SubdueCL is competitive with these ILP systems, but has the advantage of having a more natural way to represent these relational domains. These results show that SubdueCL successfully learns in relational domains. In our future work we need to test SubdueCL with more relational databases like in the 2001 Knowledge Discovery in Databases (KDD) Cup that focuses on data from genomics and drug design.

There are some enhancements that we need to make to SubdueCL to make it more competitive with ILP systems. First, we need to give SubdueCL the ability to express ranges of values. This would be very useful for domains that involve continuous variables. In this type of domain SubdueCL could find substructures containing vertices whose value expresses a range of values that the continuous variable can take. Second, we need to allow SubdueCL to express that the label of one vertex is the same as another vertex, and also that the numeric label of one vertex is less than or greater than the numeric label of another vertex. Third, we need to find a representation able to describe recursive structures. This will be part of our future work.

References

Cameron-Jones, R. M., and J. R. Quinlan. (1994). Efficient Top-down Induction of Logic Programs. *SIGART Bulletin*, 5, 1:33-42.

Cook, D. J., and L. B. Holder. (1994). Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research*. 1:231-55.

Cook, D. J., and L. B. Holder. (2000). Graph-Based Data Mining. *IEEE Intelligent Systems*, 15(2):32-41.

Gonzalez, Jesus A. (2001). *Empirical and Theoretical Análisis of Relational Concept Learning Using a Graph-Based Representation*. Doctoral dissertation, Department of Computer Science, University of Texas at Arlington.

Gonzalez, Jesus A., L. B. Holder, and Diane J. Cook. (2001). Application of Graph-Based Concept Learning to the Predictive Toxicology Domain. *Proceedings of the Predictive Toxicology Challenge Workshop*.

Keogh, E., C. Blake, and C. J. Merz. (1998). *In UCI Repository of Machine Learning Databases*.

Lehmann, F. (1992). *Semantics Networks in Artificial Intelligence*. Oxford: Pergamon Press.

Muggleton S. (1995). Inverse Entailment and Progol. *New Generation Computing*, 13:245-86.

Muggleton, S. and C. Feng. (1992). Efficient Induction of Logic Programs. *Inductive Logic Programming*, Academic Press 281-97.

Rissanen, J. (1989). *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company.

Sowa, J. F. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley.