Graph-Based Concept Learning

Jesus A. Gonzalez, Lawrence B. Holder, and Diane J. Cook

Department of Computer Science and Engineering
University of Texas at Arlington
Box 19015, Arlington, TX 76019-0015
{gonzalez,holder,cook@cse.uta.edu}

Abstract

We introduce the graph-based relational concept learner SubdueCL. We start with a brief description of other graph-based learning systems: the Galois lattice, Conceptual Graphs, and the Subdue system. We then present our new system SubdueCL and finally we show some preliminary results of a comparison of SubdueCL with the two Inductive Logic Programming (ILP) systems Foil and Progol.

Introduction

We describe our current research on graph-based concept learning based in the SubdueCL system. Graph-based systems have the potential to be competitive in the learning task, because they provide a powerful and flexible representation that can be used for relational domains. The main competitors of graph-based systems are logic based systems, especially Inductive Logic Programming (ILP) systems, which have dominated the area of relational concept learning. We are comparing our graph-based approach with the ILP systems Foil and Progol. On the theoretical side, we have studied other graph-based systems, and we are applying the related theory to our system. For example, we are working in a PAC learning analysis of the SubdueCL system in order to show that it is possible to learn using graph-based systems with a polynomial number of training examples.

The paper is organized as follows. The related work section briefly describes the graph-based systems that we have studied: Conceptual Graphs, the Galois lattice and the Subdue system. The SubdueCL section describes our graph-based concept learning system. The empirical results section presents some preliminary results from a comparison of SubdueCL with the two ILP systems Foil and Progol. The last section presents our conclusions and future work.

Related Work

In this section we briefly describe the role of ILP systems as concept learners and then we present some work related to graph-based concept learners.

ILP Systems

One approach to relational concept learning systems 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 structural domains: Progol in the Chemical Carcinogenicity domain (Srinivasan, King, Muggleton et al. 1997) and FOIL (Cameron and Quinlan 1994) for learning patterns in Hypertext domains (Slattery & Craven 1998). The system presented in this paper uses graphs as its data representation, which are flexible and descriptive. Graphs can also describe FOPC using Conceptual Graphs as introduced by John Sowa (Sowa 1992).

Conceptual Graphs

Conceptual Graphs (CGs) are a logic-based knowledge representation derived from Semantic Networks and Peirce Existential Graphs (Sowa 1992). Figure 1 shows an example of a Conceptual Graph expressing "A cat is on a mat". Square vertices are used to represent concepts and oval vertices represent relations. Edges are used to link concepts with relations.

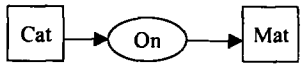

Figure 1: A Conceptual Graph's Example

CGs are being used in different areas of Artificial Intelligence like natural language processing, information retrieval and expert systems. Conceptual Graphs provide a powerful and rich knowledge representation that has been used for concept learning as presented in (Jappy and Nock 1998). Their work describes a PAC Learning (Valiant 1985) analysis using Conceptual Graphs to show its effectiveness for Learning. With this model, the authors were able to

prove that it is possible to PAC learn Conceptual Graphs for a restricted class that consists of Conceptual Graphs with at most n vertices. From that result they stated the theorem:

"Let $C_{(\alpha_i)} \in CG$ denote a conceptual graph class. If both $|C_{(\alpha_i)}|$ and the complexity of the projection test between elements of $C_{(\alpha_i)}$ are polynomial in (α_i), then $C_{(\alpha_i)}$ is PAC learnable".

Where α_i (α_1,α_2, and α_3) are the richness parameters that define the search space of the class of conceptual graphs and correspond to the number of relations, concepts and labels that are available to form valid conceptual graphs. The projection test corresponds to the test used to verify if an example belongs to a class or not (a form of graph morphism).

Galois Lattice

Another technique used to accomplish the task of concept learning using a graph representation is drawn from the framework of a Galois Lattice (Liquiere and Sallantin 1998). In the Galois Lattice framework for learning, each node in the lattice consists of a description graph and the set of examples described by the graph. The lattice construction starts with an empty lattice. In the first step, the description of all the examples is added to the lattice. For the rest of the levels, new concepts are created from each pair of concepts that have already been found in the previous step. The operation used to create a concept from two other concepts is called generalization "$\wedge$" and given two description graphs, produces the largest description graph that is contained in both original graphs. After a new concept is created, the examples that it describes are associated with it (the union of the examples associated to its parent graphs).

The search space for the Galois Lattice algorithm consists of all the possible generalizations of the set of training examples. In this case, the learning process goes from specific to general. The complexity of the Galois Lattice creation algorithm is $O(n^3p)$ (Liquiere and Sallantin 1998), where n is the number of examples and p is the size of the lattice. Liquiere and Sallantin mentioned that they had an experimental implementation called GRAAL, but that it could not be used in practical cases yet and that it was only an algorithm used for formal analysis. The Galois Lattice method is restricted to be used only with the subclass of Locally Injective Graphs (LIGs), because it was proved that the generalization operation runs in polynomial time for these types of graphs. If a different method to create the Galois Lattice (that does not use this generalization operation) is used, the restricted classes (in this case LIG's) might be different. In the future work of the Galois Lattice research, the authors mentioned that they would work to prove that LIG is PAC learnable. They also hope to make GRAAL a tool to work with practical applications.

Subdue

Subdue (Cook, Holder 1994) is a Data Mining tool that achieves the task of clustering using an algorithm categorized as an example-based and relational learning method. It is a general tool that can be applied to any domain that can be represented as a graph. Subdue has been successfully used on several domains like CAD circuit analysis, chemical compound analysis, and scene analysis (Cook, Holder and Djoko 1996, Cook, Holder and Djoko 1995, Cook and Holder 1994, and Djoko, Cook and Holder 1995).

Subdue uses a model evaluation method called "Minimum Encoding" that is a technique derived from the minimum description length principle (Rissanen 1989) and chooses as best substructures those that minimize the description length metric that is the length in number of bits of the graph representation. The number of bits is calculated based on the size of the adjacency matrix representation of the graph. According to this, the best substructure is the one that minimizes I(S) + I(G|S), where I(S) is the number of bits required to describe substructure S, and I(G|S) is the number of bits required to describe graph G after being compressed by substructure S.

The main discovery algorithm is a computationally constrained beam search. The algorithm begins with the substructure matching a single vertex in the graph. Each iteration the algorithm selects the best substructure and incrementally expands the instances of the substructure. The algorithm searches for the best substructure until all possible substructures have been considered or the total amount of computation exceeds a given limit. Evaluation of each substructure is determined by how well the substructure compresses the input graph according to the heuristic being used. The best substructure found by Subdue can be used to compress the input graph, which can then be input to another iteration of Subdue. After several iterations, Subdue builds a hierarchical description of the input data where later substructures are defined in terms of substructures discovered on previous iterations.

Figure 2 shows a simple example of Subdue's operation. Subdue finds four instances of the triangle-on-square substructure in the geometric figure. The graph representation used to describe the substructure, as well as the input graph, is shown in the middle.

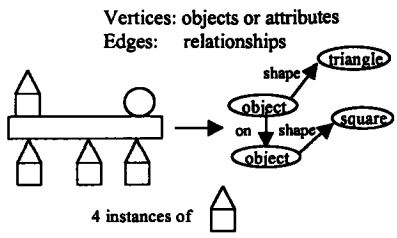

Figure 1: SUBDUE'S Example.

SubdueCL

As we mentioned before, 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 1992). Therefore, learning systems using graphical representations have the potential to learn richer concepts if they can handle the increased size of the hypothesis space. The Subdue Concept Learner (which we will refer to as SubdueCL) is an extension to the Subdue system described before. The main challenge in adding concept-learning capabilities to Subdue was 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 accepts both 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 set of disjunctions of conjunctions (substructures), i.e., the concept may contain several rules. SubdueCL forms one of these conjunctions (rules) in each iteration. Positive example graphs that are described by the substructure found in a previous iteration are removed from the graph for subsequent iterations.

The way in which SubdueCL decides if the substructures (or rules) will be part of the concept or not is also 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. Then, positive examples covered by the substructure increase the substructure value while negative examples decrease its value. In this formula the 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:

$$value = 1 - Error$$

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

Using this formula, SubdueCL chooses rules that maximize the substructure's value and in this way it minimizes the number of errors made by the substructures used to form the concept.

We have two versions of SubdueCL. The first is an *inconsistent* learner, which means that the substructures of the hypothesis are allowed to cover some negative examples. The second version is a *consistent* learner, in which the substructures that form the hypothesis are not allowed to cover any negative example.

SubdueCL's Algorithm

The SubdueCL algorithm is shown in figure 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. A substructure is added to H each time that the SubdueCL function is called. In case that the SubdueCL returns NULL, the Beam is increased so that a larger search space can be explored during SubdueCL's search. Also, after SubdueCL finds a substructure, the positive examples covered by it are removed from the positive graph.

```
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 3: SubdueCL's Main Function

Figure 4 shows the SubdueCL function, which starts building a ParentList creating 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 parent list is then expanded by one vertex or one vertex and an edge in all possible ways and evaluated according to the formula presented before. Those substructures that cover at least one positive example are kept in the BestList but 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 expand as many substructures as the Limit, but if the BestList is empty after expanding Limit substructures from the ParentList, the limit is increased until one is found. The SubdueCL function returns the BestList containing all the substructures that at least cover one positive example. It is important to mention that all the lists are ordered according to the substructures values. The only variation of the Consistent version of the SubdueCL algorithm is that it only considers substructures covering no negative examples in the BestList.

```
SubdueCL(Gp, Gn, Limit, Beam)
  ParentList = (All substructures of one vertex) mod Beam
  Repeat
    BestList = {}
    Exhausted = FALSE
    While ( (Limit > 0 ) or (ParentList ≠ {}) )
      ChildList = {}
      Foreach substucture in ParentList
        C = Expand(Substructure)
        BestList = ( BestList ∪ CoverPos(C) ) mod Beam
        ChildList = ( ChildList ∪ C ) mod Beam
        Limit = Limit – 1
      EndFor
      ParentList = ChildList mod Beam
    EndWhile
    If BestList = {}
      Then Exhausted = TRUE
      Else Limit = Limit * 1.2
  Until ( ( BestList ≠ {} ) or ( exhausted = TRUE ) )
  Return first(BestList)
End
```

Figure 4: SubdueCL Function

Learning with Graphs

Although the search space that SubdueCL considers to learn from graphs is exponential, empirical results show that it is able to learn without exhaustively reviewing it. Empirical experiments in the following section show this. We are working in a PAC Learning analysis of SubdueCL to support this statement. Using graphs as the knowledge representation for learning has several advantages, among them we have that graphs are a very expressive and comprehensible representation. Another advantage is that the hypothesis space consists of connected graphs, SubdueCL searches using the relations that connect concepts, this makes SubdueCL's approach good for both non-relational and relational domains.

Empirical Results

We are now in the testing phase of our graph-based concept learner, and as part of those tests we are comparing it with the two ILP systems FOIL and Progol. Until now we have tested it in five domains but we will continue with more and different types of 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 vote domain is the Congressional Voting Records Database available from the UCI machine learning repository (Keogh et. al 1998). The diabetes domain is the Pima Indians Diabetes Database, and the credit domain is the German Credit Dataset from the Statlog Project Databases (Keogh et. Al 1998). The Tic-Tac-Toe domain consists of 958 exhaustively generated examples. Positive examples are those where "X" starts moving and wins the game and negative examples are those where "X" looses or the game is tied. The examples are represented by the position on the board and the value for that position. Therefore the possible values for each position are either "X" or "O". The three systems were able to learn the concept correctly, that is; they learned the eight configurations where "X" wins the game. Table 1 shows the percent accuracy results on the non-relational domains. The results show that SubdueCL is competitive with Foil and Progol in these types of domains.

Table 1: Percent accuracy results on non-relational domains

	Golf	Vote	Diabetes	Credit	TTT
FOIL	66.67	93.02	70.66	66.16	100.00
Progol	33.33	76.98	51.97	44.55	100.00
SubdueCL	66.67	94.88	64.21	71.52	100.00

As for relational domains we are working with the chess, and carcinogenesis domains. 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).

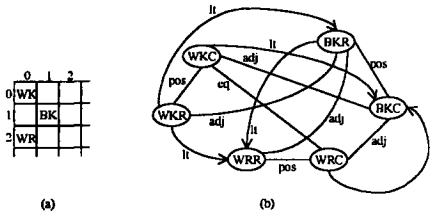

Figure 5. An example from the chess domain. (a) Board configuration and (b) SubdueCL's graphical representation of the example.

Therefore, if white's turn is next, then the positive examples are illegal configurations. Figure 5b shows

Subdue's representation for the chess domain example in figure 5a. Figure 6 shows two substructures found by SucdueCL in the chess domain. 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). Results in the Chess domain show 97% accuracy for SubdueCL, 99% for FOIL, and 86% for Progol. Due to computational constraints only a subset (5000 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. In terms of number of rules, Progol learned 5 rules, FOIL learned 11 rules, and Subdue learned 7 rules (substructures).

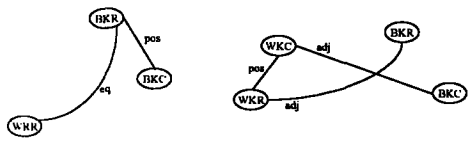

Figure 6. Two of eleven substructures found by SubdueCL in the chess domain.

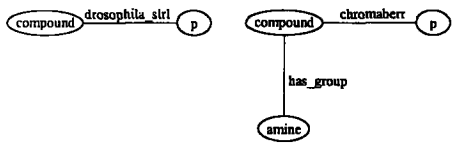

Figure 7. Two of the substructures found by SubdueCL in the cancer domain.

The carcinogenesis database is composed of information about chemical compounds and the results of laboratory tests made to rodents in order to determine if the chemical induces cancer to them or not. The information used for this experiment was taken from the web site: http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/PTE/. This database was built for a challenge to predict the result of the tests using machine-learning techniques. The dataset contains 162 positive examples and 136 negative examples. Figure 7 shows two substructures found by SubdueCL in preliminary results in the cancer domain. The first substructure says that a compound that has a positive "drosophila_slrl" test causes cancer. The second substructure says that a compound that has a positive "chromaberr" test and also has an "amine" group causes cancer.

Conclusions

We have described three approaches to graph-based concept learning: conceptual graphs, the Galois lattice, and SubdueCL. Preliminary theoretical analyses indicate that a constrained version of SubdueCL may PAC learn, but more analysis is necessary. Empirical results indicate that SubdueCL is competitive with the ILP systems FOIL and Progol. Therefore, graph-based relational concept learning is competitive with, and potentially more expressive than logic-based approaches. Future experimentation will compare the two approaches on controlled artificial domains, the cancer domain, and graph representations of the web, where we can learn hyperlink-based relational concepts distinguishing two sets of web pages.

References

Cameron, R. M. and Quinlan, J. R. Efficient top-down induction of logic programs. SIGART, 5(1):33-42, 1994.

Cook, D. J.; Holder, L. B. Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research, 1:231-255, 1994.

Cook, D. J.; Holder, L. B.; and Djoko, S. "Knowledge Discovery from Structural Data," Journal of Intelligence and Information Sciences, Vol. 5, Number 3, pp. 229-245, 1995.

Cook, D. J.; Holder, L. B.; and Djoko, S. Scalable discovery of informative structural concepts using domain knowledge. IEEE Expert, 11(5):59-68, 1996.

Jappy. and Nock, "PAC Learning Conceptual Graphs," Int. Conference in Conceptual Structures, pp. 303 – 318, 1998.

Keogh, E.; Blake, C.; and Merz, C. J. UCI repository of machine learning databases, 1998.

Liquiere, M. and Sallantin, J. "Structural Machine Learning with Galois Lattice and Graphs," Proceedings of the Fifteenth International Conference in Machine Learning, Morgan Kaufmann, pp. 305-13, 1998.

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

Quinlan, J. R. Induction of decision trees. Machine Learning, 1(1):81-106, 1986.

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

Sowa, J., "Conceptual graphs summary," Current Research and Practice, chapter 1, pp 3-52, Ellis Horwood, 1992.

Srinivasan; Muggleton; King; and Sternberg, "Mutagenesis: ILP Experiments in a Non-Determinate Biological Domain," Proceedings of the Fourth Inductive Logic Programming Workshop, 1994.

Valiant, L. "Learning Disjunction of Conjunctions," International Joint Conference on Artificial Intelligence pp. 560 – 566, 1985.