Application of Graph-Based Concept Learning to the Predictive Toxicology Domain

Jesus Gonzalez, Lawrence Holder, Diane Cook

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

1 Introduction

For concept learning systems, data representation is crucial. A good representation might make possible the learning of a concept that was not learnable using other representations. For example, in the case of the earthquake domain (earthquake data such as its location, epicenter, intensity, depth, type, etc.) we can just use an attribute-value representation. However, we are missing important information such as the distance between the earthquakes' epicenters and the difference in time between the earthquakes. This information is lost in an attribute-value representation; information that could make possible the learning of an important concept.

In the case of ILP systems, data is represented in First-Order Predicate Calculus (FOPC) in the form of Prolog logic programs. This representation supports structural relations. ILP systems have been successful in structural domains: Progol in the Chemical Carcinogenicity domain [8] and FOIL for learning patterns in Hypertext domains [6]. Graphs are also a data representation that allows for structural relations. Besides being very flexible, graphs can also represent FOPC using conceptual graphs as introduced by John Sowa [7].

In this research, we present a graph-based concept learning system. A graph-based concept learning system uses graphs to represent the examples of the concept to learn. It also uses graph operations to verify if an example belongs to a concept or not, and to create new concepts. Then, the search space is composed of graphs. Here, we focus on the application of our system to the predictive toxicology database.

2 Graph-Based Discovery

In a graph-based approach, the domain (a database) is described using graphs. These graphs become the input to the graph-based discovery algorithm, which uses a heuristic to choose the sub-graphs that are considered important or discovered knowledge. These issues will be discussed in more detail in this section, where we present a graph-based relational learning approach.

2.1 The Subdue System

Subdue [2] 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 [5], 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 I(*S*) + I(*G*|*S*), where *S* is the substructure discovered in the input graph *G*, I(*S*) is the length (number of bits) required to encode *S*, and I(*G*|*S*) is the length of the encoding of graph *G* after being compressed using substructure *S*, i.e., every instance of *S* in *G* is replaced by a single vertex.

After finding the first substructure, Subdue compresses the graph and may iterate to repeat the same process. Subdue is able to perform a polynomial-time inexact match that allows the discovery of substructures with slight variations. Another important characteristic of Subdue is that it allows the use of previous knowledge in the form of predefined substructures. The following sections describe Subdue's capabilities in detail.

2.2 Knowledge Representation

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 and attributes as objects. 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. A graph representation is flexible enough to permit having more than one possible representation for a given domain, allowing the researcher to experiment to get the best representation for the domain. As an example of the graph representation that we are using see figure 3 in section 4.

2.3 Search Method

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. The search method can also be biased using background knowledge (i.e., substructures that we think may exist in the data, but that we want to study in more detail) provided by the user. In this case, the user gives the background knowledge substructures as input to Subdue. Subdue finds the instances of the background knowledge substructures in the input graph and then continues looking for expansions of those substructures.

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 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 is a disjunction 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.

3.1 Substructure Evaluation

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. Then, 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:

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

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. The positive examples not covered by the substructure and the negative examples covered by the substructure are considered errors. #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 (positive examples already covered in a previous iteration were removed from the training set), and #NegEgs is the total number of negative examples. This number does not change, because negative examples are not removed from the training set. For hypotheses with the same error, those covering more positive examples are preferred.

3.2 SubdueCL Algorithm

The SubdueCL algorithm is shown in figures 1 and 2. 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 1. SubdueCL's main algorithm.

A substructure is added to *H* each time that the SubdueCL function is called. 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 2 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 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, 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.

```
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 in ParentList
             C = Expand(Substructure)
             if CoversOnePos(C,Gp)
                then BestList = BestList ∪ {C}
             ChildList = ( ChildList ∪ C ) mod Beam
             i = i – 1
          endfor
          ParentList = ChildList mod Beam
      endwhile
      if BestList = {} and ParentList ≠ {}
         then Exhausted = FALSE
                 Limit = Limit * 1.2
   until ( Exhausted = TRUE  )
   return first(BestList)
end
```

Figure 2. SubdueCL algorithm.

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 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. All the lists are ordered according to the substructures' values. The order-independence of SubdueCL's input graphs and its tolerance to noise and variability in the graph representation result in high stability in the results.

The SubdueCL system has been applied to a number of other domains, including finding structural signatures of DNA transcription sites and functionally similar proteins [3], determining types of earthquakes and causes of aircraft incidents based on spatial-temporal relationships [1], and identifying patterns in the hyperlink structure of websites [4]. Source code for SubdueCL, as well as publications and sample databases, can be found at http://cygnus.uta.edu/subdue/.

4 Predictive Toxicology Domain

In this section we describe two experiments performed in the predictive toxicology domain. The first experiment uses data from the first two challenges (PTC-1 and PTC-2). The second experiment describes recent results from the current PTC-3 challenge.

4.1 PTC-1 and PTC-2 Experimentation

The PTC carcinogenesis databases contain 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/. Figure 3 shows a portion of the graph representation for one compound.

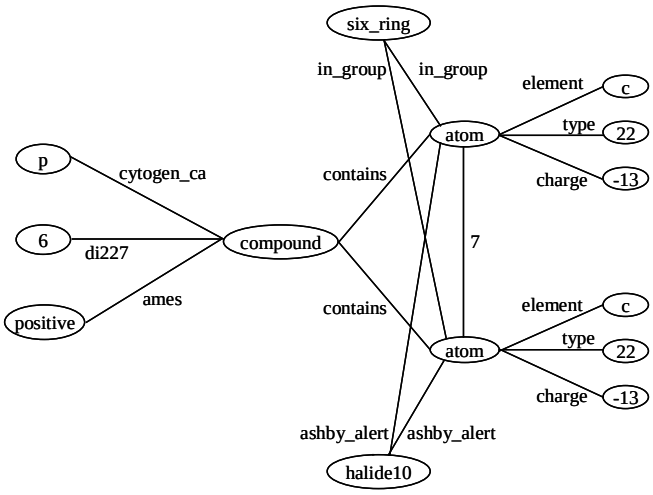

Figure 3. Portion of the graph representation of carcinogensis compound.

The data in this database comes from the predictive toxicology evaluation project conducted by the National Institute of Environmental Health Sciences (NIEHS). 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 used for training and two sets of testing examples, the first called PTE-1 with 20 positive examples and 19 negative examples, and the second called PTE-2 that consists of 30 examples, but at the time of the experiments only 7 were known to be positive, 6 negative and the others were being evaluated.

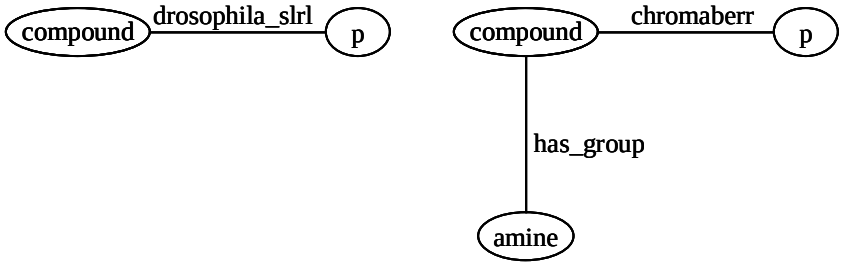

Figure 4. Two of the substructures found by SubdueCL in the cancer domain.

Figure 4 shows two of the substructures that SubdueCL found to describe a compound that produces cancer. The first substructure says that a compound that reported positive in the genetic

toxicology test “drosophila_slrl” produces cancer. The second substructure says that a compound that reported positive in genetic toxicology test “chromaberr” and that also has an amine group produces cancer.

We performed a 10-fold cross-validation on the training data with SubdueCL, which achieved an accuracy of 55.52%. We also performed an experiment training SubdueCL with the 298 training examples and testing on the PTE-1 examples. For this test SubdueCL had a 61.54% accuracy with 14 positive errors (not covering active components) and 1 negative error (covering an inactive component) from the total of 39 components.

Although this is not an encouraging result, we have to consider that this is a very difficult task where the best reported accuracy is 77% using a chemist type system that involved participation of human experts and access to biological data that was not available to Progol or SubdueCL, because we are using the same data that Progol used [8]. In this test Progol achieved an accuracy of 72% as reported in [8]. We tried to reproduce the experiments for Progol, but we could not get the 72% accuracy. Our best result for Progol is 64%, but we are trying to find the right Progol parameters in order to achieve the 72% cited in the literature. It is important to say that the background knowledge used by Progol includes a set of rules related to mutagenesis previously found by Progol. A compound that is found to be mutagenic is more likely to cause cancer. We did not use these rules with SubdueCL. We will use these rules for our experiments in future work, but before doing this we need to study more about the mutagenesis domain and obtain expert evaluation of our results.

4.2 PTC-3 Experimentation

We also applied SubdueCL to the data set provided at “The Predictive Toxicology Challenge 2000 – 2001” web page in prolog format (file KULeuven.tgz). 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 classified according to the class for which they were done (male rats MR, female rats FR, male mice MM, or female mice FM).

The information that we considered for each compound was the name (i.e., tr000), class_MR (classification of this compound for Male Rats), class_FR, class_MM, class_FM, nAmine (number of Amine groups of this compound), nP_amine, nS_amine, nT_amine, nAmine_salt, nImine, nNitrile, nIso_cyanate, nNitro, nCarbonyl2, nester, nCarbonic_acid, nAnhydrid, nAmide, nP_amide, nS_amide, nT_amide, nAldehyde, nAcetale, nKetone, nAlchohol, nFenol, nether, nThio_ether, nSulfonic_acid, nSulfonic_amide, nPhosphate_ester, nPhosphor_amidate, nMehyl, nAlkane, nAlkene, nRingstruct, nAromatic, nAro2n2, and nAro2n. For each atom we included the element, and for each bond, we included an edge linking two atom vertices labeled with the bond type. For groups we linked the name of the group to the compound with edges linking the group name to the atoms that belong to the group. The relation defining the number of each type of group was not used directly, but individual counts were done for each group and added as binary relations to the compound. We did not include the short-term assays or predictive tests as used in the previous experiments.

One of SubdueCL’s current limitations is difficulty in matching real-valued attributes. Two different, real values, not matter how close, constitute a mismatched label to SubdueCL. Facilities are available in SubdueCL to annotate real values with threshold information to allow a more flexible match, but these thresholds must be determined based on expert knowledge of the domain. Therefore, at this point, some of the real-valued attributes were not included in the graph representation. These include the atom coordinates, atom charge, number of atoms in the molecule, weight of the molecule, and distance charges.

We created four training files and four testing files (i.e., a training file for male rats with its corresponding testing file). We chose to present our findings for male rats only due to their achieving maximum accuracy among the other submissions to the challenge. Figures 5-8 show some of the substructures found by SubdueCL for the classification of compounds according to their activity in male rats. Expert evaluation of these results is currently in progress.

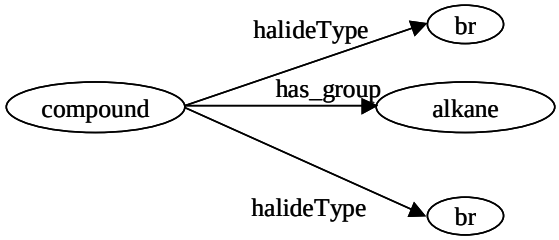

Figure 5. A compound that contains two halide groups of type "br" and that also have an "alkane" group may cause cancer.

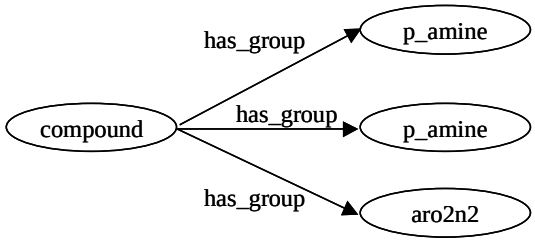

Figure 6. A compound that contains 2 "p_amine" groups and an "aro2n2" groups may cause cancer.

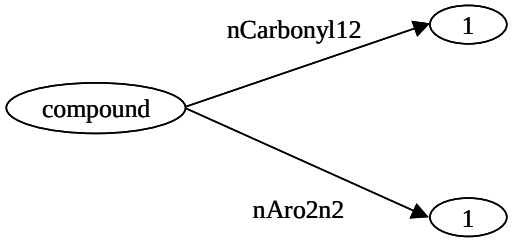

Figure 7. A compound that contains a group "Carbonyl12" and a group "Aro2n2" may cause cancer.

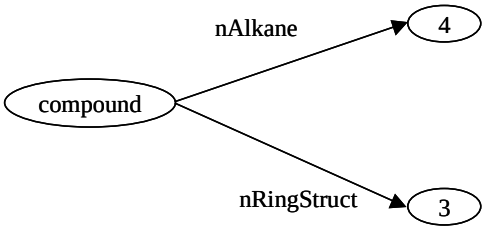

Figure 8. A compound that contains four "Alkane" groups and three "Ring Structures" may cause cancer.

5 Conclusion

In this research we introduced our graph-based concept learning approach and its implementation in the SubdueCL system. In the case of the cancer domain SubdueCL obtained a lower accuracy result than Progol, but this was due to the mutagenic information used by Progol that was not available to SubdueCL. We also show results of SubdueCL on data from the current predictive toxicology challenge. SubdueCL's results in the male rats category achieved maximum performance among the challenge submissions.

There are some enhancements that we need to make to SubdueCL. One enhancement consists in giving 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. This would make SubdueCL more competitive with ILP systems. Another enhancement that would make SubdueCL more competitive with ILP systems would be 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 greater or less than the numeric label of another vertex.

We also want to work further in the cancer domain. We want to add the mutagenetic rules (mentioned in the experiments section) that Progol used to the SubdueCL representation and see if SubdueCL is able to achieve the same accuracy as Progol did in that domain. We also plan to apply SubdueCL to other biological databases. For example, the 2001 Knowledge Discovery in Databases (KDD) Cup focuses on data from genomics and drug design. This and other such domains, where relational information is available, will provide further insight into SubdueCL's ability to efficiently and effectively learn in these domains.

References

1. R. Chittimoori, J. Gonzalez, and L. B. Holder. Structural Knowledge Discovery in Chemical and Spatio-Temporal Domains. *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.
2. D. Cook, and L. B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research*, Volume 1, pages 231-255, 1994.
3. D. Cook, L. B. Holder, S. Su, R. Maglothin, and I. Jonyer. Structural Mining of Molecular Biology Data. *IEEE Engineering in Medicine and Biology*, Volume 20, Number 4, pages 67-74, 2001.
4. J. Gonzalez, Empirical and Theoretical Analysis of Relational Concept Learning Using a Graph-Based Representation. Doctoral Dissertation, University of Texas at Arlington, 2001.
5. J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.
6. S. Slattery and M. Craven. Combining Statistical and Relational Methods for Learning in Hypertext Domains. *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 38-52, 1998.
7. J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.
8. A. Srinivasan, R. D. King, S. H. Muggleton, and M. J. Stenberg. Carcinogenesis Predictions Using ILP. *Proceedings of the Seventh International Conference on Inductive Logic Programming*, pages 273-288, 1997.