

# Learning Relational Knowledge

Larry Holder  
School of Electrical Engineering and Computer Science  
Washington State University  
Box 342752, Pullman, WA 99164  
Email: [holder@wsu.edu](mailto:holder@wsu.edu)

Also available at <http://www.eecs.wsu.edu/~holder/aiproject>

## 1 Introduction

Most approaches to machine learning assume the knowledge to be learned can be expressed as a set of attribute-value pairs, i.e., an entity and its properties. However, much richer knowledge can be found in the relationships between multiple entities. For example, computational chemistry applications look for patterns in the relationships between atoms and molecules, not just patterns in the properties of a set of atoms. Such analysis can discover chemical structures (e.g., a benzene ring); whereas, attribute-value-based learning methods cannot express such knowledge. Relational learning is an area within machine learning that specifically addresses the challenge of learning relational knowledge [7].

There are basically two approaches to learning relational knowledge, and they center around the two main representations for relational knowledge: first-order logic and graphs. First-order logic is vastly more expressive than propositional logic, which is the representation of attribute-value-based learning methods. Several methods exist for learning relational knowledge in the form of first-order logic, and these methods fall under the area of Inductive Logic Programming (ILP) [9]. Graphs (i.e., a collection of nodes and links between nodes) are not quite as expressive as first-order logic, but they are an efficient and intuitive representation for relational knowledge. Several methods exist for learning relational knowledge in the form of graphs, and these methods fall under the area of Graph-based Data Mining or Graph-based Relational Learning [8].

This project will explore the challenge of learning relational knowledge by experimenting with two relational learning methods: one logic based and one graph based. In addition to simple experiments with these learners, the project will apply the methods to an important problem in computational chemistry; namely, learning structures common to chemical carcinogens.

## 2 Project Description

The project is divided into three main parts: experimentation with the PROGOL logic-based relational learner, experimentation with the SUBDUE graph-based relational learner, and application of these learners to the task of predicting the carcinogenicity of chemical compounds. An optional advanced section describes additional tasks that enhance the experience with this project. Throughout the project description, there are several points marked as “Deliverables” that are the expected outputs of the project.

## 2.1 Logic-based Relational Learner: PROGOL

For our first foray into relational learning, we will use the PROGOL logic-based relational learning system [10]. PROGOL is an inductive logic programming (ILP) system that uses inverse entailment, guided by a user-defined mode declaration, to search for a logic theory that entails the correct classification of the training examples. A mode declaration is a constraint which imposes restrictions on the atoms and their arguments appearing in a clause of the learned theory by (1) determining which atoms can occur in the head and the body of the clause, (2) determining which arguments can be input variables, output variables or constants, and (3) determining the number of alternative solutions for instantiating the atom. PROGOL first computes the most specific clause  $S$  which covers a seed example and then uses  $A^*$  search, guided by a compression and accuracy measure, to generalize  $S$  in order to cover more positive examples. PROGOL can also accept arbitrary Prolog clauses as background knowledge that can be referred to by the learned theory.

### 2.1.1 Initial Test with PROGOL

First, we will download, install and test PROGOL. The main download site for PROGOL is given in [3]. After unpacking the distribution, go to the `source` directory and type `make` to compile `progol`. While this version should compile on most UNIX systems using the Gnu `gcc` compiler, you may have to add the `-m32` option to the `CFLAGS` variable in the `source/Makefile`, as the code has problems on 64-bit architectures. You should then read through the tutorial in `man/tutorial4.4.ps`. The tutorial describes the execution of PROGOL on the `examples4.2/aunt.pl` file.

*Deliverable:* Provide the output produced by PROGOL on the `aunt.pl` example and give an English description of the theory learned by PROGOL.

### 2.1.2 Teaching PROGOL the Concept of a Benzene Ring

Now that you have some experience using PROGOL, let's construct our own input file to teach PROGOL the concept of a benzene ring (see Figure 1). A benzene ring is a ring of six carbon (C) atoms connected by alternating single and double bonds. We will ignore the hydrogen (H) atoms.

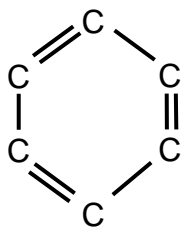


Figure 1: Benzene ring.

We will first attempt to teach PROGOL the concept of benzene by showing it only positive examples of the molecule based on two background predicates: `single_bond(C1, C2)` and `double_bond(C1, C2)`. The specific predicate to learn will be `benzene_ring(C1, C2, C3, C4, C5, C6)`. We can assume the arguments to these predicates are all of type `carbon`.

*Deliverable:* Based on the above scenario, construct a PROGOL input file with ten positive examples of a benzene ring. Following the `aunt.pl` example file, your input file will need to set the `posonly` setting, have mode declarations for the `benzene_ring`, `single_bond`, and `double_bond` predicates, have `carbon` type predicates for each carbon involved in the examples, have background knowledge about the single and double bonds involved in the ten different benzene rings, and then the ten benzene example predicates. Provide your input file and corresponding PROGOL output file. *Hint:* See the following discussion.

Most likely, despite the use of differing representations and number of examples, PROGOL will not produce the expected theory, i.e., a conjunction of three single bond predicates and three double bond predicates with appropriately common variables. However, the theory that PROGOL does produce is correct, i.e., it correctly describes (or “covers”) all the examples. For instance, PROGOL is likely to produce the following theory:

```
benzene_ring(A,B,C,D,E,F) :- single_bond(A,B).
```

The above theory is correct, i.e., every benzene ring contains a single bond between two carbon atoms.

So, how can we get PROGOL to produce the correct theory? We need to give PROGOL some negative examples of a benzene ring, e.g., a benzene ring with one bond missing. Let’s give this a try.

*Deliverable:* Starting with the above input file with ten positive examples of a benzene ring, add six negative examples. Each negative example will be a benzene ring, but with one of the bonds missing (a different bond missing for each of the six negative examples). You will also need to set the ‘`c`’ parameter to 6 (use ‘`:- set(c,6)?`’), which tells PROGOL there can be six predicates in the clause. Note: PROGOL’s running time is exponential in ‘`c`’, so the default is 3. Provide your input file and corresponding PROGOL output file.

If all goes well, PROGOL should learn the following correct theory for the benzene ring.

```
benzene_ring(A,B,C,D,E,F) :-  
    single_bond(A,B),  
    single_bond(C,D),  
    single_bond(E,F),  
    double_bond(A,F),  
    double_bond(B,C),  
    double_bond(D,E).
```

Now that we have some experience learning relational concepts using a logical representation, let’s try some similar tasks using a graph representation.

## 2.2 Graph-based Relational Learner: SUBDUE

The SUBDUE [5] graph-based relational learning system accepts input as a directed graph with labels on vertices and edges. SUBDUE uses a beam search to identify a subgraph (or substructure) of the input graph that best compresses the input 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 Extend Substructure operator. As its name suggests, it extends a substructure in all possible ways by a single edge. SUBDUE's search is guided by the minimum description length (MDL) principle, which seeks to minimize the description length of the entire data set. The evaluation heuristic based on the MDL principle assumes that the best substructure is the one that minimizes the description length of the input graph when compressed by the substructure. The description length of the substructure  $S$  given the input graph  $G$  is calculated as  $DL(G,S) = DL(S) + DL(G|S)$ , where  $DL(S)$  is the description length of the substructure, and  $DL(G|S)$  is the description length of the input graph compressed by the substructure. Subdue seeks a substructure  $S$  that minimizes  $DL(G,S)$ . The search terminates upon reaching a user-specified limit on the number of substructures extended, or upon exhaustion of the search space.

### 2.2.1 Initial Test with SUBDUE

First, we need to download, install and test SUBDUE. The main download site for SUBDUE is given in [4]. After unpacking the distribution, follow the instructions in the README file to install SUBDUE. You should then read through the user's manual in `docs/UsersManual_1.4.pdf`. The manual describes the execution of SUBDUE on the `graphs/sample.g` file.

*Deliverable:* Provide the output produced by SUBDUE on the `sample.g` example and give an English description of the best substructure learned by SUBDUE.

### 2.2.2 Teaching SUBDUE the Concept of a Benzene Ring

Now that you have some experience using SUBDUE, let's construct our own input file to teach SUBDUE the concept of a benzene ring (see Figure 1). Recall that a benzene ring is a ring of six carbon (C) atoms connected by alternating single and double bonds. Again, we will ignore the hydrogen (H) atoms.

We will first attempt to teach SUBDUE the concept of benzene by showing it only positive examples of the molecule based on two relations: `single_bond` and `double_bond`.

*Deliverable:* Based on the above scenario, construct a SUBDUE input file with three positive examples of a benzene ring. Following the `sample.g` example file, your input file will need to define each benzene-ring graph by defining six vertices labeled C for the six carbon atoms, and then six undirected edges, three labeled `single_bond` and three labeled `double_bond`. Since we need to define three such graphs, we could continue with vertices 7-12 and 13-18, but SUBDUE allows you to provide multiple positive examples, each starting from vertex 1, by including the line "XP" just before each example. Provide your input file and the corresponding SUBDUE output file.

If all goes well, SUBDUE should have learned the following correct graph structure for a benzene ring as the best substructure.

```
v 1 C
v 2 C
v 3 C
v 4 C
v 5 C
v 6 C
u 1 2 single_bond
u 3 4 single_bond
u 5 6 single_bond
u 2 3 double_bond
u 4 5 double_bond
u 6 1 double_bond
```

SUBDUE is able to learn the benzene-ring structure using only positive examples (unlike PROGOL), because SUBDUE uses a different heuristic to guide the learning; namely, compression. SUBDUE searches for the structure that maximally compresses the input graphs. In this case, once SUBDUE has enough positive examples of a benzene ring, then this same structure will afford maximal compression when each example is compressed to a single vertex. PROGOL, on the other hand, is guided by the goal of correctly covering the positive examples, and so chooses the smallest subpart contained in every example as its hypothesis.

As an exercise, let's go ahead and give SUBDUE some negative examples of the benzene ring and see what effect they have on SUBDUE's result.

*Deliverable:* Starting with the above SUBDUE input file with three positive examples of a benzene ring, add six negative examples. Each negative example will be a benzene ring, but with one of the bonds missing (a different bond missing for each of the six negative examples). Provide your input file and the corresponding SUBDUE output file.

As you will see, the results are the same; SUBDUE again learns the correct structure.

*Deliverable:* Provide a discussion of your experience learning the benzene-ring concept with PROGOL and SUBDUE in terms the differences in representations, inputs, outputs, and performance.

### **2.3 Application to Computational Chemistry: Mutagenesis**

Chemical and biological data is relational in nature, with atoms related to other atoms by bonds and other forces, along with attributes about the atoms, bonds and entire molecules. One of the benchmark datasets used to evaluate relational learning methods is the Mutagenesis dataset [14], which has been collected to identify mutagenic activity (i.e., most likely causes cancer) in a chemical compound based on its molecular structure. 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 compound's 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 acenaphthyls (Ia). Ia, I1, logP and LUMO are relevant properties in determining mutagenicity. The dataset also contains information which indicates the presence of a chemical concept (e.g., benzene) in the compounds. This dataset was originally designed for use with ILP systems, so the data is given in a logic form using the following relational predicates.

- atom(compound\_id, atom\_id, element, atom\_type, partial\_charge).
- bond(compound\_id, atom\_id, atom\_id, bond\_type).
- logP(compound\_id, hydrophobicity\_value).
- lumo(compound\_id, lumo\_value).
- ind1(compound\_id, boolean).
- inda(compound\_id, boolean).
- chemicalconcept(compound\_id, [atom\_id, atom\_id, ...]).

The Mutagenesis dataset can be downloaded from <http://www.doc.ic.ac.uk/~shm/Software/Datasets/mutagenesis/progol>. The data resides in three directories: 188, 42, and common. The 188 and 42 directories contain multiple files defining whether the 230 compounds are active or not. The common directory gives the mode declarations (mode.pl) and all the background information about the compounds: atoms, bonds, and their properties. The file log\_mutag.pl contains the numeric activity level of each compound, which is used to determine whether a compound is active, but this numeric information is not used as an attribute for learning.

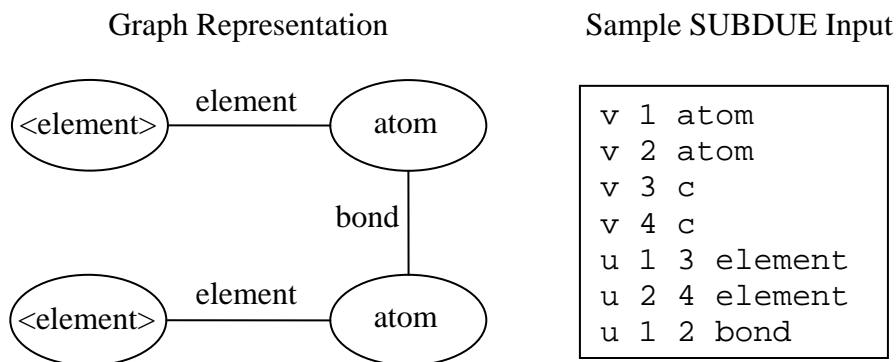
### 2.3.1 Applying PROGOL to the Mutagenesis Data

Since the Mutagenesis data comes already formatted for input to PROGOL, applying PROGOL to this task is straightforward.

*Deliverable:* Construct a PROGOL input file for the Mutagenesis dataset. Specifically, concatenate all the \*.pl files in the common directory (except log\_mutag.pl) and all the files in the 188 and 42 directories into one file. Run PROGOL on this file. Provide your PROGOL input file, PROGOL output, and a description of the learned theory (as best you can without a formal chemistry background). Note: This run takes about 20 minutes on a 3GHz PC.

### 2.3.2 Applying SUBDUE to the Mutagenesis Data

Applying SUBDUE to the Mutagenesis data requires more effort for two reasons. First, the data needs to be converted to a graph. Second, we need to decide how much of the background information to include in the graph and how to represent it. Let's try two different approaches. For the first approach, we will simply include the atoms and bonds for each compound. Specifically, we will create a positive (XP) graph for each active compound and a negative (XN) graph for each inactive compound, where the graphs take the form depicted in Figure 2.



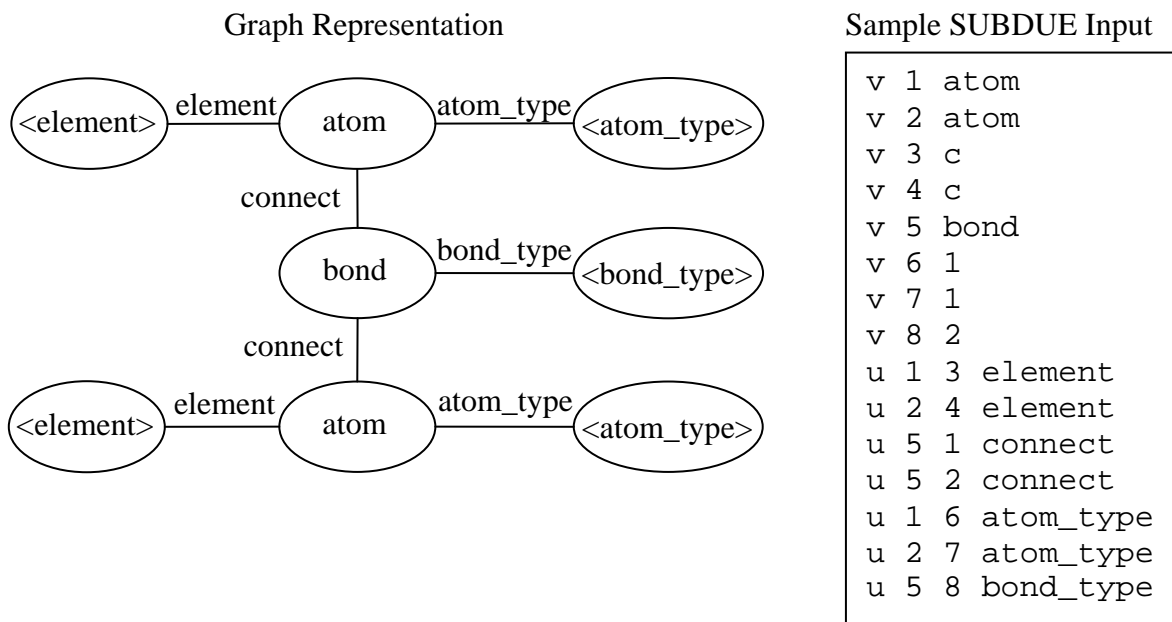
**Figure 2.** Simple graph representation of a chemical compound (left) and sample input to SUBDUE (right).

*Deliverable:* Construct a SUBDUE input file according to the above representation for the Mutagenesis data. The information about atoms and bonds is contained in the *atom* and *bond* relational predicates defined above and in the `common/atom_bond.pl` file. Run SUBDUE on your input file setting the *limit* parameter to 100 (`-limit 100`) and the evaluation method to *set cover* (`-eval 3`). See the following discussion about these parameter settings. Provide your SUBDUE input file, SUBDUE output, and a drawing and description of the best pattern found by SUBDUE. Note: This run takes about 15 minutes on a 3GHz PC.

The limit parameter controls how many different patterns will be tried by SUBDUE. The default value is half the number of edges in the positive graphs (which would be about 4000 for this representation). But since no pattern can grow to be larger than the smallest positive graph, we reduce this parameter to decrease the running time. SUBDUE's evaluation method defaults to preferring patterns compress the positive graphs well, but not the negative graphs (MDL method), but when SUBDUE is run on a set of many smaller positive and negative graphs, there will not be much compression. The set cover evaluation method prefers patterns that merely exist in the positive graphs, but not the negative graphs, which is a more appropriate evaluation method for this case.

For the second approach, let's include some additional information in our graph representation; namely, the atom types and the bond types. Figure 3 depicts this representation and a sample SUBDUE input. Notice that bonds are represented as a "bond" vertex with "connect" edges to the two atoms this bond connects. This is necessary in order to associate a property to a bond; whereas, before when bonds were edges, there was no way to attach such properties.

*Deliverable:* Construct a SUBDUE input file for this second representation for the Mutagenesis data. The information about atom and bond types is also contained in the *atom* and *bond* relational predicates defined above and in the `common/atom_bond.pl` file. Run SUBDUE on your input file this time setting the *limit* parameter to 100 (`-limit 100`), because we have more edges in each graph, and the evaluation method to again set to *set cover* (`-eval 3`). Provide your SUBDUE input file, SUBDUE output, and a drawing and description of the best pattern found by SUBDUE.



**Figure 3. Graph representation of chemical compounds including atom types and bond types.**

*Deliverable:* Provide a discussion of your experience learning relational concepts in the Mutagenesis dataset, and in general, with PROGOL and SUBDUE in terms the differences in representations, inputs, outputs, and performance.

## 2.4 Advanced Project Options

Hopefully, the above exercises have given you an understanding of logic-based and graph-based methods for learning relational knowledge. However, we have only scratched the surface. Below are some advanced project options that will deepen your understanding of this area.

- **Alternate Representations.** There are many ways to represent relational data in logic and in a graph. You experimented with a few alternatives in the above exercises. The choice of representation has a large effect on the performance of relational learning methods. For this project option, design alternative logic and graph representations for the benzene ring and mutagenesis problems. In particular, consider how to represent all the mutagenesis data as a graph, where you introduce a central “compound” vertex from which emanates all atoms, bonds, and compound-level properties.
- **Tuning Parameter Settings.** Like most relational learning systems, PROGOL and SUBDUE have numerous parameters whose settings can have a significant impact on their performance in terms of both efficiency and effectiveness. Perform a systematic study of alternative parameter settings to identify which settings yield the best performance.
- **Other Relational Learners.** In addition to PROGOL and SUBDUE, several other logic-based and graph-based relational learning systems are available. FOIL [1,12] was one of

the first inductive logic programming systems and is still used. In fact, the download website for the mutagenesis data also includes a version of the data for the FOIL system. A slightly different graph-based approach seeks only frequent patterns, not necessarily high-compression patterns. One such system is GASTON [2,11]. Recently, these frequent graph mining systems have been used to generate features for Support Vector Machines (SVMs). SVMs have also been directly applied to graph-based relational learning through the use of graph kernels [6]. Experiment with these different systems and approaches, and compare their performance to PROGOL and SUBDUE.

- Other Relational Datasets. Numerous relational databases exist today that require a relational learning approach to take maximum advantage of the relational data. Two such databases that are used frequently in relational learning studies are the Internet Movie Database (IMDb, [www.imdb.com](http://www.imdb.com)) and the 2003 KDD Cup challenge on citation data ([www.cs.cornell.edu/projects/kddcup/](http://www.cs.cornell.edu/projects/kddcup/)). The IMDb provides information about movies, which can be related according to actors, directors, awards, etc. The KDD Cup citation data provides information relating papers and authors over time. Experiment with different representations, learning tasks, and relational learning methods applied to these, and other relational datasets.

### 3 References

1. FOIL Logic-based Relational Learner. (<http://www.rulequest.com/Personal/>)
2. GASTON Frequent Graph Miner. (<http://www.liacs.nl/~snijssen/gaston>)
3. PROGOL Logic-based Relational Learner. (<http://www.doc.ic.ac.uk/~shm/progol.html>)
4. SUBDUE Graph-based Relational Learner. (<http://www.subdue.org>)
5. D. Cook and L. Holder. Graph-Based Data Mining. *IEEE Intelligent Systems*, 15(2), pages 32-41, 2000. (<http://www.subdue.org/papers/CookIEEE-IS00.pdf>)
6. D. Cook and L. Holder. *Mining Graph Data*. John Wiley and Sons, 2006.
7. S. Dzeroski and N. Lavrac (editors). *Relational Data Mining*. Springer, Berlin, 2001.
8. L. Holder and D. Cook. Graph-Based Relational Learning: Current and Future Directions. *ACM SIGKDD Explorations*, Volume 5, Issue 1, July 2003.
9. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994.
10. S. Muggleton. Inverse Entailment and Progol. *New Generation Computing Journal*, Vol. 13, pp. 245-286, 1995. (<http://www.doc.ic.ac.uk/~shm/Papers/InvEnt.ps.gz>)
11. S. Nijssen and J. Kok. A Quickstart in Frequent Structure Mining can make a Difference. *Proceedings of the 7<sup>th</sup> International Conference on Chemical Structures*, 2005.
12. J.R. Quinlan and R.M. Cameron-Jones. FOIL: A midterm report. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of Lecture Notes in Artificial Intelligence, pages 3-20. Springer-Verlag, 1993.
13. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, Second Edition*. Prentice-Hall, 2003. (<http://aima.cs.berkeley.edu>)
14. A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Mutagenesis: ILP Experiments in a Non-determinate Biological Domain. *Proceedings of the 4th International Workshop on Inductive Logic Programming*, 1994.