Comparing the use of background knowledge by two Inductive Logic Programming systems

Ashwin Srinivasan and S.H. Muggleton

Oxford University Computing Laboratory,
Wolfson Building, Parks Road, Oxford.

R.D. King

Biomolecular Modelling Laboratory,
Imperial Cancer Research Fund,
Lincoln's Inn Fields, London.

Abstract

In this paper we examine the effect of providing progressively more general background information to two Inductive Logic Programming (ILP) systems. The performance of ILP systems may degrade if the background knowledge provided contains large amounts of irrelevant information. However, in many realistic problems of scientific interest, ILP systems can only be given background knowledge that experts in the field believe to be useful. Here we undertake an empirical study of what scientists can expect from ILP systems when they are applied in such a setting. As a test-bed, the problem considered is that of constructing theories for the mutagenic activity of small molecules. In turn, we provide the ILP algorithms *FOIL* and *Progol* with more chemical knowledge and examine the effect, in terms of the predictive accuracy of the theories constructed, a measure of their simplicity, and the time taken to construct them. Interesting results from these experiments are as follows: (a) *FOIL* shows a strong preference for inequality tests on numeric values, often causing it to ignore simpler logical relationships that hold for the data; (b) some concepts involving constant symbols that are not completely determined by a single example can be difficult to learn with *Progol*; (c) for *Progol*, each addition to the background knowledge results in theories that have higher predictive accuracy, are more compact and are constructed faster; (d) for *FOIL*, significant improvements are only apparent along the scale of theory construction time; and (e) in general, as background knowledge increases, the predictive accuracies of *Progol* theories are higher than those of *FOIL*.

1 Introduction

The use of background knowledge to construct explanations for data is a hallmark of Inductive Logic Programming (ILP) [5]. However, the results from experiments concerned with learning simple logic-programs for list processing suggest that the performance of an ILP system is sensitive to the type and amount of background knowledge provided. Background knowledge that contains large amounts of information that is irrelevant to the problem being considered can, and typically does, hinder an ILP system in its search for an adequate explanation of the data [10]. However, without any notion of prior beliefs on the form of such explanations, it seems that such systems would have to rely on being given only information that was *expected* to be relevant. For at least one potential role for ILP systems, this situation would not be unusual. This concerns the use of such systems to help provide some insight into difficult unsolved scientific problems. Restricted ILP successes have already been reported for problems in chemistry and molecular biology [4, 8], using background knowledge identified by experts as being relevant to the problems considered.

How is a restriction to only potentially relevant background knowledge reflected in the performance of an ILP system? Usually, we are only in a position to deliver an *a posteriori* judgement. Within the logical framework of ILP [5], consider starting with some (possibly empty) set of clauses as background knowledge, and a set of clauses comprising an explanation for the data. Additional background information is now provided, and a revised theory is constructed. It is then not unreasonable to consider the performance of the system as having improved if all of the following were observed:

Accuracy. The predictive accuracy of the new theory is higher;

Simplicity. The new theory is 'simpler' on some complexity scale; and

Time. The new theory took less processor time to construct.

In this paper, we undertake a comparative study of two prominent ILP systems: *FOIL* [11] and *Progol* [7]. We investigate if either system shows the trends above, by progressively providing each with more background information concerning a chemistry problem. In doing so we hope to identify empirically any potential problems peculiar to either system.

The problem we consider in this paper comes from the field of organic chemistry, namely discovering rules for mutagenicity in nitroaromatic compounds [2]. Such compounds occur in automobile exhaust fumes, and in the development of industrial compounds. As mutagenic activity has often been found to be linked to carcinogenesis and damage to DNA, it is of immediate interest to determine methods of developing chemicals with low mutagenicity. In normal discourse, chemists typically use graph-based constructs such as atom/bond connectivities

or Cartesian co-ordinates of atoms to describe properties of molecules. This naturally structural setting serves as a test-bed for our empirical study of *FOIL* and *Progol*. We conduct this study in four stages. At each stage the amount of background knowledge is generalised by adding clausal definitions of potentially useful predicates.

The paper is organised as follows. Section 2 clarifies the precise scope of this study. Materials available are described in Section 3, the design of experiments in Section 4, and results in Sections 5. Section 6 concludes this study.

2 Experimental Aim

Our intention here is a comparative study of the performance of the ILP systems *FOIL* and *Progol* when used to construct theories for mutagenesis. By progressively increasing the background knowledge the theories constructed will be assessed along scales of predictive power, simplicity, and time for construction.

3 Materials

3.1 Data

The mutagenic activity of 230 compounds is listed in [2]. In that study, the authors identify these compounds as belonging to two disparate groups of 188 and 42 compounds respectively. Each compound has an associated mutagenicity value obtained from a procedure known as the Ames test. We confine this study to the task of discriminating compounds with positive log mutagenicity from those which have zero or negative log mutagenicity. Further, we concentrate only on the subgroup of 188 compounds, as these are sufficient for the purposes of the comparative study reported here. This subset has also been analysed by neural network methods that employ the back-propagation of errors [13]. The main interest in the group of 42 compounds stems from the fact that they are poorly modelled by statistical methods of regression. Elsewhere, we have looked at constructing ILP theories for this subgroup [12].[1]

Of the 188 compounds, 125 have positive levels of log mutagenicity. These are labelled "active" and constitute the source of positive examples. The remaining 63 are labelled "inactive" and constitute the source of negative examples.

[1] All data described in this paper, along with the Prolog definitions comprising the background knowledge are available by ftp access to *ftp.comlab.ox.ac.uk*. The relevant directory is *pub/Packages/ILP/Datasets/mutagenesis*.

or Cartesian co-ordinates of atoms to describe properties of molecules. This naturally structural setting serves as a test-bed for our empirical study of *FOIL* and *Progol*. We conduct this study in four stages. At each stage the amount of background knowledge is generalised by adding clausal definitions of potentially useful predicates.

The paper is organised as follows. Section 2 clarifies the precise scope of this study. Materials available are described in Section 3, the design of experiments in Section 4, and results in Sections 5. Section 6 concludes this study.

2 Experimental Aim

Our intention here is a comparative study of the performance of the ILP systems *FOIL* and *Progol* when used to construct theories for mutagenesis. By progressively increasing the background knowledge the theories constructed will be assessed along scales of predictive power, simplicity, and time for construction.

3 Materials

3.1 Data

The mutagenic activity of 230 compounds is listed in [2]. In that study, the authors identify these compounds as belonging to two disparate groups of 188 and 42 compounds respectively. Each compound has an associated mutagenicity value obtained from a procedure known as the Ames test. We confine this study to the task of discriminating compounds with positive log mutagenicity from those which have zero or negative log mutagenicity. Further, we concentrate only on the subgroup of 188 compounds, as these are sufficient for the purposes of the comparative study reported here. This subset has also been analysed by neural network methods that employ the back-propagation of errors [13]. The main interest in the group of 42 compounds stems from the fact that they are poorly modelled by statistical methods of regression. Elsewhere, we have looked at constructing ILP theories for this subgroup [12].[1]

Of the 188 compounds, 125 have positive levels of log mutagenicity. These are labelled "active" and constitute the source of positive examples. The remaining 63 are labelled "inactive" and constitute the source of negative examples.

[1] All data described in this paper, along with the Prolog definitions comprising the background knowledge are available by ftp access to *ftp.comlab.ox.ac.uk*. The relevant directory is *pub/Packages/ILP/Datasets/mutagenesis*.

3.2 Background knowledge

Potentially useful background information concerning the molecules is available in the form of (a) primitive structural descriptions of the molecules; (b) specific chemical properties thought to be relevant; and (c) generic chemical knowledge of structural templates. We describe each of these below.

3.2.1 Primitive structural representation of molecules

The atom and bond structures of the 230 compounds were obtained from the standard molecular modelling package QUANTA. For each compound QUANTA automatically obtains the atoms, bonds, bond types (for example, aromatic, single, double etc.), atom types (for example, aromatic carbon, aryl carbon etc.), and the partial charges on atoms. QUANTA automatically classifies bonds into one of 8 types, and atoms into one of 233 types (most of which relate to different types of carbon atoms). The result is a set of Prolog facts of the form:

bond(compound,atom1,atom2,bondtype), stating that *compound* has a bond of *bondtype* between the atoms *atom1* and *atom2*. For example, an aromatic bond between atoms *d2_1* and *d2_2* in drug *d2* is represented by QUANTA as *bond(d2,d2_1,d2_2,7).*

atm(compound,atom,element,atomtype,charge), stating that in *compound atom* has element *element* of *atomtype* and partial charge *charge*. For example, QUANTA encodes the fact that atom *d2_1* in drug *d2* is an aromatic carbon atom with partial charge 0.067 by the fact *atm(d2,d2_1,c,22,0.067).*

QUANTA generates 10136 ground unit clauses on atomic structure and bonding for the 188 molecules.

3.2.2 Hydrophobicity and Lower Unoccupied Molecular Orbital

The authors of [2] describe a number of chemical properties thought to be of potential interest to the mutagenesis problem. Of these two were thought to be highly relevant:

$logP$: log of the compound's octanol/water partition coefficient (a measure of hydrophobicity); and

ϵ_{LUMO}: energy of the compounds lowest unoccupied molecular orbital. This is obtained from a quantum mechanical molecular model.

3.2.3 Generic structural knowledge

Using the atom and bond description, it is possible to define libraries of elementary chemical concepts. Appendix A does this, providing definitions of methyl groups,

nitro groups, aromatic rings, hetero-aromatic rings, connected rings, ring length, and the three distinct topological ways to connect three benzene rings. These definitions are generic to the field of organic chemistry and can be used by ILP systems as 'building-blocks' for more complex theories.

3.2.4 Subsets of background knowledge

The experiments described here are conducted in four stages. Each stage is marked by a generalisation of background knowledge, thus increasing the potential hypothesis space to be explored by an ILP system. Figure 1 tabulates the predicate definitions constituting the background knowledge at each stage. Also included is the number of clauses in each definition. At each stage, definitions of predicates added are those that are expected to be most useful, given the background knowledge already present.

Background	Content	Number of clauses
B1	atoms, bonds, numeric values of partial charges	10137
B2	definitions in *B1*, and numeric inequalities	10139
B3	definitions in *B2*, and $logP$, ϵ_{LUMO}	10515
B4	definitions in *B3*, and structural definitions in Appendix A	11948

Figure 1: Background knowledge provided in the experiments. The calculation of the number of clauses assumes 1 clause each for $=, \leq$ and $\geq$. For the data used here the generic definitions in Appendix A are represented by 1433 ground facts. This transformation is required as *FOIL* is unable to use the non-ground definitions directly.

3.3 Algorithms

In [6], the shortcomings of various ILP algorithms are discussed. We have selected the algorithms *FOIL* and *Progol* as they appear to be the best suited to handle efficiently the non-determinate nature of the data used here.

The *FOIL* version used is 6.2, and is written in the C language. *FOIL6.2* is written by J.R. Quinlan and incorporates features developed by M. Cameron-Jones. Further details are available from J.R. Quinlan (electronic mail: quin-

lan@cs.su.oz.au). For convenience, in the rest of this paper, we shall refer to *FOIL6.2* as *FOIL*.

We use an early Prolog implementation of *Progol*, called *P-Progol*. At the time of writing this paper S.H. Muggleton has implemented a version called *CProgol* in the C language. Details of obtaining this version can be found in [7]. *P-Progol* is available on request from Ashwin Srinivasan (electronic mail: ashwin@comlab.ox.ac.uk). The implementation includes on-line documentation that clarifies the points of difference with the C version. The theory underlying both versions is the same and is described fully in [7]. However, differences in search strategies and pruning facilities imply that given the same language and resource restrictions, the two versions can compute different answers. For convenience, in the rest of this paper, we shall refer to *P-Progol* as *Progol*.

Both *FOIL* and *Progol* construct inductive explanations for data by performing a top-down search of the subsumption lattice of clauses. Since clauses higher in this lattice usually (although not always) have fewer literals than those that are lower, the algorithms typically examine "shorter" clauses first. There are however three main differences between the algorithms:

1. **Unbound versus Bound search space.** For both algorithms, the search space is constrained by the top of the subsumption lattice. However, *Progol* only searches those portions of the lattice that are guaranteed to cover at least one positive example. This confines the search to sub-lattices that are bound both at the top and bottom. *FOIL* on the other hand can search through spaces that eventually prove futile in that none of the positive examples may be covered. Both algorithms employ constructs to limit the number of literals in a clause. In *FOIL* this takes the form of an encoding length restriction. *Progol* requires the maximum number of literals in a clause to be predefined.

2. **Greedy versus Exhaustive search.** *FOIL* employs an incomplete heuristic search guided by information gain. *Progol* performs complete search within restricted sub-lattices, using data compression as a heuristic.

3. **Ground versus Non-ground background knowledge.** All background knowledge provided to *FOIL* has to be encoded as ground atoms. Background knowledge for *Progol* can be arbitrary Prolog programs.

The disparate language of implementation of the two programs is only of concern here in comparison of theory construction times. A direct comparison of these times is avoided here by normalising all times to the time taken to construct a theory with background knowledge *B1*. Implementation issues aside, we expect theory construction to be slower with *Progol*, given the nature of its search and use of intensional background knowledge.

4 Method

For the 188 compounds under study, we adopt the following k-fold cross-validation design:

1. Randomly assign the compounds in the set to k (approximately) equal partitions. Each partition will, in turn, be withheld to form a "test" set. The compounds in the other partitions will provide the "training" data for constructing theories for predicting members in the "active" class (see Section 3.1).
2. With background knowledge $B1, \ldots, B4$
   (a) With each of the k training data sets:
      i. Construct theories using *FOIL* and *Progol*.
      ii. Record the predictions of each theory so constructed on the data set withheld from the algorithm.
   (b) With all compounds, construct a final theory using *FOIL* and *Progol*.
   (c) For each algorithm, the predictive power of its final theory is estimated from the errors in prediction made by its theories in the k trials previously.
   (d) Analyse to find any significant difference in predictive accuracy of theories constructed by the ILP algorithms.
   (e) For each algorithm, theory size is represented by the number of clauses and average number of literals per clause in the final theory.
   (f) For each algorithm, the time taken to construct the final theory is recorded.

For the purposes of this study, k is 10. Empirical studies suggest this value to be sufficient to obtain nearly unbiased estimates of error rates on data sets of the size used here [14]. Both algorithms require specifications of various language constraints before they can construct theories. For reference, the restrictions used in this study and other details required for repeating the experiments are in Appendix B.

4.1 Comparison of performance

Of the three performance attributes considered (accuracy, simplicity and processor time) only the predictive accuracy of theories is really open for further quantitative analysis. Any comparison for significant differences in predictive accuracy should account for the fact that both algorithms are tested on the same sample. A statistical test that does this is the McNemar's test for changes [3]. The null hypothesis

is that the proportion of examples correctly classified by *FOIL* and *Progol* is the same.[2] Differences will be deemed significant only if on completing the calculations for McNemar's test, the resulting probability of differences occurring by chance is ≤ 0.05.

Comparisons between the algorithms on scales of theory complexity and processor time for theory construction are non-parametric. The measure we have adopted for theory complexity is *Number of clauses* $\times$ *Average number of literals per clause*. For both algorithms equality literals in a clause are not counted. As mentioned earlier, a direct comparison of processor times for the two algorithms is inappropriate given that their implementations are in different programming languages. With some degree of experimentation on benchmark problems, it is possible to obtain an empirical correction factor to account for this difference. However, for each algorithm, we have preferred to rescale times in proportion to the time taken by the algorithm to construct a theory with background knowledge *B1*. The algorithms are comparable on this renormalised scale which avoids problems of poor correction factor estimates. The theories themselves, along with actual times taken by each algorithm, can be found in Appendix B. Differences in the normalising constants for *FOIL* and *Progol* provide some indication of the differences in the speed of execution of either algorithm. A discussion of this is in the next section.

5 Experimental results and discussion

The results obtained from the experimental design adopted in Section 4 are shown graphically in Figure 2. Complete tabulations are in Figure 3, and the results of analysing for significant differences in predictive accuracy are in Figure 4. Methods of statistical analysis leading to these results are in Appendix C.

We first highlight the following empirical findings. *Progol* is able to achieve significantly higher predictive performance than *FOIL* with all of the background knowledge in place (that is, *B4*). For *FOIL* the only prominent effect of progressing from *B1* to *B4* is a fall in processor time. In contrast *Progol* theories progressively have higher predictive accuracy, get simpler, and take less time to construct. While the gain in processor time for *Progol* is not nearly as dramatic as those achieved by *FOIL*, the trend is nevertheless clear. The actual times taken by *FOIL* and *Progol* to construct theories with background knowledge *B1* are approximately $5,000$ seconds and $117,000$ seconds respectively (see Appendix B). A note of caution concerning the measure of theory size is in order. Although *Progol*'s theory sizes are almost always smaller than those of *FOIL*, the predictive accuracy of the former's theory is not always higher than that of the latter. This appears to be at odds with the time-honoured Occam principle. Since the measure of predictive

[2] By "correctly classified" we mean that compounds with positive log mutagenicity in the test set are classified as active, and those with zero or negative values are classified as inactive.

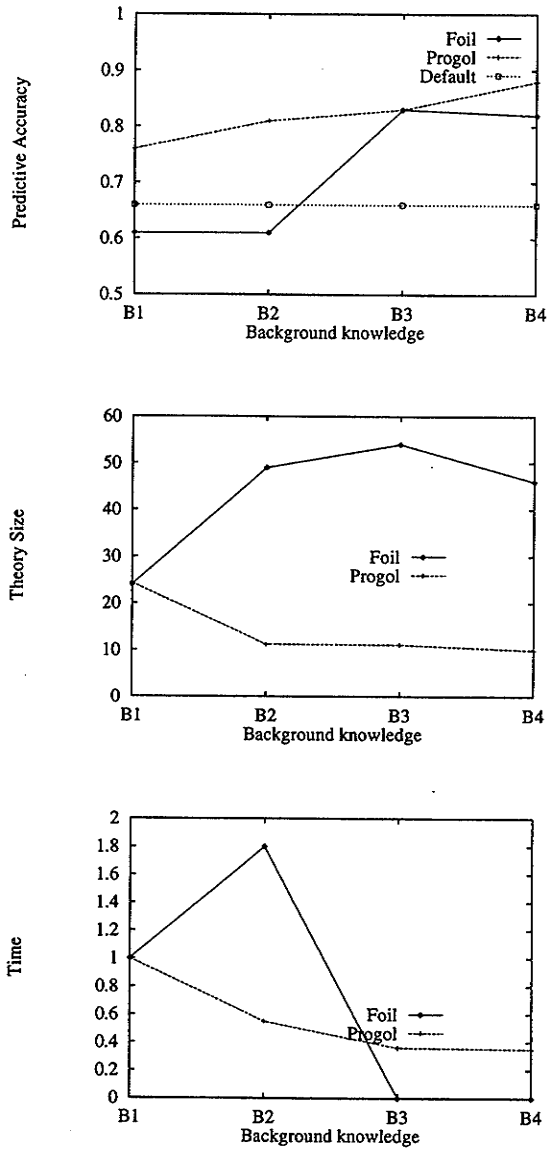

Figure 2: Comparative statistics of *FOIL* and *Progol* theories shown graphically. Accuracy estimates are from a 10-fold cross-validation. Theory Size is (Number of clauses) $\times$ (Average number of literals per clause). Time for each theory is normalised against time taken to construct theory with background knowledge *B1*.

Background Knowledge	Theory Description					
	Accuracy		Size		Time	
	FOIL	*Progol*	*FOIL*	*Progol*	*FOIL*	*Progol*
B1	0.61 (0.06)	0.76 (0.03)	24	24.3	1.0×10^{0}	1.0×10^{0}
B2	0.61 (0.06)	0.81 (0.03)	49	11.2	1.8×10^{0}	5.5×10^{-1}
B3	0.83 (0.03)	0.83 (0.03)	54	11.1	1.0×10^{-4}	3.6×10^{-1}
B4	0.82 (0.03)	0.88 (0.02)	46	9.9	1.0×10^{-4}	3.5×10^{-1}

Figure 3: Comparative statistics of *FOIL* and *Progol* theories for 188 mutagenic compounds. Estimated standard error is shown in parentheses after accuracy value. Size of final theory shown is (Number of clauses) × (Average number of literals per clause). Time for each theory is normalised against time taken to construct theory with background knowledge *B1*.

Background Knowledge	$Accuracy_{FOIL} - Accuracy_{Progol}$	Difference Significant?
B1	−0.15	√
B2	−0.20	√
B3	0.00	×
B4	−0.06	√

Figure 4: Results of analysing for significant differences in predictive accuracy of *FOIL* and *Progol* theories. The analysis is done using McNemar's test for changes. √ indicates difference is significant, and × that difference is not significant at $P = 0.05$.

accuracy is statistically sound, we suspect that the measure for theory size is far from optimal.

5.1 A discussion of *FOIL*

Overall, the performance of *FOIL* on this problem is indifferent, with theories obtained with *B1* and *B2* predicting worse than a default guesser. An evaluation of the chemical principles[3] underlying the clauses has proved difficult given their overly complex nature. The lack of any clear gain in predictive accuracy or simplicity largely detracts from the time improvement. We have noticed two factors that could contribute to this behaviour. First, *FOIL* appears to have a tendency to

[3]By R. D. King in consultation with colleagues at Imperial Cancer Research Fund.

over-specialise. This is apparent in the coverage of positive and negative examples of *FOIL* clauses listed in Appendix B. Second, the algorithm's heuristics appear to bias it strongly in favour of including literals that perform numerical comparisons. We illustrate this with a simple example. Consider learning the target clause:

p(X) :- sine(X,Y), Y $\leq$ 0.

Positive examples of this concept are angles whose *sine* values lie on or below the X axis. Negative examples are all those angles whose *sine* values are above this axis.

Given 33 examples from the range $[-2\pi^c, 2\pi^c]$, (one example every $\frac{\pi^c}{8}$, where the notation 1^c represents 1 radian) and tabulations of *sine* values as background knowledge (again, once every $\frac{\pi^c}{8}$) *FOIL* constructs the theory shown below:

p(A) :- A>2.749.
p(A) :- A$\leq$0, A>-3.534.
p(A) :- A$\leq$-6.283.

Experimenting with examples over longer periods of the *sine* function appears to do little to change this bias. With 1601 examples from the range $[-100\pi^c, 100\pi^c]$ (and correspondingly increasing the size of the *sine* table in background knowledge), a 101 clause definition is found in less than 3 seconds. A fraction of this is shown below:

p(A) :- A>310.621.
p(A) :- A$\leq$307.875, A>304.34.
p(A) :- A$\leq$301.59, A>298.059.
p(A) :- A$\leq$295.309, A>291.773.
p(A) :- A$\leq$289.023, A>285.492.
p(A) :- A$\leq$282.742, A>279.207.
p(A) :- A$\leq$276.457, A>272.926.
...

The clauses obtained on the mutagenicity data are not unlike the ones above, as they consist largely of interval-tests on the numeric quantities $logP$ and ϵ_{LUMO}. In contrast, *Progol* satisfactorily finds the target concept.

5.2 A discussion of *Progol*

Progol demonstrates a clear tendency towards an improved performance on the scales of predictive accuracy, complexity and time. Recall that any clause found by *Progol* is the result of an admissible search through a restricted portion of the subsumption lattice. For single-clause theories, this search technique guarantees that there is no shorter clause (measured by the number of literals) explaining the

set of examples. For such theories at least, new predicate definitions added to the background knowledge can at best allow for a consistent clause with fewer literals than one obtained without these definitions. Since *Progol* is guaranteed to find such a clause, the complexity of the result can only decrease. Unfortunately, this reasoning does not extend to theories that consist of more than one clause (as is the case with the mutagenicity data). For these, all we can say is that it is not unreasonable to expect that augmenting the background knowledge with highly relevant predicates would lead to fewer clauses, each explaining more positive examples. This trend is observed more clearly in the *Progol* theories than the *FOIL* ones. The high coverage, along with the high training accuracy required by the constraints given to *Progol* (see Appendix B) provide sufficient statistical reason to expect a correspondingly high predictive accuracy for each clause in the theory [1].

Given the nature of *Progol*'s complete search, there appears to be no reason to expect a decrease in processor time. For the data here, *Progol* benefits from the fact that the background predicates are sufficiently powerful to allow very short, large coverage clauses. This allows the algorithm to dismiss (admissably) significant portions of the subsumption lattice that consist of longer clauses with same or less coverage. Of the two algorithms, *FOIL*'s times for theory construction are considerably less than *Progol*'s. Particularly striking are the *FOIL* times for background knowledge *B3* and *B4* (0.5 seconds). We suspect that this is largely due to the bias for numerical comparisons highlighted earlier. It is difficult to make any direct comparison of the normalising constants mentioned earlier, given *Progol*'s search strategy, its use of intensional background knowledge and implementation in the Prolog language. Our initial comparisons suggest that on small artificial datasets, a C version of *Progol* is about 5 times faster than the Prolog encoding. If this difference is rigorously shown to hold on large problems such as the one here, then the results here suggest that where no numerical comparisons are performed (with background knowledge *B1*) *FOIL* would be about 5 times faster than a C version of *Progol*.

We conclude this discussion by examining more closely the nature and consequences *Progol*'s search technique. For simplicity, we consider the construction of definite clauses with at most 3 negative (body) literals. Further, background knowledge is restricted to atoms (including their partial charges) and bonds only. Recall that atom and bond definitions are encoded as ground unit clauses of the form:

atm(compound,atom,element,atomtype,charge), and
bond(compound,atom1,atom2,bondtype)

The language restrictions considered here force *compound*, *atom*, *atom1* and *atom2* to be variables, and the rest to be constants. These restrictions are slightly more stringent than those used by *Progol* with background knowledge *B1*, which

allows for clauses that leave partial charges unspecified. Hypothesised clauses with the restrictions here look like:

```
active(A) :- atm(A,B,c,22,-0.142).
active(A) :- atm(A,B,c,22,-0.142), atm(A,B,h,3,0.144).
active(A) :- atm(A,B,c,22,-0.118), bond(A,D,E,1), bond(A,E,B,7).
...
```

For the purposes of analysis, it is convenient to think of each *atm/5* definition as being equivalent to one that has the form:

```
atm(compound,atom,atype)
```

where each tuple of *element, atomtype, charge* maps to a unique constant. First consider the space of all possible definite clauses with at most 3 negative literals. The tree-diagram in Figure 5 shows the choices available when selecting these literals.

The 188 compounds considered in this study have 584 unique *element, atomtype, charge* tuples and 6 bond types. Denoting the number of ways of selecting n items from m objects as ${}^{m}C_{n}$, inspection of the tree-diagram shows that the number of clauses with exactly 1 negative literal is ${}^{584}C_1 + {}^{6}C_1$, with exactly 2 negative literals is approximately ${}^{584}C_2 + {}^{584}C_1 \cdot {}^{6}C_1 + {}^{584}C_1 \cdot {}^{6}C_1 + 2 \cdot {}^{6}C_2$, and with exactly 3 negative literals is approximately ${}^{584}C_3 + 3 \cdot {}^{584}C_2 \cdot {}^{6}C_1 + 5 \cdot {}^{584}C_1 \cdot {}^{6}C_2 + 4 \cdot {}^{6}C_3$. The calculations show that the total number of clauses of at most 3 literals is therefore $\approx 3.5 \times 10^7$. The approximations over-correct for choices that result in equivalent combinations of literals, and thus result in an under-estimate of the number of clauses. For example, the calculation only accounts for those combinations of *atm* literals where the types are distinct. This need not necessarily be the case. *Progol* does not search all of this space. Instead, an example is selected at random and the most specific clause, denoted $\perp$, that implies this example (within the language restrictions imposed) is constructed. Only clauses that subsume $\perp$ are then examined by selecting literals from $\perp$. For the restrictions adopted here this procedure translates to randomly selecting an "active" compound, constructing a $\perp$ clause with positive literal *active(A)* and negative *atm* and *bond* literals. An example $\perp$ would thus look like:

```
active(A) :-
        atm(A,B,c,22,-0.142), atm(A,B,c,22,-0.142),
        atm(A,B,h,3,0.144), atm(A,B,c,22,-0.118),
        bond(A,D,E,1), bond(A,E,B,7),
        ...
```

Clauses constructed by *Progol* are obtained by selecting literals from such a clause. On average, a $\perp$ clause for the 188 compounds considered here has 26 *atm*

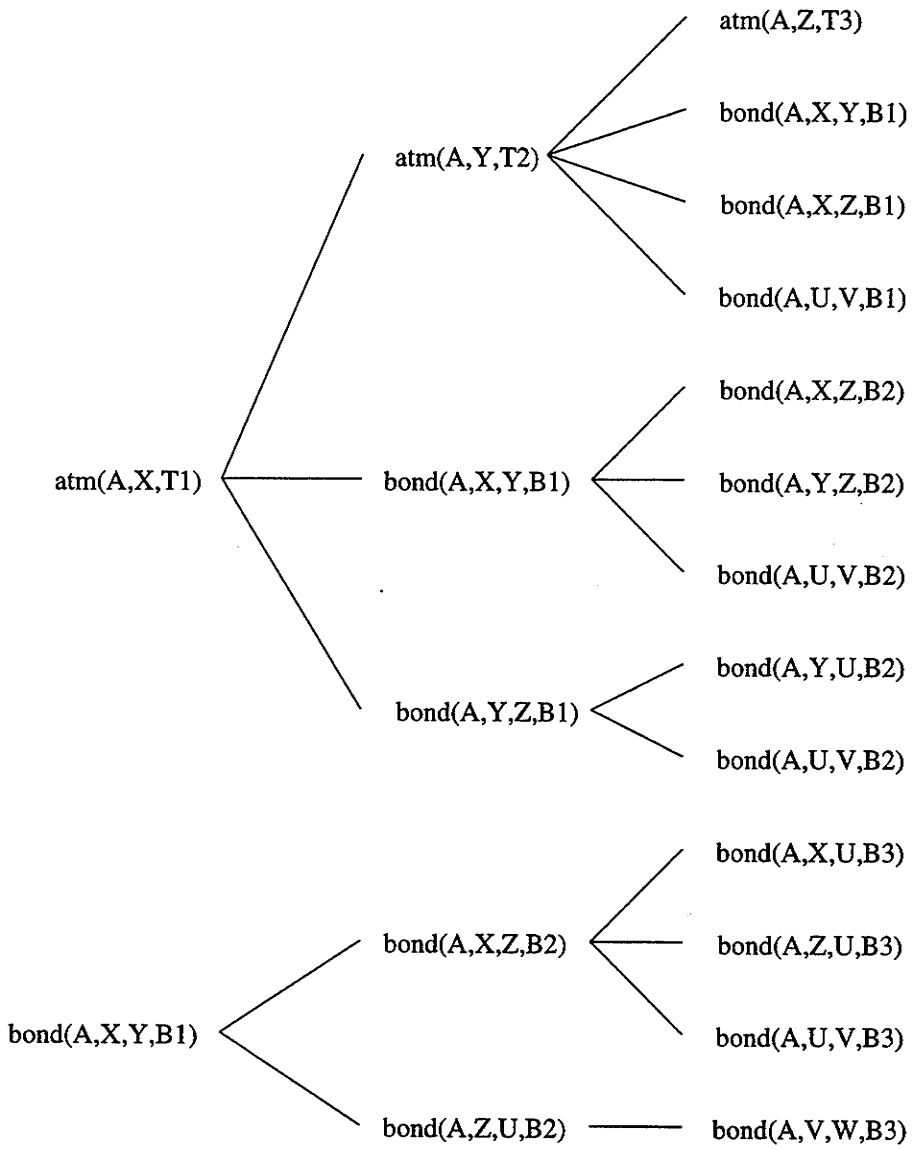

Figure 5: Tree-diagram of choices for body literals in clauses with at most 3 negative literals. Paths that result in combinations of literals that are equivalent (up to a renaming of variables) to others in the tree have been removed.

literals and 27 *bond* literals. Thus, on average, *Progol* looks at ${}^{26}C_1 + {}^{27}C_1$ clauses with 1 negative literal, ${}^{26}C_2 + {}^{26}C_1 \cdot {}^{27}C_1 + {}^{27}C_2$ clauses with 2 negative literals, and ${}^{26}C_3 + {}^{26}C_2 \cdot {}^{27}C_1 + {}^{26}C_1 \cdot {}^{27}C_2 + {}^{27}C_3$ clauses with 3 negative literals. This gives a sum total of $\approx 2.5 \times 10^4$ clauses of at most 3 negative literals. Therefore, the space of clauses searched by *Progol* is on average at least 3 orders of magnitude smaller than the space of all possible clauses that comply with the language restrictions. Intelligent pruning should result in further savings. For example, on average, *Progol* only constructs ≈ 2500 clauses with background knowledge *B1*.

In performing this type of search *Progol* relies largely on assumptions that both the maximum number of negative literals and the size of $\perp$ clauses are small. In the problem considered here, the former has been restricted to 3, and the latter has been at most 150. Besides these restrictions, using a $\perp$ clause from a single example to guide the search for clauses can make it difficult to construct automatically theories for certain types of concepts. Consider, for example the following target clause:

$$p(X) \text{ :- } X \geq 0, X \leq \pi.$$

Positive examples of this concept are X values in the intervals $[0, \pi]$. Assume the availability of built-in predicates $=<$ and *number/1*, and the following determinate definitions for $\leq$ and $\geq$ as background knowledge:

```
X ≤ Y :- number(X), number(Y), !, X =< Y.
X ≤ X :- number(X).

X ≥ Y :- number(X), number(Y), !, Y =< X.
X ≥ X :- number(X).
```

The $\perp$ clause from any positive example *p(k)* where k lies in the interval $[0, \pi]$, would be:

$$p(X) \text{ :- } X \geq k, X \leq k.$$

Inequality tests in all clauses that subsume such a $\perp$ clause will only involve the chosen constant k. Since interval expressed in the target clause requires the presence of *2* constants (0 and π), *Progol* will be unable to construct any approximation to the concept. A possible solution to this problem is to use non-determinate definitions of $\leq$ and $\geq$:

```
X ≤ Y :- float(X), float(Y), X =< Y.
X ≤ Y :- float(X), float(Delta), Y is X + Delta.

X ≥ Y :- float(X), float(Y), !, Y =< X.
```

```
X ≥ Y :- float(X), float(Delta), Y is X − Delta.

float(X) :- number(X), !.
float(X) :- member(X,[0,0.1,0.2,0.4,0.8,1.6,3.2,6.4]).
```

This allows $\perp$ clauses of the form:

```
p(X) :-
        X ≥ k, X ≥ x1, ...,
        X ≤ k, X ≤ y1, ...
```

Given an appropriate positive example, *Progol* can now find subsuming clauses that approximate the target clause. Clearly, there are limits to which such an approach can be pursued.

6 Conclusion

The ability to include and use logical specifications of prior knowledge is a characteristic feature of all modern ILP systems. This should allow a form of automated data analysis that benefits from well-established scientific and engineering knowledge. In this paper a classic chemistry problem forms the basis for examining the exploitation of such knowledge by two prominent ILP systems: *FOIL* and *Progol*. By incrementally providing both algorithms with more chemical knowledge, we have sought to examine their performance along yardsticks of predictive accuracy, theory complexity and theory construction time. Our results suggest that while an improvement in performance is possible, the *FOIL* algorithm is unable to achieve this. This seems in part, at least, due to the heuristics employed in its search for consistent hypotheses.

A surprising empirical finding for this domain is that good results are obtainable by using an admissible search. That such a search is even tractable is not apparent *a priori* from the molecular structure of the compounds considered, as demonstrated by the simplified analysis at the end of the previous section. For *Progol* tractability appears to result from restricting the search space using a clause that is guaranteed to imply at least one example. Given that the language constraints employed in constructing this clause (Appendix B) are not unduly restrictive, the results suggest that admissible search using a *Progol*-type approach may be feasible for other problems.

In the absence of an accepted theoretical model to predict the performance of ILP systems, empirical evidence of an algorithm's ability to construct simple and accurate theories in reasonable time supports its claim for a role in the analysis of scientific and engineering data. Clearly, the restriction to ground definitions of background knowledge can severely restrict the use of *FOIL*. In cases where this restriction is crippling, a variant of *FOIL* that permits background knowledge to

be arbitrary logic programs (for example, *FOCL* [9]) could prove to be more useful. For the problem considered here, the restriction to ground-level definitions has not proved to be an impediment. However the results suggest that care should be exercised when dealing with numerical data, as *FOIL* has a tendency to construct theories that exclusively involve the use of numeric inequalities. Equally, should such be the nature of theories required, then *Progol* would have considerable difficulty in constructing them. To this extent, the each algorithm has something to offer the other.

Acknowledgements

This research was supported partly by the Esprit Basic Research Action ILP (project 6020), the SERC project 'Experimental Application and Development of ILP' and an SERC Advanced Research Fellowship held by Stephen Muggleton. Stephen Muggleton is also supported by a Research Fellowship of Wolfson College Oxford. Ross King is supported by the Imperial Cancer Research Fund. Thanks are due to Rui Camacho and David Page for discussions on *Progol*. The authors also acknowledge helpful comments made by the anonymous referees of this paper.

References

[1] J. Cussens. Bayes and Pseudo-Bayes Estimates of Conditional Probabilities and their Reliability. In *European Conference on Machine Learning*. Springer-Verlag, Berlin, 1993.

[2] A.K. Debnath, R.L Lopez de Compadre, G. Debnath, A.J. Schusterman, and C. Hansch. Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786 – 797, 1991.

[3] B.S. Everitt. *The analysis of contingency tables*. Chapman and Hall, London, second edition, 1992.

[4] R. King, S. Muggleton, and M.J.E. Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proc. of the National Academy of Sciences*, 89(23):11322–11326, 1992.

[5] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[6] S. Muggleton. Inductive logic programming: derivations, successes and shortcomings. *SIGART Bulletin*, 5(1):5–11, 1994.

[7] S. Muggleton. Inverse Entailment and Progol. *New Gen. Comput.*, 13:245–286, 1995.

[8] S. Muggleton, R. King, and M. Sternberg. Predicting protein secondary structure using inductive logic programming. *Protein Engineering*, 5:647–657, 1992.

[9] M. Pazzani and D. Kibler. The role of prior knowledge in inductive learning. *Machine Learning*, 9:54–97, 1992.

[10] J. R. Quinlan. FOIL: a midterm report. In *European Conference on Machine Learning*. Springer-Verlag, Berlin, 1993.

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

[12] A. Srinivasan, S.H. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the Fourth International Inductive Logic Programming Workshop*. Gesellschaft fur Mathematik und Datenverarbeitung MBH, 1994. GMD-Studien Nr 237.

[13] D. Villemin, D. Cherqaoui, and J.M. Cense. Neural network studies: quantitative structure-activity relationship of mutagenic aromatic nitro compounds. *J. Chim. Phys*, 90:1505–1519, 1993.

[14] S.M. Weiss and C.A. Kulikowski. *Computer systems that learn.* Morgan Kaufmann, San Mateo, CA, 1991.

A Some elementary chemical concepts defined in terms of the atom and bond structure of molecules

The following are Prolog definitions for some simple chemical concepts that can be defined directly using the atomic and bond structure of a molecule.

```
% In the following QUANTA bond type 7 is aromatic.

% Three benzene rings connected linearly
anthracene(Drug,[Ring1,Ring2,Ring3]) :-
        benzene(Drug,Ring1), benzene(Drug,Ring2),
        Ring1 @> Ring2, interjoin(Ring1,Ring2,Join1),
        benzene(Drug,Ring3), Ring1 @> Ring3,
        Ring2 @> Ring3, interjoin(Ring2,Ring3,Join2),
        \+ interjoin(Join1,Join2,_), \+ members_bonded(Drug,Join1,Join2).
```

```
% Three benzene rings connected in a curve
phenanthrene(Drug,[Ring1,Ring2,Ring3]) :-
        benzene(Drug,Ring1), benzene(Drug,Ring2),
        Ring1 @> Ring2, interjoin(Ring1,Ring2,Join1),
        benzene(Drug,Ring3), Ring1 @> Ring3,
        Ring2 @> Ring3, interjoin(Ring2,Ring3,Join2),
        \+ interjoin(Join1,Join2,_), members_bonded(Drug,Join1,Join2).


% Three benzene rings connected in a ball
ball3(Drug,[Ring1,Ring2,Ring3]) :-
        benzene(Drug,Ring1), benzene(Drug,Ring2),
        Ring1 @> Ring2, interjoin(Ring1,Ring2,Join1),
        benzene(Drug,Ring3), Ring1 @> Ring3,
        Ring2 @> Ring3, interjoin(Ring2,Ring3,Join2),
        interjoin(Join1,Join2,_).

% A pair of connected rings
members_bonded(Drug,Join1,Join2) :-
        member(J1,Join1), member(J2,Join2),
        bondd(Drug,J1,J2,7).

% A ring with 6 atoms
ring_size_6(Drug,Ring_list) :-
        atoms(Drug,6,Atom_list,_), ring6(Drug,Atom_list,Ring_list,_).

% A ring with 5 atoms
ring_size_5(Drug,Ring_list) :-
        atoms(Drug,5,Atom_list,_), ring5(Drug,Atom_list,Ring_list,_).

% benzene - 6 membered carbon aromatic ring
benzene(Drug,Ring_list) :-
        atoms(Drug,6,Atom_list,[c,c,c,c,c,c]),
        ring6(Drug,Atom_list,Ring_list,[7,7,7,7,7,7]).

% 5 membered carbon aromatic ring
carbon_5_aromatic_ring(Drug,Ring_list) :-
        atoms(Drug,5,Atom_list,[c,c,c,c,c]),
        ring5(Drug,Atom_list,Ring_list,[7,7,7,7,7]).

% 6 membered carbon aromatic ring
carbon_6_ring(Drug,Ring_list) :-
        atoms(Drug,6,Atom_list,[c,c,c,c,c,c]),
        ring6(Drug,Atom_list,Ring_list,Bond_list),
        Bond_list \== [7,7,7,7,7,7].
```

```
% 5 membered carbon non-aromatic ring
carbon_5_ring(Drug,Ring_list) :-
        atoms(Drug,5,Atom_list,[c,c,c,c,c]),
        ring5(Drug,Atom_list,Ring_list,Bond_list),
        Bond_list \== [7,7,7,7,7].

% 6 membered aromatic ring
hetero_aromatic_6_ring(Drug,Ring_list) :-
        atoms(Drug,6,Atom_list,Type_list),
        Type_list \== [c,c,c,c,c,c],
        ring6(Drug,Atom_list,Ring_list,[7,7,7,7,7,7]).

% 5 membered aromatic ring
hetero_aromatic_5_ring(Drug,Ring_list) :-
        atoms(Drug,5,Atom_list,Type_list),
        Type_list \== [c,c,c,c,c],
        ring5(Drug,Atom_list,Ring_list,[7,7,7,7,7]).

% non hydrogen atoms in a drug and their types
atoms(Drug,1,[Atom],[T]) :-
        atm(Drug,Atom,T,_,_),
        T \== h.
atoms(Drug,N1,[Atom1|[Atom2|List_a]],[T1|[T2|List_t]]) :-
        N1 > 1, N2 is N1 - 1,
        atoms(Drug,N2,[Atom2|List_a],[T2|List_t]), atm(Drug,Atom1,T1,_,_),
        Atom1 @> Atom2, T1 \== h.

% 6 atoms connected together to form a ring structure
ring6(Drug,[Atom1|List],[Atom1,Atom2,Atom4,Atom6,Atom5,Atom3],
        [Type1,Type2,Type3,Type4,Type5,Type6]) :-
        bondd(Drug,Atom1,Atom2,Type1), memberchk(Atom2,[Atom1|List]),
        bondd(Drug,Atom1,Atom3,Type2), memberchk(Atom3,[Atom1|List]),
        Atom3 @> Atom2, bondd(Drug,Atom2,Atom4,Type3),
        Atom4 \== Atom1, memberchk(Atom4,[Atom1|List]),
        bondd(Drug,Atom3,Atom5,Type4), Atom5 \== Atom1,
        memberchk(Atom5,[Atom1|List]), bondd(Drug,Atom4,Atom6,Type5),
        Atom6 \== Atom2, memberchk(Atom6,[Atom1|List]),
        bondd(Drug,Atom5,Atom6,Type6), Atom6 \== Atom3.

% 5 atoms connected together to form a ring structure
ring5(Drug,[Atom1|List],[Atom1,Atom2,Atom4,Atom5,Atom3],
        [Type1,Type2,Type3,Type4,Type5]) :-
        bondd(Drug,Atom1,Atom2,Type1), memberchk(Atom2,[Atom1|List]),
        bondd(Drug,Atom1,Atom3,Type2), memberchk(Atom3,[Atom1|List]),
        Atom3 @> Atom2, bondd(Drug,Atom2,Atom4,Type3),
```

```
        Atom4 \== Atom1, memberchk(Atom4,[Atom1|List]),
        bondd(Drug,Atom3,Atom5,Type4), Atom5 \== Atom1,
        memberchk(Atom5,[Atom1|List]), bondd(Drug,Atom4,Atom5,Type5),
        Atom5 \== Atom2.

% drug has a nitro group
nitro(Drug,[Atom0,Atom1,Atom2,Atom3]) :-
        atm(Drug,Atom1,n,38,_), bondd(Drug,Atom0,Atom1,1),
        bondd(Drug,Atom1,Atom2,2), atm(Drug,Atom2,o,40,_),
        bondd(Drug,Atom1,Atom3,2), Atom3 @> Atom2,
        atm(Drug,Atom3,o,40,_).

% drug has a methyl group
methyl(Drug,[Atom0,Atom1,Atom2,Atom3,Atom4]) :-
        atm(Drug,Atom1,c,10,_), bondd(Drug,Atom0,Atom1,1),
        atm(Drug,Atom0,Type,_,_), Type \== h,
        bondd(Drug,Atom1,Atom2,1), atm(Drug,Atom2,h,3,_),
        bondd(Drug,Atom1,Atom3,1), Atom3 @> Atom2,
        atm(Drug,Atom3,h,3,_), bondd(Drug,Atom1,Atom4,1),
        Atom4 @> Atom3, atm(Drug,Atom4,h,3,_).


% utility predicate for intersection of two sets of atoms
interjoin(A,B,C) :-
        intersection(A,B,C), C \== [].

% bonds are commutative
bondd(Drug,Atom1,Atom2,Type) :-  bond(Drug,Atom2,Atom1,Type).

member(X,[X|_]).
member(X,[_|T]):- member(X,T).

% a pair of rings are connected
connected(Ring1,Ring2):-
        Ring1 \= Ring2, member(Atom,Ring1),
        member(Atom,Ring2), !.
```

B Constraints used and theories obtained

B.1 Constraints used by *FOIL*

Default settings were used for all optional parameters used by *FOIL*. Type and functional constraints on the background predicates are as in Figure 6. Type definitions are either enumerations of the ground terms in the success set of the

background knowledge, or are 'continuous' types. The type definitions used allow constants to appear in clauses. This appears to be of importance only to results with background knowledge *B1* and *B2*.

Predicate	*FOIL* constraint
atm/5	atm(drug,atomid,element,atype,charge) # - - - -
bond/4	bond(drug,atomid,atomid,btype) # - - -
lumo/2	lumo(drug,energy) # -
logp/2	logp(drug,hydrophob) # -
anthracene/2	anthracene(drug,ringnum) # -
ball3/2	ball3(drug,ringnum) # -
benzene/2	benzene(drug,ringnum) # -
carbon_5_aromatic_ring/2	carbon_5_aromatic_ring(drug,ringnum) # -
carbon_6_ring/2	carbon_6_ring(drug,ringnum) # -
connected/2	connected(ringnum,ringnum)
hetero_aromatic_5_ring/2	hetero_aromatic_5_ring(drug,ringnum) # -
hetero_aromatic_6_ring/2	hetero_aromatic_6_ring(drug,ringnum) # -
member/2	member(ringnum,ringnum) - #
methyl/2	methyl(drug,ringnum) # -
nitro/2	nitro(drug,ringnum) # -
phenanthrene/2	phenanthrene(drug,ringnum) # -
ring_size_5/2	ring_size_5(drug,ringnum) # -
ring_size_6/2	ring_size_6(drug,ringnum) # -

Figure 6: Constraints for *FOIL*.

B.2 Constraints used by *Progol*

For *Progol* a hypothesis language $\mathcal{L}$ is defined in terms of

- Mode declarations which state the 'forms' that atoms in hypothesis can take in terms of
  - the places where variables are allowed an whether they are inputs or outputs (indicated by $+$ or $-$);
  - the places where constants are allowed (indicated by #);
  - the types of these variables and constants; and
  - the degree of indeterminacy when making such a call to the background knowledge. This is either a number or $*$ meaning finite but unbounded recall of the goal.
- the maximum number of layers of variables introduced by atoms in the body of the clause from variables in the head of the clause;

- the acceptable level of consistency in terms of the maximum number of negatives that can be covered by any clause; and
- the maximum cardinality of any clause.

For the mutagenesis problem, $\mathcal{L}$ is defined in Figure 7.

In addition to the structural definitions, *Progol* was also provided the following information as background knowledge. These concern type-definitions and definitions for inequalities. *FOIL* was provided with type declarations to achieve the same effect, and inequalities are built-in predicates for that algorithm.

```
% compounds are named by an alphabet followed by a number
compound(D):-
        name(D,[_|X]), name(Num,X), int(Num),
        Num >= 1, Num =< 230, !.

% atoms are identified by the compound name, followed by an "_",
% followed by a unique number
atomid(A):-
        name(A,[_|X]), append(Z,[95|Y],X), name(N1,Y),
        name(N2,Z), int(N1), int(N2),
        N2 >= 1, N2 =< 230, N1 =< 500, !.

append([],A,A).
append([H|T],A,[H|T1]):-
        append(T,A,T1).

% charge, epsilon lumo and log P are all floating point numbers.
charge(X):-
        number(X).
energy(X):-
        number(X).
hydrophob(X):-
        number(X).


% chemical elements comprising the atoms
element(br).  element(c).  element(cl).  element(f).  element(h).
element(i).  element(n).  element(o).  element(s).

% inequality definitions
gteq(X,Y):-
        not(var(X)), not(var(Y)), !,
        number(X), number(Y),
        X >= Y.
gteq(X,X):-
        not(var(X)),
```

Mode declarations	mode(*,bond(+compound,−atomid,−atomid,#integer))
	mode(*,bond(+compound,−atomid,+atomid,#integer))
	mode(*,bond(+compound,+atomid,−atomid,#integer))
	mode(*,bond(+compound,+atomid,+atomid,#integer))
	mode(*,atm(*,+compound,+atomid,#element,#integer,-charge))
	mode(*,atm(*,+compound,−atomid,#element,#integer,-charge))
	mode(1,(+charge)=(#charge))
	mode(1,lumo(+compound,−energy))
	mode(1,logp(+compound,−hydrophob))
	mode(1,gteq(+charge,#number))
	mode(1,gteq(+energy,#number))
	mode(1,gteq(+hydrophob,#number))
	mode(1,lteq(+charge,#number))
	mode(1,lteq(+energy,#number))
	mode(1,lteq(+hydrophob,#number))
	mode(*,benzene(+drug,−ring))
	mode(*,carbon_5_aromatic_ring(+drug,−ring))
	mode(*,carbon_6_ring(+drug,−ring))
	mode(*,hetero_aromatic_6_ring(+drug,−ring))
	mode(*,hetero_aromatic_5_ring(+drug,−ring))
	mode(*,ring_size_6(+drug,−ring))
	mode(*,ring_size_5(+drug,−ring))
	mode(*,nitro(+drug,−ring))
	mode(*,methyl(+drug,−ring))
	mode(*,anthracene(+drug,−ringlist))
	mode(*,phenanthrene(+drug,−ringlist))
	mode(*,ball3(+drug,−ringlist))
	mode(1,member(+ring,+ringlist))
	mode(1,connected(+ring,+ring))
Depth of variables	2
Maximum negatives	5
Maximum literals	4

Figure 7: Language constraints for *Progol*.

```
        number(X).

lteq(X,Y):-
        not(var(X)), not(var(Y)), !,
        number(X), number(Y),
        X =< Y.
lteq(X,X):-
        not(var(X)),
        number(X).
```

B.3 Theories obtained from *FOIL*

Background knowledge *B1*

```
Coverage: 41 positive examples, 0 negative examples
active(A) :-
        atm(A,B,C,27,E), atm(A,F,C,27,G),
        B \= F, not( bond(A,B,F,H)),
        not( bond(A,F,B,I)).

Coverage: 42 positive examples, 0 negative examples
active(A) :-
        atm(A,B,C,D,E), atm(A,F,C,29,H),
        D \= 29, H \= 0.006, H \= 0.002,
        E \= -0.125, E \= -0.113, E \= -0.116, H \= -0.007.

Coverage: 4 positive examples, 0 negative examples
active(A) :-  atm(A,B,C,D,0.118).

Coverage: 6 positive examples, 1 negative example
active(A) :-  atm(A,B,C,D,-0.116).

Coverage: 3 positive examples, 0 negative examples
active(A) :-  atm(A,B,C,D,-0.148).

Coverage: 2 positive examples, 0 negative examples
active(A) :-  atm(A,B,C,D,-0.117).

Time 4950.3 secs
```

Background knowledge *B2*

```
Coverage: 48 positive examples, 0 negative examples
active(A) :-
```

```
        atm(A,B,C,D,E),
        atm(A,F,C,G,H),
        E > -0.123,
        E =< -0.108,
        H > -0.113,
        H =< 0.013,
        D \= G,
        G \= 27.

Coverage: 39 positive examples, 0 negative examples
active(A) :-
        atm(A,B,C,D,E),
        atm(A,F,C,G,H),
        H > -0.12,
        H =< -0.108,
        atm(A,I,C,D,J),
        E =< 0.021,
        E> -0.111,
        J =< -0.085.

Coverage: 16 positive examples, 1 negative example
active(A) :-
        atm(A,B,C,D,E),
        E =< 0.025,
        E > 0.017.

Coverage: 67 positive examples, 0 negative examples
active(A) :-
        bond(A,B,C,D),
        atm(A,B,E,F,G),
        atm(A,C,H,I,J),
        atm(A,K,E,L,M),
        F \= L,
        J=< -0.069,
        J > -0.097,
        G> -0.012.

Coverage: 12 positive examples, 0 negative examples
active(A) :-
        bond(A,B,C,D),
        atm(A,B,E,F,G),
        atm(A,H,E,I,J),
        G =< -0.134,
        G > -0.167,
        J=< -0.14.
```

```
Coverage: 7 positive examples, 1 negative example
active(A) :-
        atm(A,B,C,D,E),
        atm(A,F,G,H,I),
        E> -0.115,
        E =< -0.114,
        I=< 0.145,
        I > -0.084.

Coverage: 8 positive examples, 1 negative example
active(A) :-
        atm(A,B,C,D,E),
        E =< -0.116,
        E > -0.118.

Time 9138.0 secs
```

Background knowledge *B3*

```
Coverage: 66 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), B<=-1.522, C<=6.79, C>3.26.

Coverage: 43 positive examples, 0 negative examples
active(A) :- lumo(A,B), B<=-2.142.

Coverage: 8 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), C>3, B>-1.451, B<=-1.329.

Coverage: 8 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), C>3.77, B<=-1.151, B>-1.276.

Coverage: 13 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), B<=-1.749, C<=2.58, C>1.89.

Coverage: 3 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), C<=3.06, C>3, B<=-1.861.

Coverage: 3 positive examples, 0 negative examples
active(A) :- lumo(A,B), B>-1.491, B<=-1.487.

Coverage: 4 positive examples, 1 negative example
active(A) :- lumo(A,B),  logp(A,C), B>-1.228, B<=-1.161, C<=3.06.

Coverage: 8 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), B>-1.474, C>4.49.
```

```
Coverage: 4 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), B<=-1.749, C<=1.84, C>0.47.

Time 0.5 secs
```

Background knowledge *B4*

```
Coverage: 66 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), B<=-1.522, C<=6.79, C>3.26.

Coverage: 43 positive examples, 0 negative examples
active(A) :- lumo(A,B), B<=-2.142.

Coverage: 8 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), C>3, B>-1.451, B<=-1.329.

Coverage: 19 positive examples, 0 negative examples
active(A) :-  carbon_6_ring(A,B).

Coverage: 14 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), C>2.03, B<=-1.102, C<=2.58.

Coverage: 8 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), not( methyl(A,D)), B>-1.266, C>3.77.

Coverage: 29 positive examples, 0 negative examples
active(A) :-  ball3(A,B).

Coverage: 8 positive examples, 1 negative example
active(A) :- lumo(A,B), B<=-1.102, B>-1.228, not( methyl(A,C)).

Coverage: 48 positive examples, 0 negative examples
active(A) :- lumo(A,B),  logp(A,C), B<=-1.304,  ring_size_5(A,D), C>1.59.

Coverage: 1 positive example, 0 negative examples
active(A) :- lumo(A,B), B<=-1.749,  hetero_aromatic_6_ring(A,C).

Time 0.5 secs
```

B.4 Theories obtained from *Progol*

Background knowledge *B1*

```
Coverage: 34 positive examples, 4 negative examples
active(A) :-
   atm(A,B,c,27,C), bond(A,D,E,1), bond(A,E,B,7).

Coverage: 20 positive examples, 3 negative examples
active(A) :-
   atm(A,B,c,27,C), bond(A,D,E,1), bond(A,F,D,1).

Coverage: 12 positive examples, 0 negative examples
active(A) :-
   atm(A,B,c,195,C).

Coverage: 8 positive examples, 1 negative examples
active(A) :-
   atm(A,B,h,3,0.144).

Coverage: 26 positive examples, 5 negative examples
active(A) :-
   atm(A,B,c,29,C), atm(A,D,c,22,E), bond(A,F,D,1).

Coverage: 3 positive examples, 0 negative examples
active(A) :-
   atm(A,B,c,22,-0.148).

Coverage: 14 positive examples, 2 negative examples
active(A) :-
   atm(A,B,h,3,0.146).

Coverage: 9 positive examples, 2 negative examples
active(A) :-
   atm(A,B,o,40,-0.383).

Coverage: 7 positive examples, 1 negative example
active(A) :-
   atm(A,B,h,3,0.149).

Time 117039 secs
```

Background knowledge *B2*

```
Coverage: 44 positive examples, 5 negative examples
active(A) :-
```

```
   atm(A,B,c,29,C), gteq(C,0.006).

Coverage: 60 positive examples, 2 negative examples
active(A) :-
   atm(A,B,c,27,C), lteq(C,-0.079).

Coverage: 14 positive examples, 2 negative examples
active(A) :-
   atm(A,B,h,3,0.146).

Coverage: 15 positive examples, 1 negative example
active(A) :-
   atm(A,B,o,40,C), lteq(C,-0.411).

Time 64256.281 secs
```

Background knowledge *B3*

```
Coverage: 66 positive examples, 5 negative examples
active(A) :-
   atm(A,B,c,27,C), logp(A,D), gteq(D,2.580).

Coverage: 51 positive examples, 4 negative examples
active(A) :-
   atm(A,B,c,29,C), lumo(A,D), lteq(D,-1.178).

Coverage: 56 positive examples, 4 negative examples
active(A) :-
   lumo(A,B), lteq(B,-1.937).

Time 41787.922
```

Background knowledge *B4*

```
Coverage: 68 positive examples, 5 negative examples
active(A) :-
   logp(A,B), gteq(B,4.180).

Coverage: 56 positive examples, 4 negative examples
active(A) :-
   lumo(A,B), lteq(B,-1.937).

Coverage: 48 positive examples, 1 negative examples
active(A) :-
```

```
logp(A,B), gteq(B,2.740), ring_size_5(A,C).

Time 40569.875
```

C Analysis tools used for Figures 3 and 4

For reference, we provide a brief description of the analysis tools used to compute entries in these tables. More information can be found in any standard statistical textbook on analysing contingency tables (for example, [3]).

Accuracy estimates in Figure 3 are obtained as follows. For each algorithm, a 2×2 tabulation of actual values against those predicted by an algorithm is obtained as in Figure 8.

		Actual: Active	Actual: Inactive	
Predicted	Active	n_1 (e_1)	n_2 (e_2)	n_a
	Inactive	n_3 (e_3)	n_4 (e_4)	n_b
		n_c	n_d	N

Figure 8: A contingency table for estimating predictive accuracy of an algorithm. n_1 is the number of compounds known to be active, and predicted as active by the algorithm. Similarly for the entries $n_{2,3,4}$. The number e_1 is the expected value of for the Active/Active cell, under the hypothesis that the actual class is independent of the predicted one. It is calculated from the partial totals as $e_1 = (n_a n_c)/N$. Similarly for the entries $e_{2,3,4}$.

Before estimating the predictive accuracy, the first question to answer is whether there is any association between actual and predicted values. This is adequately catered for by obtaining the χ^2 value for the table. The relevant formula for this is as follows:

$$\chi^2 = \sum_{i=1}^{4} \frac{(n_i - e_i)^2}{e_i}$$

A standard correction for non-continuous numbers (referred to as Yates' correction) is introduced by replacing the numbers $n_{1,\ldots,4}$ by numbers which are 0.5 of a unit less when they exceed the expected value, and 0.5 of a unit more when they lie below the expected value. The χ^2 probability is obtained from standard tables with $\nu = 1$ degrees of freedom.

		Predicted (A_1): Correct	Predicted (A_1): Incorrect	
Predicted (A_2)	Correct	n_1	n_2	n_a
	Incorrect	n_3	n_4	n_b
		n_c	n_d	N

Figure 9: Cross-comparison of the predictions of a pair of algorithms $A_{1,2}$ n_1 is the number of compounds whose class is correctly predicted by both algorithms. Similarly for the entries $n_{2,3,4}$.

A high χ^2 value indicates a low probability of a chance association between predicted and actual values. In this study, we have stipulated that this probability is no more than 0.05. That is, χ^2 values must be at least 3.84. The predictive accuracy for theories that satisfy this constraint is then estimated as $p = (n_1 + n_4)/N$. The error in this estimate is $\pm\sqrt{pq/N}$ where $q = 1 - p$.

McNemar's test for changes is used to obtain the significance results in Figure 4. For a pair of algorithms, this is done by cross-comparison the compounds correctly and incorrectly classified as shown in Figure 9. The null hypothesis is that the proportions of examples correctly classified by both algorithms is the same. If there is no significant difference in the performance of the two algorithms, half of the $n_2 + n_3$ cases whose classifications disagree should be classified correctly by A_1 and A_2 respectively. Because of small numbers, we directly estimate the probability of a chance classification of n_2 and n_3, using the binomial distribution.