Appears in *Proceedings of the 12th European Conference on Machine Learning,*
pp. 466-477, Freiburg, Germany, September 2001

Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing

Lappoon R. Tang and Raymond J. Mooney

Department of Computer Sciences
University of Texas
2.124 Taylor Hall
Austin, TX 78712
{rupert, mooney}@cs.utexas.edu

Abstract. In this paper, we explored a learning approach which combines different learning methods in inductive logic programming (ILP) to allow a learner to produce more expressive hypotheses than that of each individual learner. Such a learning approach may be useful when the performance of the task depends on solving a large amount of classification problems and each has its own characteristics which may or may not fit a particular learning method. The task of semantic parser acquisition in two different domains was attempted and preliminary results demonstrated that such an approach is promising.

1 Motivation

While a significant portion of machine learning research has devoted to tackling tasks that involve solving a single classification problem, some domains require solving a large sequence of classification problems in order to perform the task successfully. For instance, in learning control rules for semantic parsing using inductive logic programming [1], one needs to specialize a parser by inducing control rules for a large set of parsing operators. However, each induction problem has its own characteristics. The performance hit on the task could be significant if a single (ILP) learner performs poorly on some of these problems because its language bias is "inappropriate" for them. Therefore, using a mixture of language biases might be beneficial (if not sacrificing too much computational efficiency).

A typical ILP algorithm can be viewed as a loop in which a certain *clause constructor* is embedded. A clause constructor is formally defined here as a function $f : T \times B \times E \rightarrow S$ such that given the current building theory T, a set of training examples E, and the set of background knowledge B, it produces a set of clauses S. For example, to construct a clause using FOIL [2] given an existing partial theory T_p (which is initially empty) and a set of training examples $\xi^+ \cup \xi^-$ (positive and negative), one uses all the positive examples not covered by T_p to learn a single clause C. So, we have $f_{Foil}(T_p, B, \xi^+ \cup \xi^-) = f_{Foil}(T_p, B, \{e \in \xi^+ \mid$

$T_p \not\models e\} \cup \xi^-) = \{C\}$. Notice that f_{Foil} always produces a singleton set. Since different constructors create clauses of different characteristics (like syntax and accuracy), a learner using multiple clause constructors could exploit the various language biases available to produce more expressive hypotheses. We want to examine the potential benefit of this approach on learning semantic parsers.

Section 2 reviews two ILP algorithms we used in our new approach which is described in Section 3. Section 4 presents the task of learning semantic parsers as our experimental domain. Experimental results are presented in Section 5 followed by conlusions in Section 6.

2 Background

2.1 An Overview of CHILLIN

CHILLIN [3] was the first ILP algorithm that has been applied to the task of learning control rules for a semantic parser in a system called CHILL [1]. It has a compaction outer loop that builds a more general hypothesis with each iteration. In each iteration, a clause is built by finding the least general generalization (LGG) under θ-subsumption of a random pair of *clauses* in the building definition *DEF* and is specialized by adding literals to its body like FOIL. The clause with the most compaction is returned. The compaction loop is as follows:

$DEF := \{e \leftarrow true \mid e \in \xi^+\}$
Repeat
  $PAIRS :=$ a sampling of pairs of clauses from DEF
  $GENS := \{G \mid G = Find\_A\_Clause(C_i, C_j, DEF, \xi^+, \xi^-)$ for $\langle C_i, C_j\rangle \in PAIRS\}$
  $G :=$ the clause in $GENS$ yielding most compaction
  $DEF := (DEF -$ (clauses empirically subsumed by G)) $\cup \{G\}$
Until no-further-compaction
Return DEF

Once a clause is found, it is added to the current theory *DEF*. Any other clause *empirically* subsumed by it will be removed. A clause C empirically subsumes D if all (finitely many) positive examples covered by D are covered by C given the same set of background knowledge. If a clause cannot be refined using the given set of background knowledge, it will attempt to invent a predicate using a method similar to CHAMP [4]. Now, let's define the clause constructor $f_{Chillin}$ for CHILLIN. (Strictly speaking, $f_{Chillin}$ is not a function because of algorithmic randomness. To make it a function, one has to include an additional argument specifying the state of the system. For simplicity, we just omit it here and assume it behaves like a function.) Given a current partial theory T_p (initially empty), background knowledge B, and $\xi^+ \cup \xi^-$ as inputs, $f_{Chillin}$ takes $T_p \cup \{e \in \xi^+ \mid T_p \not\models e\}$ to form the initial *DEF*. A clause G with the best coverage is then learned by going through the compaction loop for one step. So, $f_{Chillin}(T_p, B, \xi^+ \cup \xi^-) = \{G\}$. However, we are going to allow $f_{Chillin}$ to return the best n clauses in *GENS* by coverage instead and use this more relaxed version of $f_{Chillin}$ in our algorithm.

2.2 An Overview of mFOIL

Like FOIL, mFOIL is a top-down ILP algorithm. However, it uses a more direct accuracy estimate, the m-estimate [5], to measure the expected accuracy of a clause which is defined as

$$accuracy(C) = \frac{s + m \cdot p^+}{n + m} \tag{1}$$

where C is a clause, s is the number of positive examples covered by the clause, n is the total number of examples covered, p^+ is the prior probability of the class $\oplus$, and m is a parameter.

mFOIL was designed with handling imperfect data in mind. It uses a pre-pruning algorithm which checks if a refinement of a clause can be *possibly* significant. If so, it is retained in the search. The significant test is based on the likelihood ratio statistic. Suppose a clause covers n examples and s of which are positive examples, the value of the statistic is calculated as follows:

$$Likelihood\ Ratio = 2n(q^+ \log \frac{q^+}{p^+} + q^- \log \frac{q^-}{p^-}) \tag{2}$$

where p^+ and p^- are the prior probabilities of the class $\oplus$ and $\ominus$ respectively, $q^+ = \frac{s}{n}$, and $q^- = 1 - q^+$. This is distributed approximately as χ^2 with 1 degree of freedom. If the estimated value of a clause is above a particular threshold, it is considered significant. A clause, therefore, cannot be possibly significant if the upper bound $-2s \log p^+$ is already less than the threshold and will not be further refined.

The search starts with the most general clause. Literals are added successively to the body of a clause. A beam of promising clauses are maintained, however, to partially overcome local minima. The search stops when no clauses in the beam can be significantly refined and the most significant one is returned. So, given the current building theory T_p, background knowledge B, training examples $\xi^+ \cup \xi^-$, $f_{mFoil}(T_p, B, \xi^+ \cup \xi^-) = \{C\}$ where C is the most significant clause found in the search beam. Again, we use a modified version of f_{mFoil} which returns the entire beam of promising clauses when none of them can be significantly refined.

3 Using Multiple Clause Constructors in COCKTAIL

The use of multi-strategy learning to exploit diversity in hypothesis space and search strategy is not novel [15]. However, our focus here is applying a similar idea specifically in ILP where different learning strategies are integrated in a unifying hypothesis evaluation framework.

A set of clause constructors (like FOIL's or GOLEM's) have to be chosen in advance. The decision of what constitutes a sufficiently rich set of constructors depends on the application one needs to build. Although an arbitrary number of clause constructors is permitted (in principle), in practice one should use only a handful of useful constructors to reduce the complexity of the search as much as

Procedure COCKTAIL

Input:

ξ^+, ξ^-:	the $\oplus$ and $\ominus$ examples respectively
F:	a set of clause constructors
B:	a set of sets of background knowledge for each clause constructor in F
M:	the metric for evaluating a theory

Output:

T:	the learned theory

$T := \{\}$

Repeat

$\quad Clauses := \bigcup_{f_i \in F, B_i \in B} f_i(T, B_i, \xi^+ \cup \xi^-)$

$\quad$ Choose $C \in Clauses$ such that $M(T - \{\text{clauses empirically subsumed by } C\} \cup \{C\},$ $\xi^+ \cup \xi^-)$ is the best

$\quad T := T - \{\text{clauses empirically subsumed by } C\} \cup \{C\}$

Until $M(T, \xi^+ \cup \xi^-)$ does not improve

Return T

End Procedure

Fig. 1. Outline of the COCKTAIL Algorithm

possible. We have chosen mFOIL's and CHILLIN's clause constructors primarily because of their inherent differences in language bias and the relative ease to modify them to return a set of clauses of a given size.

The search of the hypothesis space starts with the empty theory. At each step, a set of potential clauses is produced by collecting all the clauses constructed using the different clause constructors available. Each clause found is then used to *compact* the current building theory to produce a set of new theories; existing clauses in the theory that are empirically subsumed by the new clause are removed. The best one is then chosen according to the given theory evaluation metric and the search stops when the metric score does not improve. The algorithm is outlined in Figure 1.

As the "ideal" solution to an induction problem is the hypothesis that has the minimum size and the most predictive power, some form of bias leading the search to discover such hypotheses would be desirable. It has been formulated in the *Minimum Description Length (MDL) principle* [6] that the most probable hypothesis H given the evidence (training data) D is the one that minimizes the complexity of H given D which is defined as

$$K(H \mid D) = K(H) + K(D \mid H) - K(D) + c \qquad (3)$$

where $K(\cdot)$ is the Kolmogorov complexity function and c is a constant. This is also called the *ideal* form of the MDL principle. In practice, one would instead find an H of some set of hypotheses that minimizes $L(H) + L(D \mid H)$ where $L(x) = -\log_2 Pr(x)$ and interpret $L(x)$ as the corresponding Shannon-Fano (or Huffman) codeword length of x. However, if one is concerned with just the *ordering* of hypotheses but not *coding* or *decoding* them, it seems reasonable to use a metric that gives a rough estimate instead of computing the complexity directly using the encoding itself as it would be computationally more efficient.

Now, let $S(H \mid D)$ be our estimation of the complexity of H given D which is defined as

$$S(H \mid D) = S(H) + S(D \mid H) - S(D) \tag{4}$$

where $S(H)$ is the estimated prior complexity of H and

$$\begin{aligned} S(D \mid H) = \; & S(\{e \leftarrow true \mid e \in \xi^{+} \text{ and } H \not\models e\}) + \\ & S(\{false \leftarrow e \mid e \in \xi^{-} \text{ and } H \models e\}) \end{aligned} \tag{5}$$

is the estimated complexity of D given H. This is roughly a worst case estimate of the complexity of a program that computes the set D given H. A much better scheme would be to compute $S(H_1 \cup \{T \leftarrow T', not\ T''\} \cup H_2)$ instead where H_1 and H_2 are some (compressive) hypotheses consistent with the uncovered positive examples of H and covered negative examples of H respectively, T is the target concept $t(R_1, \cdots, R_k)$ that we need to learn, $T' = t'(R_1, \cdots, R_k)$ and $T'' = t''(R_1, \cdots, R_k)$ are the renaming of the target concept. (All predicates t/k appearing in any clause in H and H_2 have to be renamed to t'/k and t''/k respectively.) Computing H_1 and H_2 could be problematic, however, and thus we simply take the worst case assuming the discrepancy between H and D is not compressible. A very simple measure is employed here as our complexity estimate [8]. The size S of a set of $Clauses$ (or a hypothesis) where each clause C with a $Head$ and a $Body$ is defined as follows:

$$S(Clauses) = \sum_{C \in Clauses} 1 + termsize(Head) + termsize(Body) \tag{6}$$

where

$$termsize(T) = \begin{cases} 1 & \text{if } T \text{ is a variable} \\ 2 & \text{if } T \text{ is a constant} \\ 2 + \sum_{i=1}^{arity(T)} termsize(arg_i(T)) & otherwise. \end{cases} \tag{7}$$

The size of a hypothesis can be viewed as a sum of the average number of bits required to encode a symbol appearing in it which can be a variable, a constant, a function symbol, or a predicate symbol, plus one bit of encoding each clause terminator. (Note that this particular scheme gives less weight to variable encoding.) Finally, our theory evaluation metric is defined as

$$M(H, D) = S(H) + S(D \mid H). \tag{8}$$

The goal of the search is to find the H that minimizes the metric M. The metric is purely syntactic; it does not take into account the complexity of proving an instance [14]. However, we are relying on the assumption that syntactic complexity implies computational complexity although this and the reverse are not true in general. So, the current metric does not gaurantee finding the hypothesis with the shortest proof of the instances.

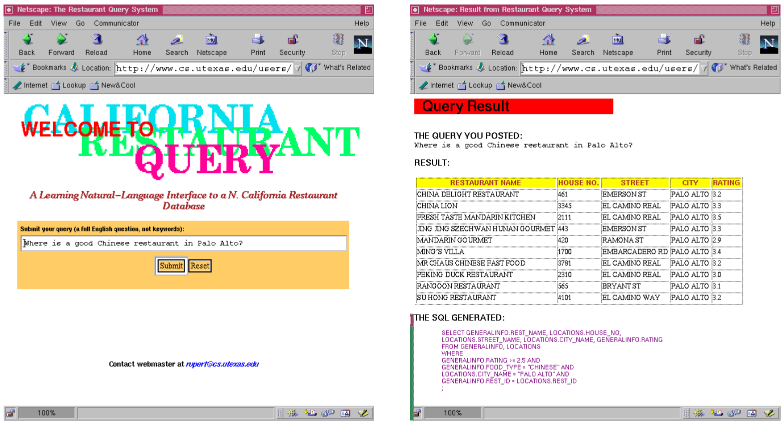

Fig. 2. Screenshots of a Learned Web-based NL Database Interface

4 Learning to Parse Questions into Logical Queries

Being able to query a database using natural languages has been an interesting task since the 60's as most users of the database may not know the underlying database access language. The need for such applications is even more pronounced with the rapid development of the Internet which has become an important channel for information delivery. Screenshots of a natural language interface (NLI) we developed are shown in Figure 2.

Traditional (*rationalist*) approaches to constructing database interfaces require an expert to hand-craft an appropriate semantic parser [9]. However, such hand-crafted parsers are time consuming to develop and suffer from problems with robustness and incompleteness. Nevertheless, very little research in empirical NLP has explored the task of automatically acquiring such interfaces from annotated training examples. The only exceptions of which we are aware are a statistical approach to mapping airline-information queries into SQL presented in [10], a probabilistic decision-tree method for the same task described in [11], and an approach using inductive logic programming to learn a logic-based semantic parser described in [1].

We are going to briefly review our overall approach using an interface developed for a U.S. Geography database (Geoquery) as a sample application [1] which is available on the Web at `http://www.cs.utexas.edu/users/ml/geo.html`.

4.1 Semantic Representation

First-order logic is used as a semantic representation language. Chill has also been applied to a restaurant database in which the logical form resembles SQL, and is translated automatically into SQL (see Figure 2). We explain the features of the Geoquery representation language through a sample query:

Input: "What is the largest city in Texas?"
Query: answer(C,largest(C,(city(C),loc(C,S),const(S,stateid(texas))))).

Objects are represented as logical terms and are typed with a semantic category using logical functions applied to possibly ambiguous English constants (e.g. stateid(Mississippi), riverid(Mississippi)). Relationships between objects are expressed using predicates; for instance, loc(X,Y) states that X is located in Y.

We also need to handle quantifiers such as 'largest'. We represent these using *meta-predicates* for which at least one argument is a conjunction of literals. For example, largest(X, Goal) states that the object X satisfies Goal and is the largest object that does so, using the appropriate measure of size for objects of its type (e.g. area for states, population for cities). Finally, an unspecified object required as an argument to a predicate can appear elsewhere in the sentence, requiring the use of the predicate const(X,C) to bind the variable X to the constant C.

4.2 Parsing Actions

Our semantic parser employs a shift-reduce architecture that maintains a stack of previously built semantic constituents and a buffer of remaining words in the input. The parsing actions are automatically generated from templates given the training data. The templates are INTRODUCE, COREF_VARS, DROP_CONJ, LIFT_CONJ, and SHIFT. INTRODUCE pushes a predicate onto the stack based on a word appearing in the input and information about its possible meanings in the lexicon. COREF_VARS binds two arguments of two different predicates on the stack. DROP_CONJ (or LIFT_CONJ) takes a predicate on the stack and puts it into one of the arguments of a meta-predicate on the stack. SHIFT simply pushes a word from the input buffer onto the stack. The parsing actions are tried in exactly this order. The parser also requires a lexicon to map phrases in the input into specific predicates. This lexicon can also be learned automatically from the training data [12].

Let's go through a simple trace of parsing the request "What is the capital of Texas?" A lexicon that maps 'capital' to 'capital(_)', 'of' to 'loc(_,_)', and 'Texas' to 'const(_,stateid(texas))' suffices here. Interrogatives like "what" may be mapped to predicates in the lexicon if necessary. The parser begins with an initial stack and a buffer holding the input sentence. Each predicate on the parse stack has an attached buffer to hold the context in which it was introduced; words from the input sentence are shifted onto this buffer during parsing. The initial parse state is shown below:

Parse Stack: [answer(_,_):[]]
Input Buffer: [what,is,the,capital,of,texas,?]

Since the first three words in the input buffer do not map to any predicates, three SHIFT actions are performed. The next is an INTRODUCE as 'capital' is at the head of the input buffer:

Parse Stack: [capital(_):[], answer(_,_):[the,is,what]]
Input Buffer: [capital,of,texas,?]

The next action is a COREF_VARS that binds the argument of capital(_) with the first argument of answer(_,_).

Parse Stack: [capital(C):[], answer(C,_):[the,is,what]]
Input Buffer: [capital,of,texas,?]

The next sequence of steps are a SHIFT followed by an INTRODUCE

Parse Stack: [loc(_,_):[], capital(C):[capital], answer(C,_):[the,is,what]]
Input Buffer: [of,texas,?]

The next sequence of steps are a COREF_VARS, a SHIFT, an INTRODUCE, and then a COREF_VARS:

Parse Stack: [const(S,stateid(texas)):[], loc(C,S):[of],capital(C):[capital], answer(C,_):[the,is,what]]
Input Buffer: [texas,?]

The last four steps are three DROP_CONJ's followed by two SHIFT's:

Parse Stack: [answer(C, (capital(C),loc(C,S),const(S,stateid(texas)))):[the,is,what]]
Input Buffer: []

This is the final state and the logical query is extracted from the stack.

4.3 Learning Control Rules

The initially constructed parser has no constraints on when to apply actions, and is therefore overly general and generates numerous spurious parses. Positive and negative examples are collected for each action by parsing each training example and recording the parse states encountered. Parse states to which an action *should* be applied (i.e. the action leads to building the correct semantic representation) are labeled positive examples for that action. Otherwise, a parse state is labeled a negative example for an action if it is a positive example for another action below the current one in the *ordered* list of actions. Control conditions which decide the correct action for a given parse state are learned for each action from these positive and negative examples.

5 Experimental Results

5.1 The Domains

Two different domains are used for experimentation here. The first one is the United States Geography domain. The database contains about 800 facts implemented in Prolog as relational tables containing basic information about the U.S. states like population, area, capital city, neighboring states, and so on. The second domain consists of a set of 1000 computer-related job postings, such as job announcements, from the USENET newsgroup `austin.jobs`. Information from these job postings are extracted to create a database which contains the following types of information: 1) the job title, 2) the company, 3) the recruiter, 4) the location, 5) the salary, 6) the languages and platforms used, and 7) required or desired years of experience and degrees [13].

Parsers learned with \ Corpora	Geo1000				Jobs640			
	R	P	S	T	R	P	S	T
COCKTAIL (f_{mFoil}+$f_{Chillin}$)	79.40	89.92	64.79	62.88	79.84	93.25	105.77	68.10
COCKTAIL (f_{mFoil} only)	75.10	88.98	127.61	76.67	63.75	82.26	427.02	66.64
COCKTAIL ($f_{Chillin}$ only)	70.80	91.38	150.69	41.24	72.50	86.24	177.99	43.81
CHILLIN	71.00	90.79	142.41	38.24	74.22	87.48	175.94	45.31
mFOIL	67.50	87.10	204.62	65.17	58.91	82.68	561.34	69.08

Table 1. Results on all the experiments performed. Geo1000 consists of 1000 sentences from the U.S. Geography domain. Jobs640 consists of 640 sentences from the job postings domain. R = recall, P = precision, S = average size of a hypothesis found for each induction problem when learning a parser using the entire corpus, and T = average training time in mins.

5.2 Experimental Design

The U.S. Geography domain has a corpus of 1000 sentences collected from undergraduate students in our department and from real users of our Web interface. The job database information system has a corpus of 640 sentences; 400 of which are artificially made using a simple grammar that generates certain obvious types of questions people may ask and the other 240 are questions obtained from real users of our interface. Both corpora are available at `ftp://ftp.cs.utexas.edu/pub/mooney/nl-ilp-data/`.

The experiments were conducted using 10-fold cross validation. In each test, the recall (a.k.a. accuracy) and the precision of the parser are reported. Recall and precision are defined as

$$Recall = \frac{\#\text{ of correct queries produced}}{\#\text{ of sentences}} \tag{9}$$

$$Precision = \frac{\#\text{ of correct queries produced}}{\#\text{ of successful parses}}. \tag{10}$$

The recall is the number of correct queries produced divided by the total number of sentences in the test set. The precision is the number of correct queries produced divided by the number of sentences in the test set from which the parser produced a query (i.e. a successful parse). A query is considered correct if it produces the same answer set as that of the correct logical query.

5.3 Discussion of Results

For all the experiments performed, we used a beam size of four for mFOIL (and therefore for f_{mFoil}), a significant threshold of 6.64 (i.e. 99% level of significance), and a parameter $m = 10$. We took the best four clauses (by coverage) found by CHILLIN. COCKTAIL using both the mFOIL's and CHILLIN's clause constructors performed the best; it outperformed all other learners by at least 4% in recall in either domains. In addition, COCKTAIL using only f_{mFoil} performed better than using only $f_{Chillin}$ in the Geography domain while the latter performed better in the job postings domain. This indicates that there are some inherent differences between the two domains as far as language bias is concerned. This will be further explained below. Notice that CHILLIN alone performed slightly better than COCKTAIL using only $f_{Chillin}$. There must be other factors in the picture we were not aware of as using a hill-climbing search actually performed better in this case. One possibility might be that the current MDL metric has problems handling complicated terms with lots of constants which could result in choosing overly specific clauses (if they are in the beam) and therefore learning a larger number of clauses for the building theory. Perhaps somewhat surprising is the result obtained from using the original mFOIL algorithm; the poor results seem to suggest that choosing the most statistically significant clause (in the search beam) does not necessarily produce the most compressive hypothesis. Apparently, this is due to the fact that some compressive clauses were wrongly rejected by a statistical based measure [14].

COCKTAIL found the most compressive hypothesis on average for an induction problem when learning a parser using the entire corpus. (There were 138 induction problems for Geo1000 and 87 for Jobs640.) Before we proceed to explain the results, let's go through an example to see how f_{mFoil} and $f_{Chillin}$ construct clauses of very different language biases which are good for expressing different types of features present in a set of training examples. Suppose we want to learn a concept class of lists of atoms and tuples of atoms and we have positive examples $\xi^+ = \{t([a,[e,c],b]), t([c,[a,b],a])\}$ and negative examples $\xi^- = \{t([[e,c],b]), t([b,c,[a,b]]), t([c,b,c]), t([d,[e,c],b,b])\}$. One possible hypothesis H_1 consistent with the data would be

$$t(X) \leftarrow member(a, X). \tag{11}$$

which states that any list containing the constant a is in the concept. Another possible hypothesis H_2 would be

$$t([W,[X,Y],Z]) \leftarrow true. \tag{12}$$

which asserts that the concept class contains lists of three elements and the second element has to be a tuple. Both H_1 and H_2 are qualitatively different in the sense that they look for different *types* of features for classification; the former looks for the presence of a certain specific *element* that might appear anywhere in a given list while the latter looks for a specific *structure* of a list.

This is exactly what is happening here. mFOIL and CHILLIN learn very different features for classification; mFOIL is given background predicates which check the presence (or absence) of a particular element in a given parse state (e.g. a certain predicate or a certain word phrase) while CHILLIN is not given any such background predicates but it learns the structural features of a parse state through finding *LGGs* with good coverage (and inventing predicates if necessary). Each learner is effective in expressing each type of feature using its own language bias; if one were to learn structural features of a parse state using mFoil's language bias (e.g. not allowing function terms), the hypothesis thus expressed would have a very high complexity and vice versa.

When inspecting the set of control rules learned by Cocktail, we discovered that on small induction problems involving only a few training examples, only one type (either mFoil's or Chillin's) of clause constructors was sufficient; the resulting hypothesis contained clauses built only by one of them. However, different (similarly small) problems require different constructors. On larger induction problems involving a few hundred examples, Cocktail learned hypotheses with clauses constructed by both f_{mFoil} and $f_{Chillin}$. This suggests that some problems do require inspecting structural features of a parse state and examining its elements; using a variety of language biases allows the learner to discover and effectively express these different features in a problem which is illustrated by the fact that the average size of a hypothesis found is minimal (as irrelevant features tend to make complicated hypotheses), at least in the domains attempted. The training time of Cocktail using all the given clause constructors in each domain was much less than the sum of using each of them alone (unlike what one might expect) and in fact in some case closer to the average time.

6 Conclusion

An ILP learning approach which employs different clause constructors from different ILP learners is discussed. It was applied to the task of semantic parser acquisition and was demonstrated to perform better than using a single learner. Future work could explore using a larger set of clause constructors and examine their effects on language learning problems. Since problems that require solving a large sequence of induction problems do not seem to occur very often, we would like to see if the existing approach is applicable to domains where only a single or a relatively small number of classification prolems are involved. It is also interesting to use a more precise theory evaluation metric like the one described by Srinivasan et. al. [7].

7 Acknowledgements

This research was supported by the National Science Foundation under grant IRI-9704943. Special thanks go to Sugato Basu for proofreading an earlier draft of this paper.

References

1. John M. Zelle and Raymond J. Mooney: Learning to Parse Database Queries Using Inductive Logic Programming. Proceedings of the Thirteenth National Conference on Artificial Intelligence (1996) 1050–1055
2. J. Ross Quinlan: Learning Logical Definitions from Relations. Machine Learning **5** (1990) 239–266

3. John M. Zelle and Raymond J. Mooney: Combining Top-Down and Bottom-Up Methods in Inductive Logic Programming. Proceedings of the Eleventh International Conference on Machine Learning (1994) 343–351
4. Boonserm Kijsirikul and Masayuki Numao and Masamichi Shimura: Discrimination-Based Constructive Induction of Logic Programs. Proceedings of the Tenth National Conference on Artificial Intelligence (1992) 44–49
5. Bojan Cestnik: Estimating probabilities: A crucial task in machine learning. Proceedings of the Ninth European Conference on Artificial Intelligence (1990) 147–149
6. Jorma Rissanen: Modeling by Shortest Data Description. Automatica **14** (1978) 465–471
7. Ashwin Srinivasan, Stephen Muggleton, Michael Bain: The Justification of Logical Theories based on Data Compression. Machine Intelligence **13** (1994) 87–121
8. Stephen Muggleton and W. Buntine: Machine Invention of First-order Predicates by Inverting Resolution. Proceedings of the Fifth International Conference on Machine Learning (1988) 339-352
9. G. G. Hendrix and E. Sacerdoti and D. Sagalowicz and J. Slocum: Developing a Natural Language Interface to Complex Data. ACM Transactions on Database Systems **3** (1978) 105–147
10. Scott Miller and David Stallard and Robert Bobrow and Richard Schwartz: A Fully Statistical Approach to Natural Language Interfaces. Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (1996) 55–61
11. Roland Kuhn and Renato De Mori: The Application of Semantic Classification Trees to Natural Language Understanding. IEEE Transactions on Pattern Analysis and Machine Intelligence **17** (1995) 449–460
12. Cynthia A. Thompson and Raymond J. Mooney: Automatic Construction of Semantic Lexicons for Learning Natural Language Interfaces. Proceedings of the Sixteenth National Conference on Artificial Intelligence (1999) 487–493
13. Mary Elaine Califf and Raymond J. Mooney: Relational Learning of Pattern-Match Rules for Information Extraction. Proceedings of the Sixteenth National Conference on Artificial Intelligence (1999) 328–334
14. S. Muggleton, A. Srinivasan, and M. Bain: Compression, significance and accuracy. Proceedings of the Ninth International Machine Learning Conference (1992) 338-347
15. Attilio Giordana, Filippo Neri, Lorenza Saitta, and Marco Botta: Integrating Multiple Learning Strategies in First Order Logics. Machine Learning **27**(3): 209-240 (1997)