Efficient and Effective Induction of First Order Decision Lists

Mary Elaine Califf

Department of Applied Computer Science, Campus Box 5150, Illinois State University, Normal, IL 61790 USA
mecalif@ilstu.edu

Abstract. We present BUFOIDL, a new bottom-up algorithm for learning first order decision lists. Although first order decision lists have potential as a representation for learning concepts that include exceptions, such as language constructs, previous systems suffered from limitations that we seek to overcome in BUFOIDL. We present experiments comparing BUFOIDL to previous work in the area, demonstrating the system's potential.

1 Introduction

In machine learning, there are a variety of metrics that can be applied to algorithms. An important measure of any learning algorithm is, of course, predictive accuracy. However, the degree to which the representation used fits the problem and the comprehensibility of the resulting set of rules can also have a significant impact on the usefulness of a system.

Our primary area of interest is in applying machine learning algorithms to language problems. Here, much of the successful work has been statistical in nature.

However, statistical methods are limited in their expressiveness and produce rules that are not easy for humans to understand. Therefore, we believe that work in language learning that uses more expressive representations that are more amenable to human understanding is desirable.

One rule representation that seems to be a very good fit for language learning is that of decision lists. Decision lists are particularly good for representing concepts with general rules that have exceptions. In previous work, we showed that *first-order decision lists* are a highly effective representation for learning a morphological concept (specifically generating the past tense form of a verb given the base form) [8]. However, the FOIDL system has one major flaw: efficiency. FOIDL is very effective when run on a fairly small example set and produces a very understandable set of rules. However, it is a top-down algorithm that quickly blows up in the face of large numbers of examples and a large search space of constants, as will be demonstrated below. This property limits the practical usefulness of the algorithm, since language learning generally involves a large search space and a large number of examples.

This issue has been previously addressed by Manandhar, Džeroski, and Erjavec [7], who recognized the potential of FOIDL's representation and attempted to address the efficiency problems of the algorithm in their CLOG system. CLOG is a very fast

algorithm. However, it has its own set of flaws, particularly in terms of generality and accuracy, as will be discussed further below.
We have developed BUFOIDL, a system that addresses the issues of effective and efficient induction of first order decision lists, providing accuracy comparable to that of FOIDL and time complexity that (although not rivaling CLOG's speed) is far more acceptable than that of FOIDL.
The remainder of this paper is organized as follows: Section 2 presents background material on first order decision lists, FOIDL, and CLOG. Section 3 presents the BUFOIDL algorithm. Section 4 presents experimental results comparing the three systems. Section 5 briefly discusses other relevant work. The final section concludes and presents some directions for future work.

2 Background

In this section, we explain what we mean by first-order decision lists and briefly describe the two systems that inspired our work: FOIDL and CLOG.

2.1 First-Order Decision Lists

First-order decision lists are ordered sets of clauses, each of which ends in a cut. Thus, as the clauses are tried in order, only the answer produced by the first matching clause is obtained. This is basically a first-order extension of prepositional decision lists [13], where each rule consists of a set of tests and a category label, and an example is assigned to the category label for the first rule such that the example meets the tests. Initial work in this area learned rules in the order in which they are applied [13,6], but Webb and Brkič [14] pointed out the advantages of learning the rules in reverse order, since more general rules tend to be learned first given most preference functions. FOIDL takes this approach, and we follow it as well.

2.2 The FOIDL Algorithm

FOIDL is a top-down inductive logic programming algorithm based largely on FOIL [11]. To FOIL, FOIDL adds three primary features:

- the ability to handle intensionally defined background predicates,
- the ability to handle implicit negative using a concept of output completeness,
- and the ability to learn first-order decision lists.

The first of these is fairly straightforward, but is one of the causes of time spent by the algorithm. FOIDL must actually execute each rule formed for each example because of the intensional background.
The second distinguishing feature of FOIDL is its use of an output completeness assumption to handle implicit negative examples. In order to make use of this assumption, the predicate to be learned must have a *mode* associated with it, indicating which arguments are input(s) and which are output(s). Then the

assumption may be made that for any unique input pattern for which a positive example is provided, all other positive examples with that input pattern are also in the training set. Clearly, this is the case for any predicate that represents a function with a unique output for each input.

Once we have this assumption, we can quantify the number of implicit negative examples covered by a clause. For each example, we produce an *output query* that specifies the inputs. If the clause applied to the output query produces a ground answer that does not match a correct output for the input, it covers a single negative for that example. If it produces a non-ground answer, FOIDL estimates the number of examples covered based on a parameter indicating the size of the universe from which an answer might be taken.

The output completeness assumption is not specific to the induction of first-order decision lists and has been used in learning relations that are not functions [4,5]. Note that it is less restrictive than the closed world assumption in the sense that the system must be guaranteed only all correct outputs for each input actually present in a given data set. For further discussion of the advantages of the output completeness assumption, see [4].

The third distinguishing feature of FOIDL is its ability to learn first-order decision lists. It does this by first learning a clause that covers as many positive examples as possible. Note that this clause may produce incorrect answers for examples that have not yet been covered. For the case of producing the past tense in English, this might be a rule such as:

```
past(A,B) :- split(B,A,[e,d]).
```

where *split* is a predicate whose second and third arguments are non-empty lists that produce the first argument when appended together.

In developing subsequent rules, clauses are constrained to not cover previously covered examples. So the next rule to be learned might be:

```
past(A,B):- split(B,A,[d],
      split(A,C,[e]).
```

where the literal that constrains *A* to end in *e* is required to keep this special case rule from firing for examples that should be covered by the default rule.

The basic algorithm for learning a clause in FOIDL is explained in Figure 1.

This is complicated slightly be the idea of exceptions to exceptions. For example, consider the case of a verb ending in *y* in English. To produce the past tense, you typically remove the *y* and add *ied*. However, to learn this rule requires covering some examples that end in *y* but still add *ed* to produce the past tense (consider *delay*). To accommodate this, FOIDL will "uncover" previously covered positive examples (adding them to *positives-to-cover*) if doing so allows the system to learn a rule that covers a sufficient number of positive examples.

```
Initialize C to R(V1,V2,…Vk) where R is the target
        predicate with arity k
Initialize T to contain all examples in positives-to-
        cover and output queries for all positive
        examples
While T contains output queries
   Find the best literal L to add to the clause
   Let T' be the subset of positive examples in T whose
           output query still produces a first answer
           that unifies with the correct answer, plus
           the output queries in T that either
      1)Produce a non-ground first answer that unifies
                  with the correct answer, or
      2)Produce an incorrect answer but produce a
                  correct answer using a previously
                  learned clause
Replace T by T'
```

Fig. 1. FOIDL algorithm for learning a clause

FOIDL seems to be very effective at learning concepts that fit the first-order decision list representation. However, it does have some drawbacks. It requires that all constants be explicitly specified, since it works exclusively in a top-down fashion. For the past tense problem, this requires the generation of all possible prefixes and suffixes that appear in multiple examples, and it requires searching through that space. The primary issue with FOIDL is simply that it cannot be applied to too many examples, particularly if the search space of constants and/or background predicates is large.

2.3 CLOG

The CLOG system was developed because attempts to apply FOIDL to other problems showed its limitations in regard to processing speed [7]. CLOG shares a number of characteristics with FOIDL, but takes a different approach to learning individual clauses.

Like FOIDL, CLOG uses the concept of output completeness to allow for implicit negatives and it produces first-order decision lists in the same order as FOIDL, prepending each new clause to the decision list.

However, the way in which CLOG learns rules is quite different. In the following discussion, PTC represents positive examples not covered by the decision list, and CPE represents positive examples already covered by the decision list. Until all examples are in CPE, CLOG selects an arbitrary example and creates all of the possible clauses that cover that example (using a user-defined predicate that accepts an example and returns the appropriate clauses for the example). For each of those clauses, it counts the number of examples in PTC that the clause covers positively and

negatively and then the number of examples in CPE that are positively or negatively covered by the clause. Given those four counts, CLOG calls a user-defined gain function to determine which of the clauses is best. Once a clause has been selected, examples positively covered by that clause are removed from PTC and added to CPE and examples negatively covered by the clause are added to PTC and removed from CPE.

This approach has some advantages over that of FOIDL. First, it is very fast, as will be shown in the discussion of experimental results. Second, it does not require the search through the space of constants, since those are created in the development of the possible covering clauses. Third, the management of exceptions to exceptions is integral to the algorithm

However, the approach also has its own set of drawbacks. It requires that the user of the system specify a predicate to generate the possible clauses covering a given example. In the case of morphological learning, the domain to which CLOG has been applied, this is relatively straightforward; however, it is easy to imagine cases where it would not be easy to do. The system is also being given a considerable amount of knowledge about what a clause is "supposed" to look like, so that needs to be taken into consideration when doing comparisons.

The second major drawback of the system is that the arbitrary choice of an example for building a rule means that rules are likely to not be constructed from most general to least general. An examination of the rules produced by CLOG quickly shows that it often fails to learn a general "default" rule. Manandhar, Dzeroski, and Erjavec [7] point this out as a positive feature in comparison to FOIDL, since the system is more likely to fail to produce any answer than to produce an incorrect answer. However, we would argue that a wrong answer may be more desirable than no answer at all. When trying to generate the past tense form of "steal", we believe that it is better to produce "stealed" than to produce nothing, since the incorrect answer is, in fact, comprehensible, even though it is wrong. Certainly, there are cases where it may be desirable to fail instead, but first-order decision lists seem to be most applicable to those cases where a default answer is likely to be appropriate. We will also show that CLOG's accuracy is not consistently competitive with FOIDL's.

3 BUFOIDL

The BUFOIDL (Bottom-Up First Order Induction of Decision Lists) algorithm was inspired by a desire to overcome the limitations of FOIDL without trading off accuracy and flexibility. Since the primary cause of FOIDL's long learning times stems from its top-down search through a large space of constants and possible literals, we decided to take a bottom-up approach to the problem.

Many characteristics of BUFOIDL come directly from FOIDL. BUFOIDL handles intensionally defined background predicates in much the same way as FOIDL. It uses the output completeness assumption to handle implicit negative examples just as FOIDL and CLOG do. It constructs the decision list in the same way as FOIDL, first learning a most general clause, then learning more specialized

```
Get-Generalizations(PTC,PCE,old-clauses)
 Initialize example-pool to PTC
 Set prev-clauses to empty
 For each cur-clause in old-clauses
   Evaluate cur-clause
   If cur-clause covers more positives than negatives
      Remove covered positives from example-pool
   Else old-clauses = old-clauses - clause
 Repeat
   Select
     k pairs of examples from example-pool
     k pairs of one example from example-pool and one
        clause from prev-clauses
     k pairs of clauses from prev-clauses
   Generate the most-specific clause covering each
        selected example
   For each pair
      cur-clause = the LGG of the pair
      Evaluate cur-clause
      If cur-clause covers more positives than
           negatives
         Add cur-clause to prev-clauses
         Remove covered positives from example-pool
         Move parent(s) of cur-clause to old-clauses
 Until no new clauses are found
 Return old-clauses + prev-clauses
```

Fig. 2. BUFOIDL algorithm for creating a set of generalizations from which to select a new clause.

"exceptions" to the clauses below. BUFOIDL also uncovers positive examples under the same circumstances as FOIDL.

Thus, the algorithm for the outer loop is very similar to FOIDL's outer loop. BUFOIDL, however, takes a different approach to the construction of individual clauses, inspired by previous bottom-up approaches, particularly from GOLEM [9] and PROGOL [10].

3.1 Clause Construction

BUFOIDL constructs a group of potential clauses and selects from that group the best next. The clause construction algorithm, which is very close to GOLEM, much as FOIDL follows FOIL, is shown in Figure 2.

Note that old-clauses parameter refers to clauses other than the one selected on previous iterations. It does not include clauses from the current definition.

Several aspects of this algorithm require explanation. First, the concept of a "most-specific clause" is a difficult issue, and one of the problems that all bottom-up

```
Example: past([k,6,m],[k,e,m]).
Type: past(word,word)
Mode: past(+,-)

Background: split(X,Y,Z)
Type: split(word, prefix, suffix)
Mode: split(+,-,-) or split(-,+,+)

Most-specific clause:
    past([k,6,m],[k,e,m]) :-
        split([k,e,m],[k],[e,m]),
        split([k,e,m],[k,e],[m]),
        split([k,6,m],[k],[6,m]),
        split([k,6,m],[k,6],[m]).
```

Fig. 3. Example of most-specific clause generation in BUFOIDL. split(X,Y,Z) is equivalent to append(Y,Z,X) with non-empty Y and Z

approaches to ILP must address in some way. We allow intensionally defined background predicates, so we cannot use RLGGs as GOLEM does [9]. We chose not to require extensive information about the syntactic form of the learned clause as systems such as CLOG and PROGOL do [7,10]. Instead, we take the approach of using only type and mode information for each of the predicates, and applying the background predicates to the initial example in all possible ways, collecting the resulting literals and using the conjunction of all of those literals as the "most-specific clause". For some problems, this process might need to be repeated multiple times, and could lead to significant increase in complexity of the LGG process, but BUFOIDL attempts to control this complexity in two ways. A parameter is provided to limit the maximum times the technique is applied, and BUFOIDL attempts to learn clauses using a single level, using multiple applications of the technique only when the system fails to learn a new clause (on the principle that the system is designed to learn more general clauses first). As an example of the construction of a most-specific clause, consider Figure 3, taken from the English past tense task. .

For some problems, this collection of literals needs to be done multiple times, so BUFOIDL has a *depth* parameter that determines how deep this search for literals is permitted to go. The system then performs an iterative deepening search, increasing the depth of the literal collection process when the algorithm fails to produce an acceptable new clause.

For the past tense problem that we focus on here, a depth of 1 is sufficient, and, in fact, the types and modes used by FOIDL and BUFOIDL for the problem, as well as the clause construction predicate used with CLOG, all constrain the learned rules to have a depth of 1. This actually may limit the ability of the representation to appropriately generalize exceptions such as *drink-drank*, but it is not a problem for the task in general.

Clauses are evaluated using output queries as described in Section 2.2. Because clause creation is bottom-up, we need not be concerned with estimating the number of negatives covered by non-ground responses. We simply eliminate any clause that

produces non-ground answers as overly general. Note that negative examples come in two forms: BUFOIDL allows for explicit negative examples, which are treated as negatives when covered in the usual way, but we also consider as negative any incorrect answer to an output query if and only if some previously learned clause produces a correct answer to that query. Wrong answers for positive examples that have not yet been correctly covered are ignored, since we assume that they are exceptions to the current rule which will be handled by a rule that will be learned later (thus appearing earlier in the decision list).

Note that clauses covering examples negatively are kept for consideration and generalization if they cover more positives than negatives (i.e. produce more correct than incorrect ground responses). This is to allow for the possibility of exceptions to exceptions as discussed in Section 2.2 and below.

Another important aspect of BUFOIDL's clause construction is the process of search for a good generalization. Rather than randomly selecting one or several pairs and then generalizing each of those as much as possible without over-generalizing, BUFOIDL attempts a wider search, randomly selecting pairs of examples and rules repeatedly in a single clause construction phase and generalizing each pair as little as possible. The reason for this approach is that we would like to select the single most general clause possible at each step. This is not as important for an unordered set of rules such as GOLEM constructs. Of course, our approach cannot guarantee finding the best clause, but no heuristic search method can. Like GOLEM, BUFOIDL prunes the learned clause, dropping unnecessary literals.

Finally, note that k, the number of pairs to be generalized at each iteration, is a parameter to the algorithm. It should be set with some attention to the expected number of rules in the decision list to help ensure that at least one acceptable generalization will be found. Too many pairs should not harm the quality of the decision list learned, but will increase learning time, since the algorithm's execution time is highly dependent on the number of pairs selected at each pass.

3.2 Building the Decision List

As stated previously, BUFOIDL builds its decision list from the bottom up, placing the first, most general, rule learned at the end and prepending newly learned clauses to the list. However, a few details of the algorithm merit further explanation. The basic algorithm appears in Figure 4.

In order to make the search for clauses more efficient, BUFOIDL saves all of the acceptable clauses from previous iterations and evaluates them at the beginning of the search for a new clause, removing examples they cover from the pool to be generalized from if the clause is acceptable. While this is unlikely to helpful in all cases, as after the creation of the default clause, it may be very helpful later, when two clauses to be learned may not interact.

```
PTC = positive-examples
old-clauses = {}
CPE = {}
dec-list = empty
while PTC not empty and not done
   gen-list = Get-Generalizations(PTC,CPE,old-clauses)
   if gen-list is empty
      done = true
   else if some gen in gen-list covers no negatives
      best-gen = the generalization covering no
           negatives and the most positives
      Add best-gen to beginning of dec-list
      newCPE = the positives covered by best-gen
      PTC = PTC – newCPE
      CPE = CPE + newCPE
      old-clauses = gen-list – best-gen
   else
      best-gen = the generalization with the greatest
           difference between # of positives covered
           and # of negatives covered
      newPTC = the positive examples corresponding to
           the negatives covered by best-gen
      PTC = PTC + newPTC
      CPE = CPE – newPTC
      old-clauses = gen-list
```

Fig. 4. Algorithm for building the decision list in BUFOIDL.

For example, in handling English plurals, we may learn a default clause that adds *s* to a word. We then may create two clauses, one that adds *es* if the word ends in *s* and one that adds *es* if the word ends in *z*. Both are clauses that we want in the decision list, and they do not interact with each other. BUFOIDL will be able add the clause that covers more examples and save the other to add in the next iteration of the loop without having to reconstruct it. Given BUFOIDL's broad search, this is important to the overall efficiency of the algorithm.

4 Experimental Evaluation

To evaluate BUFOIDL, we ran experiments using the English past tense task for which FOIDL was initially developed. This data set is one that all three systems can easily be applied to; it is fairly well known; and it is large enough to allow us to determine whether BUFOIDL actually overcomes the performance problems that limit FOIDL.

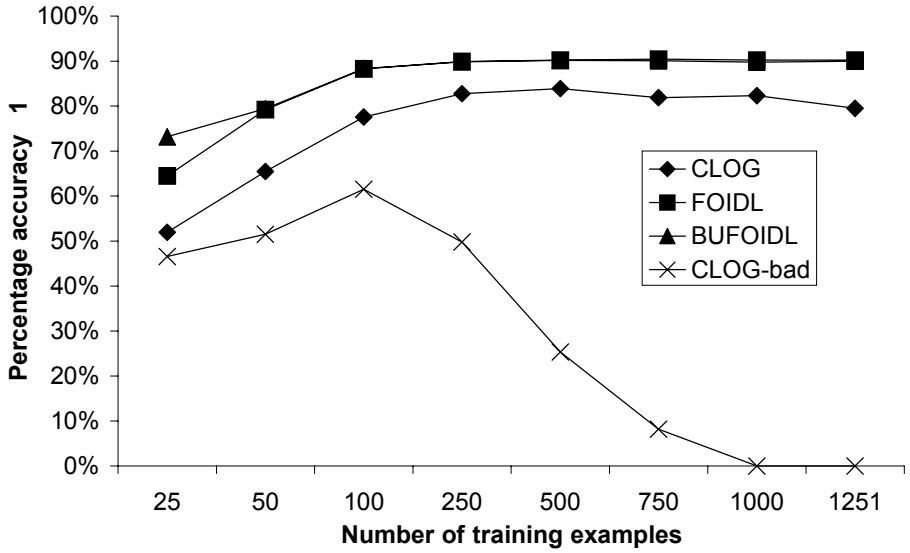

Fig. 5. Accuracy on full past tense task.

4.1 Experimental Design

The data used consists of 1390 verbs paired with their past tense forms in UNIBET phonemic encoding. The task is to generate the past tense form of the verb given the base form. We actually performed two experiments: one using the full set of verbs and a second that used only the regular verbs. The second task is much simpler than the first, and allows for 100% accuracy in theory (as the full task does not).
All three systems used the background predicate *split*/3 described earlier in the paper. For CLOG, we used an intermediate predicate called *mate* that simplifies the construction of the possible generalizations of an example. All of the user-defined portions of CLOG are used as provided for the tasks of generating English plurals. This is an example that comes with the system, and is highly similar in nature to the past tense task. The primary parameter for BUFOIDL is the number of pairs to select. For the regular task, the number of pairs was set to 10, and we used 25 pairs when irregular verbs were included, since there is a much larger number of clauses to be learned in this case. In these experiments, we used a 10-fold cross-validation of the data, and also ran learning curves. Statistical significance was determined using 2-tailed paired t-tests.

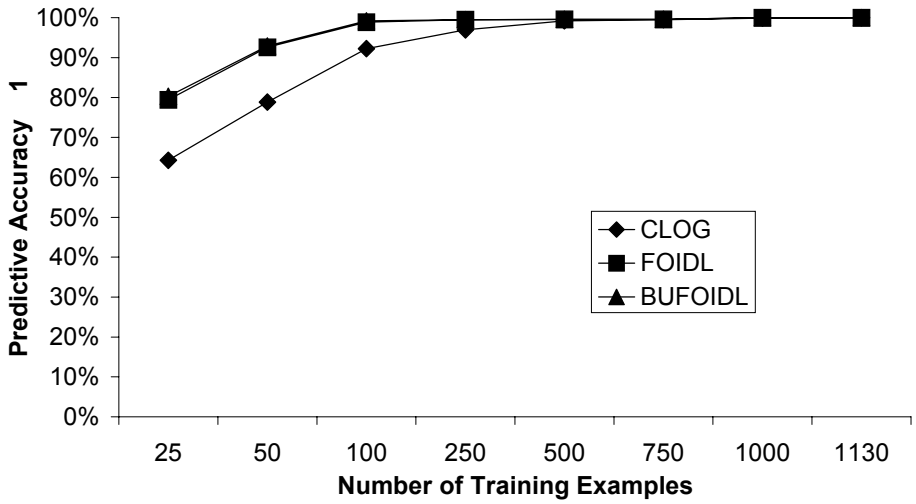

Fig. 6. Performance on past tense task using only regular verbs

4.2 Experimental Results

Figure 5 shows the results of running the three systems on the full past tense task. BUFOIDL and FOIDL perform well and very similarly on the task. With very few examples, BUFOIDL actually outperforms FOIDL, though FOIDL does slightly better than BUFOIDL with large numbers of examples. The differences between these two systems are attributable to the different search mechanisms. Like its predecessor FOIL, FOIDL uses a hill-climbing approach that is typically effective but can fail. BUFOIDL uses a broader search through the space, but it does have a random element. It seems that each approach can outperform the other at times, but there are no statistically significant differences between the two systems.

It is important to note, regarding CLOG's performance on the past tense task, that running these experiments highlighted one of the potential problems with CLOG's approach. As explained earlier, CLOG requires that the user of the system supply predicates to construct the set of clauses that are possible generalizations of an example, to specify whether a clause is a generalization of another clause, and to specify a gain function. In order to run the past tense task, we used predicates for these purposes that the system developers supply for the task of producing English plurals, since that task has many similarities to the past tense task. However, the supplied predicates assume that the original word and the modified word (in the original task, the singular and plural forms; in our case, the base form and the past tense form of the verb) share a common prefix. In general, this is the case, but our data set includes the pair *go-went* and *eat-ate*, for which the assumption does not hold true. The presence of either of these verb pairs in a training set caused CLOG to fail

Table 1. Learning times in seconds for the full phonetic past tense task using different numbers of examples

	FOIDL	CLOG	BUFOIDL
25	1.003	0.342	178.431
50	4.541	1.076	199.121
100	25.379	3.343	204.16
250	407.225	11.487	179.859
500	3956.708	30.505	201.558
750	16,102.84	59.296	208.714
1000	42,237.62	92.733	188.925
1251	92,623.57	132.612	241.837

without producing a decision list that could be tested. This situation let to the results label CLOG-bad. Therefore, we re-ran the experiments on slightly modified training sets from which those two verb pairs have been removed. The performance of FOIDL and BUFOIDL was identical, but CLOG's performance improves. Note that the removal of these verbs, rather than reducing potential accuracy, actually makes the task of the learner slightly easier, since both pairs function purely as noise in learning the decision list.

Clearly, this task demonstrates that CLOG is not always competitive with FOIDL and BUFOIDL in terms of predictive accuracy. Its approach seems to suffer greatly from the large number of exceptions, and accuracy is actually lower with larger numbers of examples. All of the differences between CLOG and BUFOIDL and those between CLOG and FOIDL with more than 25 training examples are statistically significant at the 0.01 level or better.

Figure 6 shows the result of running the systems on just the regular verbs. Here the systems are more comparable. FOIDL and BUFOIDL again perform very similarly, while CLOG is significantly lower with 250 examples or fewer, but eventually catches up to the other two systems.

So far we have shown that BUFOIDL has accuracy comparable to FOIDL's, but this is not sufficient to motivate the development of a new algorithm. The other issue is learning time. Table 1 shows the learning times for the complete phonetic past tense task, and Table 2 shows the learning times for the task involving regular verbs only. These numbers clearly demonstrate the issue of learning time in FOIDL. While the system learns quickly from a small number of examples, the learning time increases rapidly, quickly becoming unreasonable. On the full phonetic past tense task, FOIDL averages over 25 hours of CPU time to learn from 1251 examples. This problem is exacerbated by a related increase in memory use that makes it difficult to run larger problems at all.

The learning times also show that CLOG's designers achieved their goal of creating decision lists in far less time than FOIDL requires. The difference is dramatic. It is

Table 2. Learning times in seconds for the regular past tense task using different numbers of examples

	FOIDL	CLOG	BUFOIDL
25	0.771	0.298	86.802
50	3.347	0.941	87.525
100	18.636	2.22	93.101
250	230.096	7.777	109.334
500	1716.971	15.422	121.364
750	5586.535	24.072	101.839
1000	15,930.2	32.318	116.58
1125	17,951.2	36.391	113.085

interesting to note that the increasing number of exceptional cases seems to have a strong impact on both CLOG's and FOIDL's learning times. However, it is very clear that CLOG can handle many more examples than FOIDL.
Unlike FOIDL and CLOG, BUFOIDL's time depends less on the size of the training set than on the number of pairs selected. Because of this, the system takes quite a bit longer than the others on small example sets. In the full phonetic past tense task, the time required by BUFOIDL does not even show a clear trend toward increasing learning times. In the case of regular verbs only, the learning times trend more clearly upward, but they increase fairly slowly.

4.3 Discussion

In considering the results presented here, it is important to recognize that each of the three systems discussed in this paper have both strengths and weaknesses.
FOIDL provides consistent good accuracy, and can be very fast with small numbers of examples. It is important to note that FOIDL performs quite well on tasks where a top-down approach would be expected to do well (relatively few constants, smaller space of possible literals, fewer examples). Preliminary experiments with the finite mesh domain showed that FOIDL consistently outperformed BUFOIDL for that task, with at least comparable accuracy and better speed. However, FOIDL has significant problems dealing with larger search spaces.
CLOG was developed in response to this key problem with FOIDL, and, as a result, it is very fast. However, its two major drawbacks can be significant. First, the decision list learned (and its quality) may depend heavily on the order in which examples are presented, since CLOG simply generalizes the first example in the training example set at each iteration. The learning algorithm does not seem to be as effective as the other two in producing decision lists with good predictive accuracy.

The second issue with CLOG is not a major drawback for the past tense task on which it was evaluated here, but could greatly limit the applicability of the approach. For the past tense task (and other similar tasks), it is fairly easy to determine what possible clauses can be constructed to generalize a given example. However, this is not the case for all tasks of potential interest. To construct the needed user-defined predicates would be an onerous task for some problems.

BUFOIDL is presented here as an alternative that does not share its predecessors' weaknesses; however, it is not a perfect answer. The system relies on the random selection of pairs of examples to generalize. If an insufficient number of examples is chosen at each iteration, BUFOIDL may fail to learn an accurate decision list. Of course, the system should not perform worse when selecting more pairs than required. The primary trade-off here is that the learning time is impacted by the number of pairs selected to learn from. Selecting many more pairs than are required will result in longer learning times than necessary.

A second weakness of BUFOIDL is its very long learning time for small training sets. Although BUFOIDL's learning time for larger training sets is clearly superior to FOIDL's, it takes far longer to learn from smaller training sets, since the training time is more closely tied to the number of pairs of examples chosen than to the number of examples in the training set. However, we perceive the necessity of spending minutes rather than seconds to learn from 25 examples well worth the advantage of learning from 1000 examples in minutes (rather than hours) as well.

Although BUFOIDL does not approach the lightning speed of CLOG, it clearly makes the learning of first-order decision lists with the level of accuracy that FOIDL provides a realistic possibility with larger example sets.

5 Related Work

The systems most closely related to BUFOIDL are CLOG and FOIDL. However, three other systems deserve mention.

Around the time of the development of FOIDL, Quinlan developed an alternate system for learning decision lists called FFOIL [12]. Quinlan's approach is based on FOIL, requiring extensional definitions of background predicates. While FFOIL does learn a decision list, it places the rules in the opposite order from BUFOIDL and FOIDL, so it does not take an approach of learning exceptions to previously learned rules. It does, however, learn a default rule that simply predicts the most common output and places that rule at the bottom of the decision list.

The TILDE system [1,2] is also very closely related to BUFOIDL. TILDE induces logical decision trees using a top-down approach that incorporates Blockeel and De Raedt's method of *learning from interpretations*. Blockeel and De Raedt show that their binary logical decision trees are equivalent in expressiveness to first order decision lists. However, their approach is not easily applicable to problems such as the past tense task discussed in this paper for two reasons. First, their approach is a top-down approach that requires the specification of constants to be used in the definitions (a problem FOIDL also suffers from). More importantly, the method of

learning from interpretations is specifically focused on classification tasks, and this particular task is not easily transformed into a classification paradigm.
Another somewhat related area of work is transformation-based learning [3]. Transformation-based learning systems learn a list of rules, and each rule strives to correct the errors made by the previous rules. Thus, we can see similarities of concept between the approaches of first-order decision list learning and transformation-based learning, as both do learn lists of rules, from general to specific, and both focus on learning rules to handle exceptions to previously learned rules. However, transformation-based learning systems apply all of the learned rules in order, while decision lists apply only the first applicable rule. Thus, one would expect the decision lists systems to be faster. It remains to be seen whether either approach is more accurate than the other.

6 Conclusions and Future Directions

In this paper, we have presented a new approach for learning first-order decision lists and have shown that it provides considerable speed-up over the most accurate existing system for learning this representation, while also providing comparable accuracy. However, we have only applied the system to data that the existing top-down approach could handle. The purpose in developing such a system is, of course, to be able to apply this learning approach to data sets too large for FOIDL. Therefore, our major direction for future work is to attempt to apply BUFOIDL to appropriate tasks. We will be looking primarily at language tasks, since those seem to fit the decision list paradigm. We also hope to do some comparisons between transformation-based learning approaches to language learning and decision list approaches.

References

1. Blockeel, H., De Raedt, L.: Top-Down Induction of First-Order Logical Decision Trees. Artificial Intelligence 101 (1998) 285-297
2. Blockeel, H., De Raedt, L., Jacobs, N., Demoen, B.: Scaling up Inductive Logic Programming by Learning from Interpretations. Data Mining and Knowledge Discovery. 3 (1999) 59-93
3. Brill, E.: Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. Computational Linguistics. 21 (1995) 543-565
4. Califf, M.E., Mooney, R.: Advantages of Decision Lists and Implicit Negatives in Inductive Logic Programming. New Generation Computing. 16 (1998) 263-281
5. Califf, M.E., Mooney, R.:. Relational Learning of Pattern-Match Rules for Information Extraction. In Proceedings of the Sixteenth National Conference on Artificial Intelligence. AAAI Press Menlo Park, CA (1999) 328-334
6. Clark, P., Niblett, T.: The CN2 Induction Algorithm. Machine Learning, 3 (1989) 261-284
7. Manandhar, S., Džeroski, S., Erjavec, T.: Learning Multilingual Morphology with CLOG. In Proceedings of the 8th International Workshop on Inductive Logic Programming. Springer-Verlag Berlin Heidelberg New York (1998) 135-144

8. Mooney, R., Califf, M.E.: Induction of First-Order Decision Lists: Results on Learning the Past Tense of English Verbs. Journal of Artificial Intelligence Research. 3 (1995) 1-24
9. Muggleton, S., Feng, C.: Efficient Induction of Logic Programs. In Proceedings of the First Conference on Algorithmic Learning Theory. Tokyo, Japan (1990) 368-381
10. Muggleton, S.: Inverse Entailment and Progol. New Generation Computing. 13 (1995) 647-657
11. Quinlan, J.R.: Learning Logical Definitions from Relations. Machine Learning. 5 (1990) 245-286
12. Quinlan, J.R.: Learning First-Order Definitions of Functions. Journal of Artificial Intelligence Research. 5 (1996) 139-161
13. Rivest, R.L.: Learning Decision Lists. Machine Learning. 2 (1987) 229-246
14. Webb, G.I., Brkič, N.: Learning Decision Lists by Prepending Inferred Rules. In Proceedings of the Australian Workshop on Machine Learning and Hybrid Systems. Melbourne, Australia (1993) 6-10