Revising First-order Logic Theories from Examples through Stochastic Local Search

Aline Paes[1], Gerson Zaverucha[1], and Vitor Santos Costa[2]

[1] Department of Systems Engineering and Computer Science - COPPE
Federal University of Rio de Janeiro (UFRJ), Brazil
{ampaes, gerson}@cos.ufrj.br
[2] LIACC and DCC/FCUP, Universidade do Porto, Portugal
vsc@dcc.fc.up.pt

Abstract. First-Order Theory Revision from Examples is the process of improving user-defined or automatically generated First-Order Logic (FOL) theories, given a set of examples. So far, the usefulness of Theory Revision systems has been limited by the cost of searching the huge search spaces they generate. This is a general difficulty when learning FOL theories but recent work showed that Stochastic Local Search (SLS) techniques may be effective, at least when learning FOL theories from scratch. Motivated by these results, we propose novel SLS based search strategies for First-Order Theory Revision from Examples. Experimental results show that introducing stochastic search significantly speeds up the runtime performance and improve accuracy.

1 Introduction

A variety of Inductive Logic Programming systems have been developed to automatically learn First-Order Logic (FOL) theories [11], [4], with good results on a number of important applications [12, 10], [3]. Most such systems are designed to learn theories from scratch, given a set of examples and a fixed body of prior knowledge, the *background knowledge*. There has been relatively less work on the problem of *repairing* incorrect or incomplete theories. One example of theories that could be repaired or improved are theories that had been elicited from a domain expert, and thus may include useful information, but on the other hand may be incomplete, and/or rely on incorrect assumptions, or even be inconsistent. A second common example is the case where new examples are not well explained by the original theory. In such cases standard ILP systems would take one of the two following positions: they could either discard the initial theory, or consider it as part of background knowledge which can not be modified.

Since the task of knowledge acquisition is difficult and time-consuming, and since the original theory may contain valuable prior information, one would like to take advantage of the original theory as a start point to the learning process. Ideally, this should accelerate learning time and result in more accurate theories. Several *theory refinement* systems have been proposed towards this goal [15, 6, 14]. Such systems assume the initial theory is approximately correct. If so, then only some *points* in the theory prevent it from correctly modeling the dataset.

The idea is therefore that it should be more efficient to search for such points in the theory and *revise* them, than to use an algorithm that learns a whole new theory from scratch. Prior studies show that this approach can work well and even require less examples in some cases [6]. Note that these revision algorithms can be seen as a generalization from learning from scratch, as performed by most ILP systems. However, in constrast to most ILP systems, theory revision algorithms do not apply cover removal, instead they perform search in the space of whole theories. Arguably, cover removal, that sequentially searches for a clause that explain unconvered examples, frequently generate unnecessarily long hypothesis with too many clauses.

Theory revision systems operate by searching for revision points, that is, the points which explain faults in the theory, and then proposing revisions to such points, through applying at each point a number of matching *revision* operators. Therefore, theory revision can be seen as a search process, and very much as most ILP algorithms, revising logic programs may need to search a very large search space and therefore may incur big time and storage requirements. Search space grows quickly with the size of the knowledge base. We would also expect for search to be harder if the theory has more faults. Last, Theory Revisions systems are particularly ambitious in that they tackle whole theories, which is known to be a hard problem [1].

One possible way of alleviating the huge requirements of searching in theory revision algorithms is to take advantage of clever search strategies such as stochastic local search (SLS). Such methods have been successfully applied to solve difficult combinatorial propositional problems [7, 9, 8] and recently they have also been applied to learn theories from scratch in ILP systems [5, 13], substantially improving the efficiency of both domains. Motivated by these works and by the increased combinatorial explosion of searching in entire theories, we investigate the relevance of applying SLS in theory revision algorithms. To do so, we develop greedy and hill-climbing algorithms that performs stochastic local search, when proposing revisions and when searching for a revision to be implemented, and we compare them to a theory revision algorithm that performs only greedy hill-climbing search.

The outline of the paper is as follows. Some preliminary knowledge concerning SLS and theory revision are reviewed in sections 2 and 3, respectively. The algorithms developed to revise FOL theories from examples through SLS are devised in section 4. Experimental results are presented in section 5, followed by conclusions and future work in section 6.

2 Stochastic Search

Stochastic Search algorithms are a family of search algorithms that strongly rely on randomized decisions while searching for solutions. Stochastic Local search algorithms (SLS) are based on local search techniques. They therefore abandon completeness in favor of trying to achieve the best exploitation of bounded resources [7]. Unlike the greedy hill climbing approach, it avoids getting caught in local optima by allowing random steps. One major motivation and successful application of SLS has been in satisfiability checking of propositional formulae, namely through the well-known GSAT [9] and WalkSAT [8] algorithms. A large number of tasks in areas such as planning, scheduling and constraint solving can be encoded as a satisfiability problem, and empyrical observations show that SLS often can substantially improve their efficiency [2, 7].

Stochastic Search in Machine Learning Several machine learning algorithms can be described as search algorithms. There has therefore been some interest in applying SLS and related techniques to this area.

Chisholm and Tadepalli [2] used stochastic search to perform rule learning in their system LERILS and compare its performance to other learning algorithms, with encouraging results. Again in the area of rule learning, Rückert et.al. present an SLS algorithm for learning k-term DNFs, that is, theories at most k clauses [7]. They proposed a novel SLS algorithm specific for this task, and evaluated its performance with excellent results.

Stochastic Search in ILP Several Inductive Logic Programming algorithms perform search on a vast search spaces, and most of them do include a limited amount of stochastic search. As an example, Progol-like systems randomly select examples as *seeds* to start their search [4, 11]. It is therefore unsurprising that research on stochastic search has taken place since early ILP days [13].

A recent study takes this point further by implementing and evaluating the performance of several randomization strategies in the ILP system Aleph [13]. The authors use a deterministic general-to-specific search as reference. They then compare a variety of randomized algorithms. The stochastic strategies were framed in terms of a single clause search algorithm. The results indicate that the randomized search strategies outperforms in terms of search space deterministic clause search across large intervals of the parameter space, and suggests that similar techniques should be worthwhile across ILP.

3 First-order Logic Theory Revision

First-order theory revision is a challenging subject, particularly complex because we do not revise single clauses; instead we must deal with the issues arising from including a theory with multiple clauses.

In this work we follow the FORTE revision system (First Order Revision of Theories from examples) [6]. FORTE performs hill-climbing search through a space of both specialization and generalization operators in an attempt to find a minimal revision to a theory that makes it consistent with the set of training examples. The top-level algorithm is exhibited as Algorithm 1. The key ideas are:

1. Identify all the revision points in the current theory.
2. Generate a set of proposed revisions for each revision point starting from that one with the highest potential and working down the list. *Potential* is defined as the number of misclassified examples that could be turned into correctly classified from a revision in that point. The revision are proposed

through the revision operators. In this work we consider *Delete-rule* and *Add-antecedent* as specializations operators and *Delete-antecedent* and *Add-rule* as generalization operators. [3]
3. Score each revision through the actual increase in theory accuracy it achieves.
4. Retain the revision which most increases the score.

FORTE stops when the potential of next revision point is less than the score of the best revision to date. If the best revision really improves the theory it is implemented. Conceptually, each operator develops its revision using the entire training set. However, in practice, this is usually unnecessary and thus FORTE considers only the examples whose provability can be affected after proposing some revision.

Algorithm 1 FORTE Algorithm (Richards and Mooney, 1995)

```
repeat
   1. generate revision points;
   2. sort revision points by potential (high to low);
   3. for each revision point
   4.      generate revisions;
   5.      update best revision found;
   6. until potential of next revision point is less than the score of the best revision
   to date
   7. if best revision improves the theory
   8.      implement best revision
until no revision improves the theory;
```

4 Stochastic First-order Logic Theory Revision

Previous research on stochastic search for ILP has focused on clause search starting from the empty theory. Next, we study whether stochastic search can improve performance on the theory-level search performed by theory revision systems. Although our studies could be performed on any theory revision algorithm, we choose the FORTE system to implement and experimentally evaluate our approach. Following WalkSAT and related SLS algorithms, the approaches we propose perform a local, randomized-walk search, alternating between stochastic and greedy moves with the type of the move to be executed being chosen according to a prior probability p.

As observed above, a first major difference between our problem and prior research is that we need to randomize search over theories. A second major difference between our problem and prior approaches is that we would not like to start from a random or empty hypothesis. Instead, we would like to take advantage of an initial theory provided to the system.

[3] There are others operators not used here, such as *predicate invention* and *abduction*. Actually, any operator used in first-order machine learning cn be used in a theory revision system.

In order to choose the revision to be implemented, theory revision system such as FORTE proceed in two steps. First, the system searches for proposed revisions through considering all revision operators at all revision points. This process can be quite expensive as some operators must add antecedents, which requires searching through all possible goals in the database. Next, the revision system chooses the revision with highest potential and considers its score. This argues for two different phases where one could take advantage of a randomized strategy:

1. Revision search: instead of considering all possible revisions, we might search for revisions that improve the score, as in SLS algorithms.
2. Operator Search: as operator search is dominated by antecedent search, we may benefit from randomizing antecedent search.

4.1 Stochastic Local Search for Revisions

We propose two algorithms for randomizing revision search. The first algorithm, *greedy*, implements greedy stochastic local search. The algorithm stops either when it reaches a maximum number of steps or when it reaches a maximum score. The second algorithm, *hill-climbing* chooses a move at random if it *improves* the score, and only if so. The *hill-climbing* algorithms stops when further revisions cannot improve the score in the same way FORTE does. The greedy and hill-

Algorithm 2 A greedy SLS theory revision Algorithm

```
while score < maxScore and steps < maxSteps do
  1. generate revision points;
  2. with probability $p$
  3.     list all possible revisions from the generated revision points;
  4.     next_revision = a revision chosen at random from the list of possible revisions;
  5. otherwise
  6.     FORTE Generate and Search revisions procedure
  7.     next_revision = best revision;
  8. implements next_revision;
  steps ++;
end while
```

climbing SLS revision theory algorithms are presented in detail as Algorithms 2 and 3, respectively, where *FORTE Generate and Search procedure* corresponds to lines 2 to 6 in Algorithm 1. As usually in SLS methods, the algorithms:

1. With probability p, do a random move;
2. With probability $1 - p$, do a FORTE-like move;

The two algorithms differ on the random move and on how to terminate. In the greedy algorithm, a revision is just chosen at random. In the hill-climbing algorithm, a revision is chosen at random but is only accepted if it actually improves the score. The two algorithms also differ on their stopping criteria. The greedy algorithm stops on finding a best solution and on time. The hill-climbing algorithm stops if finding that no revision can actually improve the score, just like FORTE does. Thus, Algorithm 3 replaces the lines 3-8 in Algorithm 1 while Algorithm 2 replaces the whole algorithm only because the stopping criteria in this last case is different from the original FORTE.

Notice that we do not need to explicitely enumerate all possible revisions. In fact, given a revision point, we know that it either contributes to the misclassification of positives, and is therefore a generalization point, to the misclassification of negatives, and is therefore a specialization point, or to the misclassification of both negatives and positives, and is therefore both. Thus, given all revision points and their types, we can estimate the number of possible revisions and select one at random.

Algorithm 3 A hill-climbing SLS theory revision Algorithm

```
1. with probability $p$:
2.     list all possible revisions from the generated revision points;
3.     next_revision = a revision chosen at random from the list of possible revisions
                                          whose score > current ;
4. otherwise:
5.     FORTE Generate and Search revisions procedure
6.     next_revision = best revision;
7. if next_revision improves the theory
8.     implement next_revision;
```

4.2 Stochastic Local Search for Antecedents

Except for the *Delete-Rule* operator, all revision operators must *search* the best antecedent to add/delete. This suggests using stochastic methods in order to explore the search space in a more efficient way. Next, we investigate a hill-climbing stochastic search. We believe that a greedy algorithm could lead to either too large or too small clauses, as it would run for a fixed number of steps. Therefore, the search strategies devised here always execute until adding/deleting more antecedents cannot improve the score. The stochastic version of algorithms for adding or deleting antecedents is exhibited in Algorithm 4. FORTE provides two separate algorithms for producing a specialized clause: hill-climbing antecedent addition and relational pathfinding. In this work we focus on hill-climbing, as it is more suitable to most datasets we consider. It is important to notice that the delete-antecedent operator benefits less from stochastic local search than add-antecedent, since the search space is restricted to goals in the clause, and is therefore much smaller.

The process of adding or deleting antecedents using a SLS component starts by getting all antecedents that could be added (deleted) in (from) a clause chosen as revision point. If the antecedents are being added, the algorithm must generate all possible antecedents from the database. If they are being deleted, it is enough to collect the antecedents from the clause. As usual, we can execute a stochastic or a greedy move, depending upon a fixed probability $p1$ (addition) or $p2$ (deletion). In a stochastic move, an antecedent is chosen at random and then scored. If this antecedent improves the current score it is added (deleted) in (from) the clause. Otherwise, another antecedent will be chosen at random. If the move is greedy, all the antecedents are scored and the one with the highest score is chosen.

Algorithm 4 Algorithm for adding/deleting antecedents using hill-climbing SLS

```
repeat
   get all antecedents;
   with probability $p1/p2$:
      choose an antecedent at random whose score improves the current one, and,
        add/delete it to/from the clause;
   otherwise:
      for each antecedent
         calculate score;
      add/delete to/from the clause the antecedent with the highest score if it im-
   proves
       the current score of the clause;
until no antecedent can improve the score;
```

5 Experimental Results

We performed a number of experiments in order to evaluate the different approaches using three ILP benchmarks. *Mutagenesis* is a well-known domain for predicting structure-activity relationship (SAR). We use here the "regression friendly" dataset as discussed in [12]. *Carcinogenesis* is another well-known domain from SAR for predicting carcinogenic activity in rodent bioassays [10]. *Alzheimer*, compares analogues of Tacrine, which is a drug against Alzheimer's disease, according to four properties [3]. In this work we used only the dataset related to the inhibit **amine** reuptake property.

Algorithms Our main aim is to investigate whether applying stochastic search in a theory revision algorithm makes the revision task more efficient. Therefore, in this work we analyse the following algorithms. **(a)** The original FORTE; **(b)** Greedy stochastic revision selection presented in Algorithm 2; **(c)** Hill-climbing stochastic revision selection presented in Algorithm 3. (b) and (c) use the FORTE search for revision operators. **(d)** Hill-climbing antecedents search according to Algorithm 4. In this case the search for revisions is executed as the original system. **(e)** Greedy stochastic revision search plus stochastic antecedents search (Algorithm 2 plus Algorithm 4); **(f)** Hill-climbing stochastic revision plus stocahstic antecedent search (Algorithm 2 plus Algorithm 4).

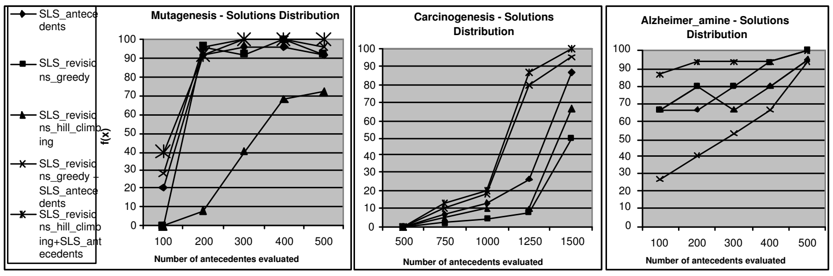
Experimental methodology All the algorithms were run using 10-fold cross validation; additionally, the stochastic algorithms were run 25 times. From previous experiments we define the following default parameters (future work could include using validation sets in an internal cross-validation to determine the best ones): **(a)** $p = 70\%$ for algorithms 2 and 3; **(b)** $p = 50\%$ for algorithms 4 and **(c)** maximum number of steps = 5 for Mutagenesis and 10 for Carcinogenesis and Alzheimer_amine when running algorithm 2, since these two last ones are more complex domains. The hypothesis were evaluated through their accuracy, which is the same evaluation function used in original FORTE. As this is a simple evaluation function and therefore can guide to overfitting, future work will makes use of an internal cross-validation procedure to decide the best stop point through a validation set.

5.1 Results

In this section we present the results through two kinds of experiments. Fist, we want to observe the behaviour of the stochastic algorithms, to find out which one has the best performance. Next, we compare the algorithms which performed better with a state-of-art ILP system.

Performance of stochastic algorithms Our first set of experiments aims at studying the performance of our stochastic algorithms versus the hill-climbing algorithm used by FORTE. Our methodology is as follows. First, we run Aleph on the datasets to obtain a theory. Second, we randomly perturb the theories by performing operations such as removing clauses, removing antecedents, changing variables, adding antecedents, and adding clauses. Thus, the theories actually sent to revision theory algorithms have accuracies on 67.02% on the training and of 67.09% on the test set for Mutagenesis; of 56.72% and 57.08% for Carcinogenesis; and of 61.08% and 61.02% for Alzheimer_amine. Last, we run the revision algorithms until it is *acceptable*. We define that a theory can be accepted if Mutagenesis reaches a training-set accuracy of 0.8, and if Carcinogenesis and Alzheimer reached an accuracy of 0.7 Actually, such accuracies were chosen because they are nearly the training accuracies of the initial theories, prior to the perturbations inserted on them. This procedure is similar to the *goodScore* heuristic defined in [16].

Figure 1 shows the percentage of acceptable theories found over 25 runs of each stochastic algorithm. The curves are plotted against how many antecedents we needed to evaluate until reaching an acceptable theory. Fig. 1(a) shows that in the case of Mutagenesis all but hill-climbing algorithm revision almost always reach acceptable theories after evaluating between 100 to 200 antecedentes only. Figure 1(c) shows equivalent results for Alzheimer_amine, but when evaluating more than 300 antecedents. Figure 1(b) shows that the threshold is much harder to achieve for Carcinogenesis. The percentage only goes above 80% for the algorithms that randomize antecedent evaluation, and they need to evaluate more than a 1,000 antecedents to achieve this goal.



(a) (b) (c)

Fig. 1. Solutions Distribution considering the number of antecedentes evaluated

Figure 2 show how predictive accuracy evolves as the theories are being revised. In order to compare the predictive accuracy, through the number of antecedents evaluated, stochastic algorithms and original FORTE we considered the same parameters above which indicate that a solution was found. Thus Figure 2 present test set accuracies for the algorithms considered. Figure 2(a) indicates that for Mutagenesis the original FORTE and SLS FORTE with hill-climbing on revisions improve performance accuracy to around 76% and then stabilize. In contrast, the algorithms that randomize antecedent selection very quickly improve performance with few revisions, and seem to still be improving even after 500 antecedents. Note that greedy SLS on revisions improves more slowly but still achieves over 80% performance. The results for Carcinogenesis show that the best accuracies are achieved by the algorithms that randomize antecedent search (close to 68%), except for the greedy revision algorithm. As usual just randomizing revision search does not do very well, and the greedy revision algorithm is actually outperformed by standard FORTE. Figure 2(c) shows that original FORTE stabilize in 71% and the hill-climbing stochastic procedure searching for revisions and antecedentes reach the best results. The greedy stochastic revision algorithm is outperfomed by all the others algorithms.

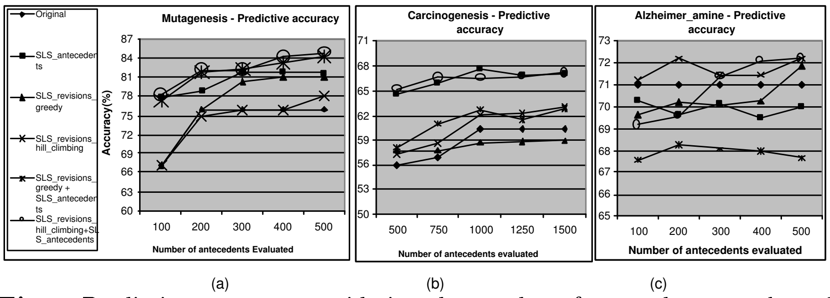


Fig. 2. Predictive accuracy considering the number of antecedentes evaluated

Table 1. Run time in seconds for each algorithm and dataset

Datasets	Original FORTE(a)	sls_rev_ greedy (b)	sls_rev_ hc (c)	sls_ antec. (d)	sls_rev_greedy& sls_antec. (d)	sls_rev_hc& sls_antec. (f)
Mutagenesis	28.11	**24.25**	**20.68**	**15**	**12.7**	**12.14**
Carcinogenesis	5046.15	**2329.34**	**2573.77**	**272.28**	**98.03**	**217.31**
Alz_amine	18672.18	**1558.67**	16178.89	**1607.19**	**373.06**	**730.95**

Table 1 compare actual run-times for these algorithms. Randomizing revision operators introduces a speedup, noticeably so for *Carcinogenesis* and *Alzheimer_amine*, but the real performance gain seems to be obtained from randomizing antecedent search. Stochastic search can achieve a speedup of over 50×, while actually improving predictive accuracy. The bold numbers correspond to a statistically significant improvement, according to a paired two-tailed t-test with 95% of significance level: all speedup improvements are statistically significant.

Table 2 shows test-set accuracy results. In general, although the results seem to indicate an overall improvement over the original FORTE, the only numbers that pass a significance test are the ones in bold face. All the significant re-

Table 2. Predictive accuracies for each algorithm and dataset

Datasets	Original FORTE(a)	SLS_rev_ greedy (b)	sls_rev_ hc (c)	sls_ antec. (d)	sls_rev_greedy& sls_antec. (d)	sls_rev_hc& sls_antec. (f)
Mutagenesis	78.21	78.32	78.53	83.74	**84.25**	83.13
Carcinogenesis	62.42	63.65	63.76	**66.41**	63.36	**66.73**
Alz_amine	**71.04**	65.92	**70.39**	**72.78**	65.65	**72.52**

sults require randomizing antecedent search. In Mutagenesis this is sufficient, Carcinogenesis and Alzheimer_amine benefits from randomizing revision search.

Comparison to Aleph From the first set of experiments it was possible to observe that the best trade-off between runtime and accuracy is reached by combined versions of stochastic search for revisions added by randomized search of antecedents. Thus, the last set of experiments study the question of whether these combined approaches of stochastic methods can improve theories obtained from the state-of-art ILP system Aleph, escaping from local minima in a reasonable time. Thus, we execute the following two experiments with the datasets mentioned above using as theory revision algorithm the Algorithms 3 and 2 added by a randomized search of antecedents performed by Algorithm 4:

1. In each fold of a 10 fold cross-validation procedure using 50% of the examples a theory is generated by Aleph (45% of the examples in the training data 5% in the test). Then each one of these theories returned by each fold are revised by the SLS algorithm and respective folds from all examples of the dataset. That is SLS revises the theory obtained from ALEPH (that was generated through the use of only 45% of the dataset) using the whole dataset (obviously without the 10% of the testset).
2. In each fold of a 10 fold cross-validation procedure a theory is generated by Aleph from all the examples in the dataset. Then each one of these theories are revised considering their respective fold (i.e., the same folds are used to generate and revise the theories).

Tables 3 and 4 show the values of accuracy and run time returned by theory revision stochastic algorithms, Aleph and original FORTE. In such tables, $SLSHC$ means that the stochastic search for revision is hill-climbing and $SLSgreedy$ means that the stochastic search for revisions is a greedy one. The probability for algorithms 3 and 2 were fixed in 50% in order to become less dependent from this parameter. Notice that we always show the predictive accuracy and runtime of the algorithms considering the whole dataset. The symbol $\star$ means that the stochastic algorithm makes a statistically significant improvement in Aleph and $\bullet$ indicate an improvement over original FORTE, both considering 95% of significance level. For Alzheimer_amine we run the revision algorithms with and without the relational pathfinding search for antecedents (the last row of the tables show the results without relational pathfinding).

From the tables we can see that acuracy is always significantly improved compared to Aleph and runtime is always improved compared to original FORTE. Only in Mutagenesis we could improve both measures returned by Aleph and

Table 3. Accuracy of Aleph, FORTE and combined stochastic revision theory algorithms. The values are obtained from the whole dataset

Dataset	Method	Aleph accuracy	Forte accuracy	SLS HC accuracy	SLS greedy accuracy
Mutag.	1	77.52	78.08	82.37 ⋆ ●	78.94⋆
	2	73.08	73.63	79.74 ⋆ ●	80.16 ⋆ ●
Carcinog.	1	50.58	54.91	61.60 ⋆ ●	59.91 ⋆ ●
	2	54.94	59.20	61.84 ⋆ ●	61.21 ⋆ ●
Alz_amine	1	62.11	63.79	68.13 ⋆ ●	66.53 ⋆ ●
	2	62.19	72.56	69.56⋆	66.83⋆
Alz_amine - rel. path.	1	62.11	62.91	68.57 ⋆ ●	67.71 ⋆ ●
	2	62.19	72.14	70.59⋆	68.75⋆

Table 4. Runtime of Aleph, FORTE and combined stochastic revision theory algorithms. The values are obtained from the whole dataset

Dataset	Method	Aleph runtime	Forte runtime	SLS HC runtime	SLS greedy runtime
Mutag.	1	20.74	11.23	5.55 ⋆ ●	10.12⋆
	2	20.74	6.65	5.93⋆	10.49⋆
Carcinog.	1	69.78	4958.72	591.78●	381.89●
	2	69.78	4929.16	380.39●	242.11●
Alz_amine	1	15.58	493.74	190.97●	253.41●
	2	15.58	4362.94	539.74●	519.28●
Alz_amine - rel. path.	1	15.58	509.79	62.31●	105.36●
	2	15.58	2710.82	124.58●	121.73●

FORTE. Ideally, both measures should be improved in most of domains. However, it seems that the search space still needs to be reduced (specially the one from antecedents). One way towards that is to use Mode Directed Inverse Entailment search [4]. We are currently enhancing in this way the algorithms presented here. Anyway, these results show that stochastic revision is a good alternative for improving the accuracy of theories returned by standard ILP systems in a reasonable time.

6 Conclusions

We design and evaluate a number of stochastic local search techniques for the revision of first-order theories. Our first results indicate that such techniques can significantly improve the run-time and even the accuracy of a revision algorithm. A more detailed analysis shows that much of the benefit comes from reducing the cost of searching new antecedents to add to a clause in a theory, as this search is quite expensive and seems to dominate revision costs. On the other hand, benefits are also achieved through randomizing search from revision operators.

Our current results suggest that improving antecedent generation should be a major concern in theory revision systems. One interesting future work would therefore be to constrain this space by using modes and the bottom-clause to guide the revision process (remembering that FORTE was based on FOIL).

Our results also seem to indicate that stochastic search has the potential to very significantly improve the performance of theory-revision systems, and that such improved systems may be useful in improve the theories generated by Inductive Logic Programming systems. We believe this is because theory revision systems can take a global perspective of theory, in contrast to the local approach used by the greedy cover-removal algorithms. Moreover, if Theory Revision systems can be made more efficient, this would make it more practical to revise theories given new examples. Initial experiments suggest this may be indeed the case, and that research in this direction may be very worthwhile.

Acknowledgments

The first author is financially suported by CAPES, the second author by CNPq and the third author by Fundacão para a Ciência e Tecnologia, Program POSI. We would like to thank Bradley Richards and Raymond Mooney for making the FORTE system available and Kate Revoredo for useful discussions.

References

1. Ivan Bratko. Refining complete hypotheses in ILP. In *Proc. of the 9th ILP*, LNAI 1634, pages 44–55. Springer, 1999.
2. Michael Chisholm and Prasad Tadepalli. Learning decision rules by randomized iterative local search. In *Proc. of the 19th ICML*, pages 75–82, 2002.
3. Ross D. King, Michael J. E. Sternberg, and Ashwin Srinivasan. Relating chemical activity to structure: An examination of ilp successes. *New Generation Computing*, 13(3-4):411–433, 1995.
4. Stephen Muggleton. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 1995.
5. Aline Paes, Filip Železný, Gerson Zaverucha, David Page, and Ashwin Srinivasan. ILP through propositionalization and stochastic k-term DNF learning. In *Proc. of the 16th ILP*, LNAI 4455, pages 379–393. Springer, 2007.
6. Bradley L. Richards and Raymond J. Mooney. Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.
7. Ulrich Rückert and Stefan Kramer. Stochastic local search in k-term DNF learning. In *Proc. of the 20th ICML*, pages 648–655, 2003.
8. Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:521–532, 1996.
9. Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the 10th AAAI*, pages 440–446, 1992.
10. A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. In *Proc. of the 7th Int. Workshop on ILP*, pages 273–287. Springer-Verlag, 1297, 1997.
11. Ashwin Srinivasan. *The Aleph Manual*, 2001.
12. Ashwin Srinivasan, Stephen Muggleton, Michael J. E. Sternberg, and Ross D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.
13. Filip Železný, Ashwin Srinivasan, and David Page. Randomised restarted search in ILP. *Machine Learning*, 64(1-3):183–208, 2006.
14. Stefan Wrobel. Concept formation during interactive theory revision. *Machine Learning*, 14(2):169–191, 1994.
15. Stefan Wrobel. First-order theory refinement. In Luc De Raedt, editor, *Advances in Inductive Logic Programming*, pages 14–33. IOS Press, 1996.
16. Filip Zelezný, Ashwin Srinivasan, and David Page. Lattice-search runtime distributions may be heavy-tailed. In *12th ILP, LNAI 2583*, pages 333–345, 2002.