ILP Through Propositionalization and Stochastic k-Term DNF Learning

Aline Paes[1], Filip Železný[2], Gerson Zaverucha[1], David Page[3], and Ashwin Srinivasan[4]

[1] Dept. of Systems Engineering and Computer Science - COPPE
Federal University of Rio de Janeiro (UFRJ)
{ampaes, gerson}@cos.ufrj.br
[2] Dept. of Cybernetics - School of Electrical Engineering
Czech Institute of Technology in Prague
zelezny@fel.cvut.cz
[3] Dept. of Biostatistics and Medical Informatics and
Dept. of Computer Sciences
University of Wisconsin
page@biostat.wisc.edu
[4] IBM India Research Laboratory and
Dept of CSE and Centre for Health Informatics
University of New South Wales, Sydney, Australia
ashwin@cse.iitd.ernet.in

Abstract. One promising family of search strategies to alleviate runtime and storage requirements of ILP systems is that of stochastic local search methods, which have been successfully applied to hard propositional tasks such as satisfiability. Stochastic local search algorithms for propositional satisfiability benefit from the ability to quickly test whether a truth assignment satisfies a formula. Because of that many possible solutions can be tested and scored in a short time. In contrast, testing whether a clause covers an example in ILP takes much longer, so that far fewer possible solutions can be tested in the same time. Therefore in this paper we investigate stochastic local search in ILP using a relational propositionalized problem instead of directly use the first-order clauses space of solutions.

1 Introduction

ILP has been successfully applied to a variety of tasks [12], [6]. Nevertheless, ILP systems have huge time and storage requirements, owing to a large search space of possible clauses. Therefore, clever search strategies are needed [13]. One promising family of search strategies is that of stochastic local search methods. These methods have been successfully applied to propositional tasks, such as satisfiability, substantially improving their efficiency. Following the success of such methods, a promising research direction is to employ stochastic local search within ILP, to accelerate the runtime of the learning process. An investigation in that direction was recently performed within ILP [22].

S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad (Eds.): ILP 2006, LNAI 4455, pp. 379–393, 2007.

Stochastic local search algorithms for propositional satisfiability benefit from the ability to quickly test whether a truth assignment satisfies a formula. As a result, many possible solutions (assignments) can be tested and scored in a short time. In contrast, the analogous test within ILP—testing whether a clause covers an example—takes much longer, so that far fewer possible solutions can be tested in the same time. Therefore, motivated by both the success and limitations of the previous work, we also apply stochastic local search to ILP but in a different manner. Instead of directly applying stochastic local search to the space of first-order Horn clauses, we use a propositionalization approach that transforms the ILP task into an attribute-value learning task. In this alternative search space, we can take advantage of fast testing as in propositional satisfiability. Our primary aim in this paper is to reduce ILP run-time.

The standard greedy covering algorithm employed by most ILP systems is another shortcoming of typical ILP search. There is no guarantee that greedy covering will yield the globally optimal hypothesis; consequently, greedy covering often gives rise to problems such as unnecessarily long hypothesis with too many clauses. To overcome the limitations of greedy covering, the search can be performed in the space of entire theories rather than clauses. A strong argument against this larger search is the combinatorial complexity, giving us another reason to transform the relational domains into propositional ones and to use stochastic local search in the resulting, simpler search space. Therefore, our secondary aim in this work is to verify the benefits of a non-covering approach to perform search in ILP systems.

In a recent work, a novel stochastic local search algorithm (SLS) was presented to induce k-term DNF formulae. The SLS algorithm performs refinements on an entire hypothesis rather than a single rule. A detailed analysis of SLS performance compared to WalkSAT shows the advantages of using SLS to learn a hypothesis as short as possible [15]. In this work we specifically investigate the relevance of that SLS algorithm to learn k-term DNF formulae in relational domains through propositionalization.

The outline of the paper is as follows. First, some background knowledge related to propositionalization and Stochastic Local Search are reviewed in Sections 2 and 3, respectively. Then the proposal of this paper, SLS in ILP through propositionalization and k-term DNF learning, is devised in Section 3.4. Before concluding, some experimental results which validate our method are shown in section 4.

2 Propositionalization

Propositionalization can be understood as a transformation method, where a relational learning problem is compiled to an attribute-value problem, which one can solve using propositional learners [9,7]. During propositionalization *features* are constructed from the background knowledge and structural properties of individuals. Each feature is defined as a clause in the form $f_i(X) := Lit_{i,1}, ..., Lit_{i,n}$ where the literals in the body are derived from the background knowledge and

the argument in clause's head refers to an individual as an example identifier. The features are the attributes which form the basis for columns in single-table (propositional) representations of the data. If such a clause defining a feature is called for a particular individual and this call succeeds, the feature is set to "true" in the corresponding column of the given example; otherwise it is set to "false". Recently several propositionalization systems have been proposed. Examples include: RSD [21] and SINUS [8] among others. In the next section we briefly review RSD since it is the propositionalization system used in the experiments of our approach.

2.1 RSD

RSD is a system that uses propositionalization through first-order feature construction for discovering statistically interesting relational subgroups in a population of individuals [21][1]. RSD performs the following three stages in order to propositionalize data: (1) identifies all first-order literal conjunctions that by definition form a first-order feature, and at the same time comply to user-defined mode-language constraints. Such features do not contain any constants and the task can be completed independently of the input data; (2) employs constants by copying certain features several times with some variables substituted to constants chosen from the input data. In this step irrelevant features are also detected and eliminated; (3) generates a propositionalized representation of the input data using the generated feature set. Such representation is a table consisting of truth values of the first-order features computed for each example.

First-order feature construction. The feature language declarations accepted to RSD are very similar to those used by Aleph [19] and Progol [11]. Thus, in the declaration section the predicates that can appear in a feature are listed. A type and a mode are assigned to each argument of these predicates. If two arguments have different types they can not hold the same variable. A mode is either input or output. Input arguments are labelled by the + sign, and output variables by the − sign. Every variable in an input argument of a literal must appear in an output argument of some preceding literal in the same feature. Others setting parameters such as the maximum length of a feature (number of contained literals), maximum variable depth [11], maximum number of occurrences of a given predicate symbol among others, can be specified or acquire a default value.

RSD generates an exhaustive set of features satisfying the language declarations. A connectivity requirement, which stipulates that no feature may be decomposable into a conjunction of two or more features, must also be satisfied.

RSD implements several pruning techniques to reduce the number of examined expressions, while preserving the exhaustiveness of the resulting feature set. Such techniques may often drastically decrease the run times needed to achieve the feature set.

Employing constants and filtering features. In this step RSD substitutes selected variables in the features with constants extracted from the input data

[1] RSD is publicly available at http://labe.felk.cvut.cz/~zelezny/rsd

using the declared predicate *instantiate/1*. When such predicate appears in a feature having a variable as its arguments it means that all occurrences of that variable should be eventually substituted with a constant. In case of multiple instantiate/1 predicates appear in a single feature with different variables, a number of features are generated , each one corresponding to a possible combination of grounding of the indicated variables. Only those groundings which make the feature true for at least a pre-specified number of individuals are considered.

The feature filtering is performed during the feature construction process described above. Therefore, RSD discard features considering three constraints: (a) no feature should have the same Boolean value for all the examples, (b) no two features should have the same Boolean values for all the examples and (c) no feature should be true for less than a minimum prescribed number of examples.

Generating a propositional representation. After constructing an appropriate set of features RSD can use such features and the examples to generate a single relational table using an attribute-value representation. For more details about RSD we refer the reader to [21].

3 Stochastic Local Search

Stochastic Local Search (SLS) algorithms have been used to solve hard combinatorial problems such as satisfiability. SLS algorithms are characterised by the following properties [14]: (a) they are search algorithms, i.e. given a problem P they search through an instance space S_P for instances $i_P \in S_P$ which might be solutions; (b) They perform a local search. That means during their search they only consider instances which are direct neighbors of the current instance according to a neighborhood relation $R \subseteq S_P \times S_P$; (c) they use a global scoring function $score_P : S_P \times S_P \mapsto R$. The decision on which instance should be examined next depends – at least partially – on the scoring function.

SLS algorithms such as GSAT [18] and WalkSAT [17] have been successfully used to solve challenging satisfiability problems. They have also been applied on propositional tasks encoded as a satisfiability problem, substantially improving their efficiency [2]. Next section brings a brief review of these algorithms.

3.1 GSAT and WalkSAT Algorithms

GSAT is based on a hill-climbing procedure with a stochastic component. It searches for a truth assignment which satisfies a set of propositional clauses. The basic GSAT algorithm starts with a randomly generated assignment and then repeatedly changes ("flips") the assignment of a single variable that leads to the largest decrease in the number of unsatisfied clauses. These flips continue until either a satisfying assignment is found or a pre-specified maximum number of flips is reached. GSAT can easily become trapped in local minima and the only way employed to it to escape from them is restart with a new randomly generated assignment after reaching the maximum number of flips. The process is repeated until a pre set number of tries is reached.

Another mechanism for escaping from local minima is to randomly alternate between greedy minimizing moves and stochastic moves, randomly selected from the variables which appears in unsatisfied clauses. Therefore, GSAT with Random Walk [16], with probability p, takes a random variable from an unsatisfied clause and flips its value and with probability $1-p$ follows the schema of GSAT, changing the value of the variable which minimizes the number of unsatisfied clauses the most.

WalkSAT is derived from GSAT with Random Walk, but including significant modifications. Different from the later, which maintains a list of variables appearing on unsatisfied clauses and picks a variable at random from that list, WalkSAT employs a two-steps random process: first, it picks randomly a clause not satisfied by the current assignment and then it picks a variable, at random or using a greedy heuristic, within that clause to flip. Another modification is related to the scoring function. Instead of considering the overall decrease of unsatisfied clauses, it counts the number of clauses which will become unsatisfied if each variable in the clause chosen at random is flipped.

3.2 Stochastic Local Search in k-Term DNF Learning

The aim in k-term DNF learning is to induce a formula of k terms in disjunctive normal form, where each term is a conjunction of literals. Formally, the k-term DNF learning can be defined in the following way [5]:

Given:

- a set of Boolean variables *Var*,
- a set *Pos* of truth value assignments $p_i : Var \rightarrow 0, 1$,
- a set *Neg* of truth value assignments $n_i : Var \rightarrow 0, 1$ and
- a natural number k

Find:

- a DNF formula with k terms
- that evaluates to 1(true) for all variable assignments in *Pos*
- and evaluates to 0 (false) for all variable assignments in *Neg*.

k-term DNF learning is a NP-hard problem of combinatorial search. Therefore, SLS algorithms can be applied to solve it, sacrificing completeness for better runtime behavior. A novel SLS algorithm was designed in [15] to solve k-term DNF learning and it is reproduced here in Fig. 1.

The algorithm starts generating randomly a hypothesis, i.e., a DNF formula with k-terms and then refines this hypothesis in the following manner. First, it picks a misclassified example at random. If this example is a positive one the hypothesis must be generalized. To do so, a literal has to be removed from a term of the hypothesis. Now, with probability p_{g1} and p_{g2} respectively, the term and a literal in this term are chosen at random. Otherwise the term in the hypothesis which differs in the smallest number of literals from the misclassified example and the literal whose removal from the term decreases the score the most are

chosen. On the other hand, if the example is a negative one, it means that the hypothesis must be specified. Therefore, a literal has to be added in a term. The term is chosen at random from those ones which cover the misclassified negative example. In a similar way to the last case, either with probability p_s the literal to be added in this term is chosen at random or a random literal which decreases the score the most is taken. This iterative process continues until the score is equal to zero or the algorithm reaches a maximum number of modifications. All the procedure is repeated a pre-specified number of times.

$search(k, maxTries, maxSteps)$: Given integer numbers k,$maxTries$ and $maxSteps$; probability parameters p_{g1}, p_{g2} and p_s; a set of Examples E, returns a k-term DNF formulae

1. **for** $i \leftarrow 1$ to $maxTries$ **do**
2. $H \leftarrow$ a randomly generated k-term DNF formula;
3. $steps \leftarrow 0$;
4. **while** $steps < maxSteps$ **and** $score_L(H) \neq 0$ **do**
5. $steps \leftarrow steps + 1$;
6. $ex \leftarrow$ a random example $\in E$ that is misclassified by H;
7. **if** ex is a positive example
8. with probability p_{g1}: $t \leftarrow$ a random term in H;
9. otherwise: $t \leftarrow$ the term in H that differs in the smallest number of literals from ex
10. with probability p_{g2}: $l \leftarrow$ a random literal in t;
11. otherwise: $l \leftarrow$ the literal in t whose removal decreases $score_L(H)$ most;
12. $H \leftarrow H$ with l removed from t
13. **else if** ex is a negative example
14. $t \leftarrow$ a (random) term in H that covers ex;
15. with probability p_S: $l \leftarrow$ a random literal m so that $t \wedge m$ does not cover ex;
16. otherwise: $l \leftarrow$ a literal whose addition to t decreases $score_L(H)$ most
17. $H \leftarrow H$ with l added to t
18. **end if**
19. **end while**
20. **end for**

Fig. 1. An SLS algorithm for k-term DNF learning [15]

It is important to mention that SLS algorithm performs refinements of an entire hypothesis rather than a single rule. A detailed analysis of SLS performance compared to WalkSAT shows the advantages of using SLS to learn a hypothesis as short as possible [15].

3.3 Stochastic Local Search in ILP

The recent study in [22] compared the performance of several randomized strategies (GSAT, WalkSAT, Randomized General to Specific, Rapid Random Restarts) to search the ILP subsumption lattice. All these methods can be viewed

search$(B, H, E, s^{suf}, c^{all}, \gamma)$: Given background knowledge B; a set of clauses H; a training sequence $E = E^+, E^-$ (i.e. positive and negative examples); a sufficient clause score s^{suf} $(-\infty \leq s^{suf} \leq \infty)$; the maximum number of clauses the algorithm can evaluate c^{all}, $(0 < c^{all} < \infty)$; and the maximum number of clauses evaluated on any single restart or the 'cutoff' value γ $(0 < \gamma \leq \infty)$, returns a clause D such that $B \cup H \cup \{D\}$ entails at least one element e of E^+. If fewer than c^{all} clauses are evaluated in the search, then the score of D is at least s^{suf}.

1. $S := -\infty$; $C := 0$; $N := 0$
2. repeat
3. **Select** e^{sat} from E^+
4. **Select** D_0 such that $D_0 \succeq_\theta \bot(e^{sat}, B)$
5. $Active = \emptyset$; $Ref = \{D_0\}$
6. repeat
7. $S^* = \max_{D_i \in Ref} \underline{eval}_{B,H}(D_i)$; $D^* := \arg\max_{D_i \in Ref} \underline{eval}_{B,H}(D_i)$
8. if $S^* > S$ then $S := S^*$; $D := D^*$
9. $N := N + |Ref|$
10. $Active :=$ **UpdateActiveList**$(Active, Ref)$
11. $Prune :=$ **Prune**$(Active, S^*)$
12. $Active := Active \setminus Prune$
13. **Select** D^{curr} from $Active$; $Active := Active \setminus D^{curr}$
14. $Ref :=$ **Refine**$_{B,H,(\gamma-N)}(D^{curr})$
15. until $S \geq s^{suf}$ or $C + N \geq c^{all}$ or $N = \gamma$
16. $C := C + N$; $N := 0$
17. until $S \geq s^{suf}$ or $C \geq c^{all}$
18. if $S = -\infty$ then return e^{sat} else return D^*.

Fig. 2. A general skeleton of a search procedure—possibly randomized and/or restarted—in the clause subsumption lattice bounded by the clause $\bot(e^{sat}, B)$. This clause is derived using the saturant e^{sat} and the background knowledge B. In Step 4, $\succeq_\theta$ denotes Plotkin's (theta) subsumption between a pair of Horn clauses. Individual strategies considered in this paper are obtained by different implementations of the bold-typed commands. Clauses are scored by a finite evaluation function $\underline{eval}$. Although in the formal notation in Step 7 the function appears twice, it is assumed that the 'max' and 'arg max' operators are computed simultaneously. In Step 11 **Prune** returns all elements of *Active* that cannot possibly be refined to have a better score than S^*. If the number of refinements of the current clause is greater than $(\gamma - N)$, **Refine** returns only the first $(\gamma - N)$ computed refinements, to guarantee that no more than γ clauses are evaluated between restarts. The search is terminated when score s^{suf} is reached or c^{all} clauses have been evaluated, and restarted (from Step 3) when γ clauses have been evaluated since the last restart. If all **Select** commands are deterministic then restarting (setting $\gamma < c^{all}$) results in mere repetitions of the identical search.

as variations of the basic skeleton in Fig. 2, by instantiating the bold-faced commands.

It was observed that if a near-to-optimal value of the cutoff parameter (the number of clauses examined before the search is restarted) is used, then the

mean search cost (measured by the total number of clauses explored rather than by cpu time) may be decreased by several orders of magnitude compared to a deterministic non-restarted search. It was also observed that differences between the tested randomized methods were rather insignificant. In the present study we accept the GSAT strategy for sakes of comparison. In terms of the algorithm in Fig. 2, GSAT has the following properties: randomized saturant (example seed) and start clause selection, greedy updating of the active list (only the best scoring neighbor state is retained), deterministic next state selection (determined by the scoring function), no pruning, and bidirectional refinement (combining specialization and generalization).

A limitation of the study in [22] was that the stochastic strategies were framed in a single clause search algorithm. One consequence of this is that the statistically assessed performance ranking of individual strategies may not be representative of their performance when used for an incremental entire-theory construction due to the statistical dependence between the successive clause search procedures. Thus in the present study we compare performance measured for entire-theory construction processes.

3.4 Stochastic Local Search in ILP Through Propositionalization

In this work we investigate the relevance of using stochastic local search to learn k-term DNF formulae in relational domains and to do so we have to propositionalize the relational problem. Therefore, we implemented a k-term DNF formulae inducer using the SLS algorithm joined to the first-order feature construction part of the RSD system. We compare the run-time when performing stochastic local search through propositionalization with the run-time when doing stochastic search or enumerative heuristic search directly in the relational space.

4 Experiments

4.1 Data and Methods Tested

The goals discussed before in this paper are experimentally evaluated using two ILP benchmarks: the East-West Trains [10] and Mutagenesis Data [20].

Methods:

- **Aleph** [19]. Working in its default mode, with the following exceptions. For the Mutagenesis data set, negative examples were allowed to be covered by the constructed theory while the minimum accuracy of each rule included in the theory was set to 0.7. For both data sets, the total number of nodes searched within each rule-search procedure was not limited by a constant. It was rather determined by setting the maximum number of literals in each rule (the *clauselength* parameter), ranging from 3 to 7 in Mutagenesis and 3 to 12 in East-West Trains.

- **Aleph/SLS.** This method uses the GSAT [22] stochastic local search procedure implemented in Aleph. The same settings as for the previous method were applied and additionally, the number of *tries* (randomly initiated local searches) was set to 10 and the number of *moves* (clauses explored in each local search) was set to 100.
- **DNF/SLS.** This method first propositionalizes the relational data with RSD [21] using the same language declarations as used in Aleph [2] and then applies the stochastic k-term DNF search procedure as described in [15] onto the propositional representation of the data.
- **Propositional algorithms.** In order to observe the behavior of stochastic local search in relational problems through propositionalization when the search is performed in the whole theory instead of individual clauses, we also compared DNF/SLS with two popular rule learner algorithms which use the covering approach, Part [4] and Ripper [3] with pruning disabled since we are interested in decreasing the runtime.

4.2 Experimental Procedure

The aim of the experiments was to determine the statistical dependencies between three random variables:

- N: the number of rules contained in the resulting theory (the compression achieved)
- T: the cpu time consumed by the search
- A: the estimated predictive accuracy of the resulting theory

We specifically model the two dependencies N vs. T and N vs. A.

We first split both datasets into 10 cross-validation folds, and for all methods, A was estimated on the independent test data part in each fold.

For Aleph and Aleph/SLS, all the three variables acquire different values as a result of:

- applying the method on each of the 10 cross-validation folds
- within each fold, varying the maximum *clauselength* parameter within its limits as mentioned above.

The DNF/SLS method allows to directly specify k, the number of rules (i.e. the number of DNF terms) in the constructed theory. Also Part and Ripper were modified to allow a specified maximum number of rules in the theory. Upon executing the Aleph and Aleph/SLS experiments, we extracted the

[2] Despite the same declarations, the theory spaces explored by the two approaches are necessarily different. On one hand, only a fraction of clauses explored by Aleph form a correct feature (as defined e.g. in [21]). On the other hand, the Aleph setting of maximum number of literals in a rule here translates into the maximum number of literals in a feature; however, in the subsequent k-DNF search, the features are combined in conjunctions and the total length of a single rule thus is unbounded.

range of the values of N from the set of resulting theories and executed repeatedly the DNF/SLS method with k acquiring all values in the detected range. This procedure was further repeated for each of the 10 cross-validation folds.

4.3 Results

Figures 3 and 4 present the empirical N vs. T dependency by plotting the mean cpu-time (on logarithmic scale) consumed by search executions resulting in a theory containing a given number of rules. This is shown for both data sets and all three methods. For DNF/SLS, we separately plot the k-DNF learning time excluding/including time consumed by propositionalization.

Figures 5 and 6 present the empirical N vs. A dependency by plotting the mean estimated predictive accuracy achieved by search executions resulting in a theory containing a given number of rules. This is shown for both data sets and all three methods.

4.4 Principal Trends

The results indicate the following trends. DNF/SLS performs faster w.r.t. all other tested methods when it comes to short theories (in number of rules). This results change when allowing an increasing number of rules in the theory, with standard Aleph being ultimately the fastest algorithm if propositionalization time is taken into account for DNF/SLS. Comparing to relational methods, the performance gap is significantly large (in orders of magnitude), while corresponding predictive accuracy do not favor either SLS/DNF or the relational methods. Comparing to the propositional methods, this performance gap is much smaller, while SLS/DNF's short theories exhibit slight superiority in terms of predictive accuracy. Note however, that e.g. in the East-West Trains domain, each 9-rule theory produced by standard Aleph is a trivial list of the positive examples[3], which is not the case for the other two methods (This fact can be verified from Fig. 5 where Aleph's 9-rule theories achieves accuracy 0.5. in the East-West Trains domain.)

In general, the size of theories (in the number of rules contained) does not demonstrate any significant influence onto their predictive accuracy. This is not surprising for the East-West trains data set, where the small number of examples (10 of each class) renders all theories almost random fits of the training data. For Mutagenesis, an 'Occam's razor' intuition would suggest that higher accuracy should be expected from a theory with fewer rules, given the same required performance on the training data. A possible explanation of the observation is that this classification problem is well modelled by a theory with a large number of short rules (few literals). Our next experiments will thus address classification problems requiring complex rules (such as the graph classification challenge described in [22]) and where stochastic search have shown to play an important role as in relational phase transition benchmarks [1].

[3] 9 of the 10 positive examples fall in the training split in each cross-validation fold.

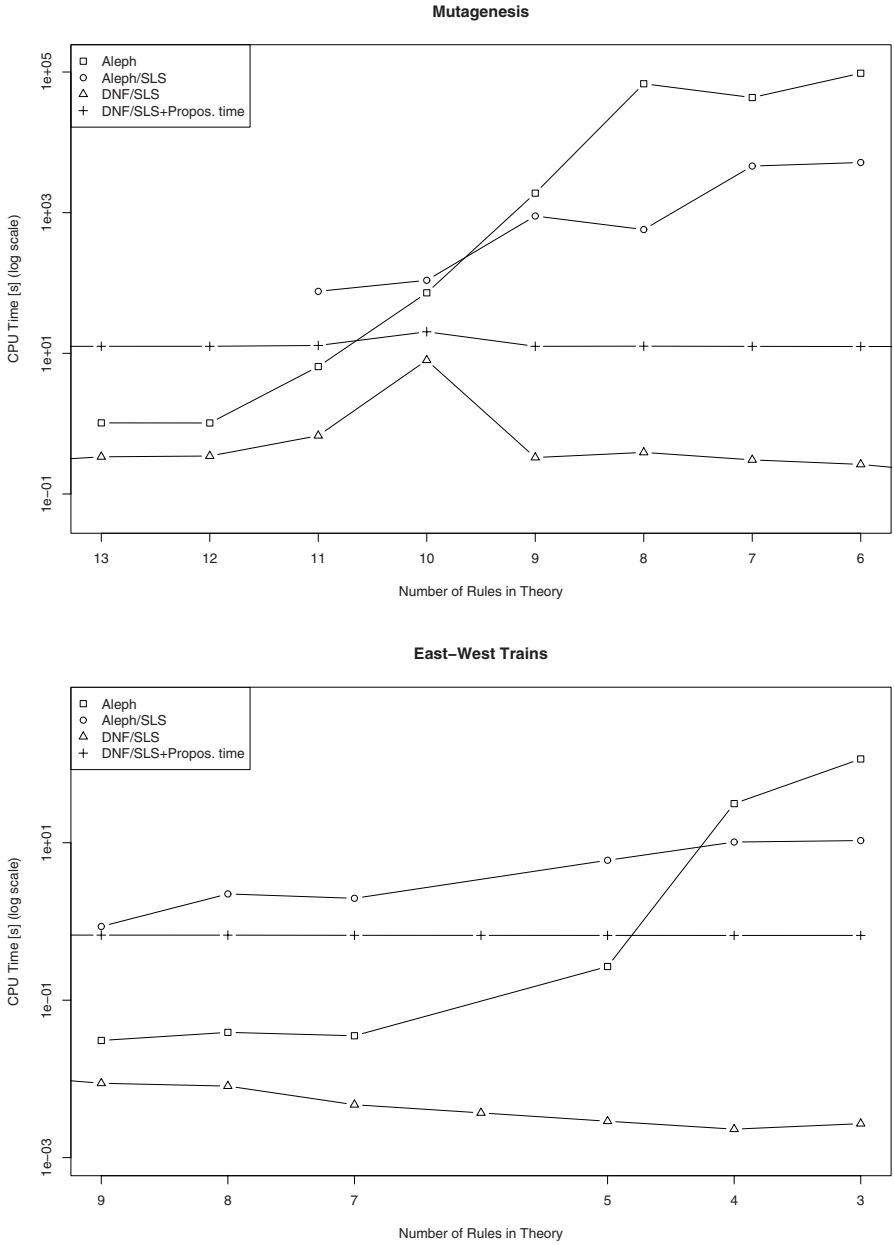

Fig. 3. Mean cpu-time (on logarithmic scale) consumed by search executions resulting in a theory containing a given number of rules: comparing to relational learners

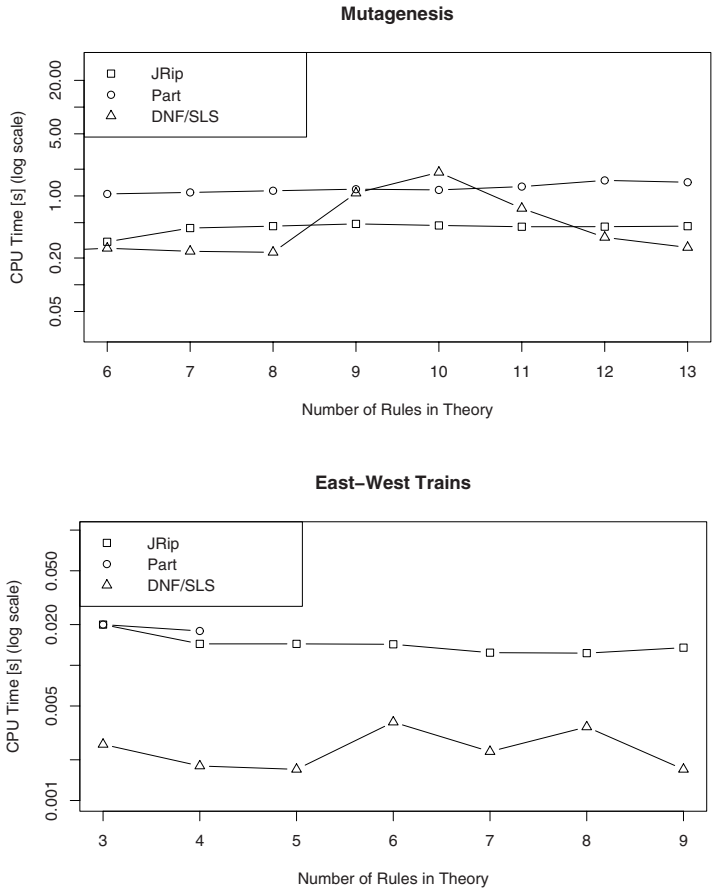

Fig. 4. Mean cpu-time (on logarithmic scale) consumed by search executions resulting in a theory containing a given number of rules: comparing to propositional learners

5 Conclusions

In this work we investigated the performance of stochastic local search in ILP using a propositionalized form of relational problems. To do so, we experimentally compared standard rule learners, standard and stochastic ILP system to a SLS algorithm which searches for k-terms DNF formulaes. The results indicated that DNF/SLS performs faster than all other tested methods when it comes to short theories (in number of rules). Two main observations follow: (1) a very significant speed-up was achieved by using SLS/DNF search on the propositionalized form of the learning data, as compared to the default enumerative search conducted by Aleph, (2) the k-term SLS run-time distribution exhibits a rapid decay, unlike the heavy-tailed clause-search run-time distributions we observed in the relational domain [22]. When comparing to standard propositional rule learners the gap performance is much smaller. Therefore our future work will extend

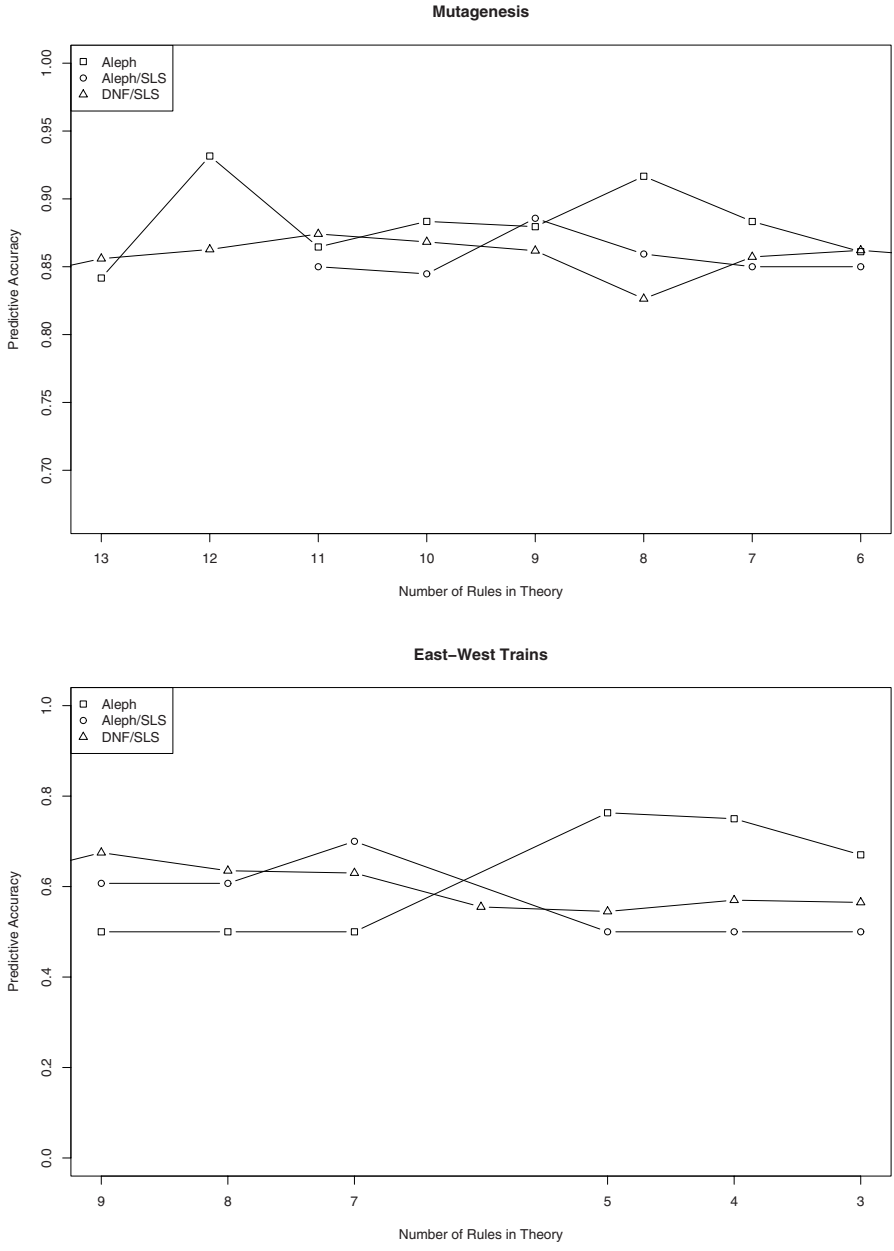

Fig. 5. Mean estimated predictive accuracy achieved by search executions resulting in a theory containing a given number of rules: : comparing to a relational learner

experiments to a larger set of ILP benchmarks and relational phase transition datasets [1] to achieve a more conclusive ranking between the DNF and greedy strategies in the propositionalized domain.

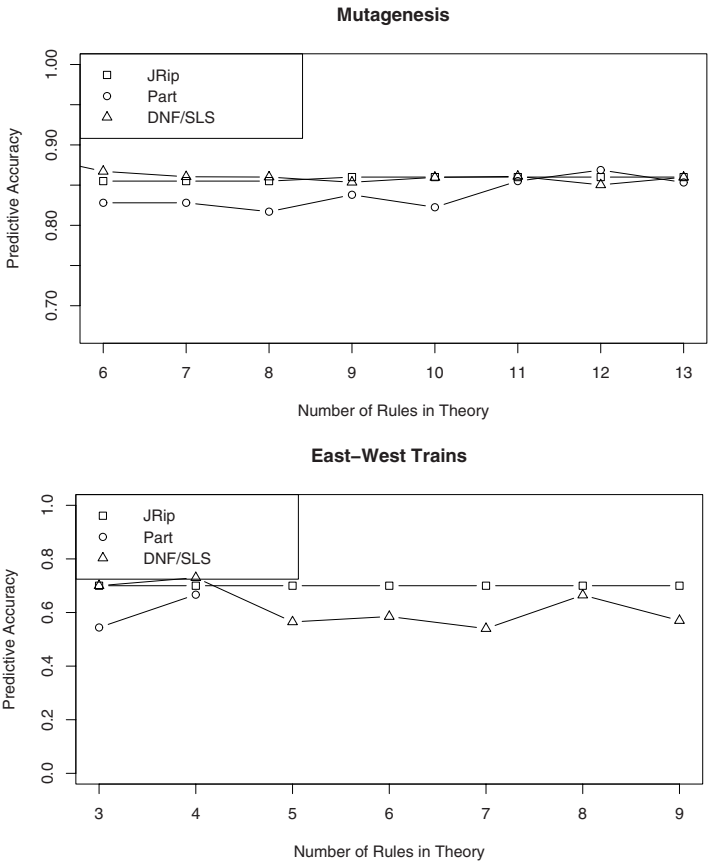

Fig. 6. Mean estimated predictive accuracy achieved by search executions resulting in a theory containing a given number of rules: comparing to propositional learners

Acknowledgements

The first and second authors are financially supported by the Brazilian research agencies CAPES and CNPq, respectively. The second author is Supported by the Czech Academy of Sciences through the project 1ET101210513 Relational Machine Learning for Biomedical Data Analysis. The authors would like to thank Ulrich Rückert and Stefan Kramer for giving us their SLS code.

References

1. Botta, M., Giordana, A., Saitta, L., Sebag, M.: Relational learning as search in a critical region. J. Mach. Learn. Res. 4, 431–463 (2003)
2. Chisholm, M., Tadepalli, P.: Learning decision rules by randomized iterative local search. In: Proc. of the 19th ICML, pp. 75–82 (2002)
3. Cohen, W.W.: Fast effective rule induction. In: Proc. of 12th ICML, pp. 115–123. Morgan Kaufmann, San Francisco (1995)

4. Frank, E., Witten, I.H.: Generating accurate rule sets without global optimization. In: Proc. of 15th ICML, pp. 144–151. Morgan Kaufmann, San Francisco (1998)
5. Kearns, M.J., Vazirani, U V.: An Introduction to Computational Learning Theory. Cambridge, Massachusetts (1994)
6. King, R.D., Whelan, K.E., Jones, F.M., Reiser, P.K.G., Bryant, C.H., Muggleton, S.H., Kell, D.B., Oliver, S.G.: Functional genomic hypothesis generation and experimentation by a robot scientist. Nature 427, 247–252 (2004)
7. Krogel, M.-A., Rawles, S., Železný, F., Flach, P.A., Lavrac, N., Wrobel, S.: Comparative evaluation of approaches to propositionalization. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 197–214. Springer, Heidelberg (2003)
8. Lavrač, N., Džeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood (1994)
9. Lavrač, N., Flach, P.A: An extended transformation approach to inductive logic programming. ACM Trans. on Comp. Logic 2(4), 458–494 (2001)
10. Michalski, R., Larson, J.B.: Inductive inference of vl decision rules. Workshop in pattern-Directed Inference Systems, SIGART Newsletter 63, 38–44 (1977)
11. Muggleton, S.: Inverse entailment and progol. New Generation Computing Journal 13, 245–286 (1995)
12. Muggleton, S.: Scientific knowledge discovery using inductive logic programming. Communications of the ACM 42(11), 42–46 (1999)
13. Page, D., Srinivasan, A.: Ilp: A short look back and a longer look forward. Journal of Machine Learning Research 4, 415–430 (2003)
14. U. Rückert. Machine learning in the phase transition framework. Master's thesis, Albert-Ludwigs-Universität Freiburg (2002)
15. Rückert, U., Kramer, S.: Stochastic local search in k-term dnf learning. In: Proc. of the 20th ICML, pp. 648–655 (2003)
16. Selman, B., Kautz, H.A.: Domain-independent extensions to gsat: Solving large structured satisfiability problems. In: Proc. of the 13th IJCAI, pp. 290–295 (1993)
17. Selman, B., Kautz, H.A., Cohen, B.: Local search strategies for satisfiability testing. In: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, pp. 521–532 (1996)
18. Selman, B., Levesque, H.J., Mitchell, D.G.: A new method for solving hard satisfiability problems. In: Proc. of the 10th AAAI, pp. 440–446 (1992)
19. Srinivasan, A.: The Aleph Manual (2001), `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html`
20. Srinivasan, A., Muggleton, S., Sternberg, M.J.E., King, R.D.: Theories for mutagenicity: A study in first-order and feature-based induction. Artificial Intelligence 85(1-2), 277–299 (1996)
21. Železný, F., Lavrac, N.: Propositionalization-based relational subgroup discovery with RSD. Machine Learning 62(1–2), 33–63 (2006)
22. Železný, F., Srinivasan, A., Page, D.: A monte carlo study of randomised restarted search in ILP. In: Camacho, R., King, R., Srinivasan, A. (eds.) ILP 2004. LNCS (LNAI), vol. 3194, Springer, Heidelberg (2004)