REDA-ILP: Learning Theories using EDA and Reduced Bottom Clauses

Cristiano Grijó Pitangui[1], Gerson Zaverucha[1]

[1]Federal University of Rio de Janeiro /COPPE/PESC
Rio de Janeiro – RJ – Brazil
{cris_pi, gerson}@cos.ufrj.br

Abstract. In our previous work we have introduced EDA-ILP, an Inductive Logic Programming (ILP) system based on Estimation Distribution Algorithm (EDA). EDA-ILP showed to be superior when compared to GA-ILP, a variation of EDA-ILP created replacing the EDA by a "conventional" Genetic Algorithm. Additionally, EDA-ILP proved to be very competitive when compared to the state of the art ILP system Aleph. This work presents REDA-ILP, an extension of EDA-ILP that employs the well-known Reduce algorithm in order to considerably reduce the search space. Preliminary results show that REDA-ILP, when compared to EDA-ILP, achieves equivalent accuracies results while considerably reduces the search time in order to find simpler theories.

Keywords: Inductive Logic Programming, Estimation Distribution Algorithm, Reduce Algorithm.

1 Introduction

Estimation of Distribution Algorithms (EDAs) [3] are algorithms based on the explicit use of probability distributions. These algorithms completely (or partially) replace the traditional variation operators of the Genetic Algorithms (GAs) [6], such as mutation and crossover, by building a probabilistic model of promising solutions and sampling the built model to generate new candidate solutions.

In our previous work [1], we have introduced EDA-ILP, an Inductive Logic Programming (ILP) [2] system based on EDA. Briefly, EDA-ILP searches for a whole theory [18] instead of single clauses, but limits its search to the clauses whose bodies are subsets of the literals in the bottom clauses. EDA-ILP uses Bayesian Networks (BNs) [4] whose structures captures the dependences of the literals in the bottom clauses and applies PBIL's [5] update rule in order to guide the search towards promising areas in the search space. To evaluate EDA-ILP, we also built GA-ILP replacing the EDA in EDA-ILP by a "conventional" GA. EDA-ILP showed to be superior when compared to the GA-ILP and proved to be very competitive when evaluated against the Aleph [8] in two classical datasets.

In the present work, we introduce an extension of EDA-ILP, the REDA-ILP, which, in short, uses the Reduce [7] algorithm to considerably reduce the search space. REDA-ILP was inspired by the QG/GA (Quick Generalization/Genetic Algorithm) presented in [10]. In such work, the QG algorithm rapidly builds consistent clauses using the Reduce algorithm while a GA is used to evolve the clauses initially found by the QG. QG/GA was plugged into the Progol to create several variations of such system. The Progol variation that specifically motivated this

work is the one that used both the GA and the QG. In this version of Progol, called here by Progol-QG/GA, a GA was used in substitution to the standard A* while the QG was used to seed the initial population of the GA. Despite of being motivated by the cited variation of Progol, REDA-ILP holds some fundamentals differences when compared to the Progol-QG/GA. These differences are presented bellow.

i) REDA-ILP searches for theories whose literals are subsets of the reduced clauses (clauses that undergo to the Reduce algorithm). In Progol-QG/GA, the search for clauses is done in the complete bottom clauses, i.e., the Reduce algorithm is just applied to seed the initial population of the GA, but its search considers the entire search space defined by the bottom clause.
ii) Progol-QG/GA (and all other variations created using only QG or GA) use the classical covering procedure in order to build a theory, this way, all the variations of the Progol system search for one clause at a time while the REDA-ILP, following EDA-ILP, searches for a whole theory.
iii) The Progol-QG/GA uses a canonical Genetic Algorithm to perform its search while the REDA-ILP uses an Estimation Distribution Algorithm.

In order to evaluate REDA-ILP, it was compared to the EDA-ILP and to the Aleph. In relation to EDA-ILP, preliminary results points that REDA-ILP achieves equivalent accuracies results while considerably reduces the search time in order to find simpler theories. In relation to Aleph, REDA-ILP proved to be very competitive.

This paper is organized as follows. Section 2 briefly reviews the QG/GA. Section 3 presents the REDA-ILP and relates it to our previous work. Preliminary experiments are presented in section 4. Finally, the section 5 concludes and discusses further works.

2 Related Works

2.1 The QG\GA

Motivated by the low average density of consistent clauses in relation to inconsistent ones that are evaluated during the Progol's [11] search in several datasets, the work [10] introduced the stochastic search called Quick Generalization/Genetic Algorithm and plugged it to the ILP system Progol. The QG algorithm has the objective to rapidly build consistent clauses (that are typically found in the fringe of the refinement graph) without needing to explore this graph in detail. Briefly, the QG algorithm receives a bottom clause, randomly permutes its literals to build a permuted head-connected bottom clause, and then applies the Reduce [7] algorithm to this permuted clause. As the output, the QG algorithm returns a reduced consistent clause built from the input bottom clause.

First, the work evaluates the Progol standard search (that is performed by the A*) against the Progol using the QG, and founds that the configuration Progol-QG obtains solutions with the same or similar accuracies in less time. Latter, the work considers two variations of Progol using a GA instead of the A*. In the first variation, called here by Progol-GA, the initial population of the GA is randomly generated, while in the second variation of the system, called here by Progol-QG/GA, the initial population of the GA is seeded using the clauses returned by the QG algorithm. In order to evaluate Progol-GA and Progol-QG/GA, two batches of experiments were made. First, Progol-GA was compared to Progol-A* and to Progol-QG, latter, Progol-QG/GA was compared to Progol-A*, Progol-QG, and Progol-AG.

The first batch of experiments was performed using standard datasets, while the second batch was performed both in standard datasets as well on artificial generated phase transitions data. In the first batch of tests made, the QG/GA achieved higher accuracy results (but not statistically significant) using fewer evaluated clauses in relation to Progol-A* and Progol-GA. On the second batch of experiments, the Progol-QG/GA clearly tends to overcome the other variations of the system as the size of the concepts to be learned increase.

3 The REDA-ILP System

As the search performed by REDA-ILP is done for a whole theory instead of a single clause, each individual in the population codifies a set of clauses (a theory). REDA-ILP adopts the bottom clauses as one of its search bias, i.e., REDA-ILP searches for clauses whose literals are subsets of the literals in a given bottom clause. To do so, REDA-ILP uses the Aleph system to generate the bottom clauses and uses a binary string to represent its individuals. A binary string is constructed over the bottom clause and both have the same size, i.e., the binary string has as many bits as the number of literals of the bottom clause. During the search of the REDA-ILP, a bit with value 1 at the *i*th position of the binary string represents that the *i*th literal of the bottom clause is being used, while 0-valued *i*th position represents that the *i*th literal of the bottom clause is not used. This form of encoding is found in [9] and [10]. The next example illustrates this form of codification.

Example 1: Assume that the h(A,B) :- p(A,C), q(B,C), r(C,D) was the bottom clause generated. Fig. 1 shows a possible clause generated by the system.

BC	h(A,B) :-	p(A,C)	q(B,C)	r(C,D)
S1_bin	1	0	0	1
C1	h(A,B) :-			r(C,D)

Fig. 1. A possible clause generated by the EDA-ILP system

From fig 1 one can see that the binary mapping can be easily done by looking at the positions of the binary string and its corresponding literals in the bottom clause. Thus, S1_bin (1001) transformed into clause will result in h(A,B) :- r(C,D).

The probabilistic model used by REDA-ILP was inspired by [12]; however, the current version of REDA-ILP uses BNs as the only search mechanism (differently from [12], which uses a couple of BNs to generate a clause that will be used as a seed to a local search procedure -- for more details, see [1]). As occurred in [12], the BNs are responsible for storing the dependencies between the literals of a bottom clause. During the execution cycle of REDA-ILP, these Bayesian Networks are sampled to generate binary strings that subsequently will be mapped into clauses. As in [12], the structures of Bayesian networks in REDA-ILP do not change during the course of the search; however, the Conditional Probabilities Tables (CPTs) are updated in order to guide the Bayesian networks towards promising areas of the search space. REDA-ILP updates its CPTs using PBIL's update rule. Briefly, PBIL's update rule selects the best individual from the current population and updates its probabilistic model in order to increase the probability to generate this best individual in the next generation. In the current version of the system, the number of the clauses in a theory is provided by the user, thus, given that the user has set the number of the clauses in a theory to k, the REDA-ILP will construct k bottom clauses and codifies each clause into a binary string. In addition, for each bottom clause constructed, REDA-ILP also builds a BN. Next, we present and discuss the main execution cycle of REDA-ILP.

```
Input parameters:
(k): number of clauses in theories; (p1, p2, p3): the initial probabilities
for the CPTs; (λ): learning rate for PBIL's update rule; (num_Pop): number
of individuals in the population; (num_Gen): number of generations.
REDA-ILP-Begin:
1 – Create k bottom clauses using k positive examples randomly chosen.
2 – Apply the Reduce algorithm to each bottom clause created to obtain k
    reduced bottom clauses.
3 – Create k Bayesians networks (one for each reduced bottom clause);
4 - Repeat (num_Gen) times
    5 - Generate num_pop individuals (theories) by sampling all k
    Bayesians networks num_Pop times;
    6 - Evaluates all num_pop individuals with the fitness function;
    7 - Select the fittest individual in the population and updates the
    CPTs of all the k Bayesian networks using the selected individual;
    End-Repeat.
8 - Generate num_pop individuals and returns the fittest one.
End-REDA-ILP.
```

Step 1 creates k bottom clauses by randomly selecting k positive examples. In step 2, REDA-ILP applies the Reduce algorithm to each bottom clause generated in order to obtain k reduced bottom clauses. Step 3 build k BNs (one for each reduced bottom clause). As said, these BN captures the dependencies between the literals of the bottom clauses. Step 5 generates num_pop individuals by sampling all the BNs num_pop times. This step is performed as follows: to generate one individual containing k clauses, the first BN is sampled (resulting in the first clause of the individual), next, the second BN is sampled (resulting in the second clause of the individual) and so on, until sampling the *k*th BN to generate the *k*th clause in the individual. To generate a population with num_pop individuals, this whole process is repeated num_pop times. The step 6 evaluates the population with a predefined fitness function (accuracy, for example). The step 7 updates all the BNs. To update each one of the k BNs, all the k clauses from the best individual are sequentially used, thus, the first clause in the best individual is used to update the first BN, the second clause of the best individual is used to update the second BN, and so on, until using the *k*th clause in the best individual to update the *k*th *BN*. Step 8 generates a new population and returns the fittest individual as the learned theory.

The fundamental difference between REDA-ILP and EDA-ILP is highlighted as the step 2. While EDA-ILP directly generates k BNs from the k bottom clauses created, the REDA-ILP applies the Reduce algorithm to each one of the k bottom clauses created, before constructing its BNs. This way, one can see that the search space of the REDA-ILP is smaller when compared to the EDA-ILP, since the reduced bottom clauses have fewer literals when compared to the bottom clauses that were not reduced.

4 Preliminary Experiments

Materials and Methods: In order to empirically evaluate REDA-ILP, we used two datasets, namely, carcinogenesis (Carc) [13] and alzheimers-amine (Alz) [14], since they are good examples of practical ILP problems. All the systems were evaluated in relation to the accuracy (using the corrected two-tailed paired t-test [15] with p<

0.05), and the complexity (number of clauses) of the induced theory; in addition, the systems were compared in relation to its execution times (measured in seconds). All experiments were performed using stratified 10-fold cross-validation and the results presented for each system are the average (on the test set) of the results in these 10 folds[1].We used stratified 10-fold internal cross-validation to set the parameters of the EDA-ILP. For REDA-ILP, since its search space is considerably reduced in relation to EDA-ILP, we simply reduced the generation number by half and doubled the learning rate of the PBIL's update rule[2]. The configurations of EDA-ILP and REDA-ILP are presented in table 1. Both EDA-based systems used accuracy as the fitness function. For Aleph, the configuration was taken from [16] and [17] since these works suggest good values of parameters for Aleph[3]. All the experiments were performed in a Dual Core Pentium 2.0 GHz with 2.0 GB of RAM. All systems used Yap latest version (6.2.1).

Table 1. Configuration of EDA-ILP and REDA-ILP

-	Alzheimers-amine		Carcinogenesis	
	EDA-ILP	REDA-ILP	EDA-ILP	REDA-ILP
Generation Number	500	250	100	50
Population Size	20	20	10	10
Number of Clauses in Theories	3	3	1	1
Learning rate (λ)	0.005	0.01	0.01	0.02
p1\p2\p3	0.5\0.5\0.1	0.5\0.5\0.1	0.1\0.1\0.1	0.1\0.1\0.1

Results and Discussion: Table 2 shows the obtained results, where (Acc) is the accuracy achieved, (#Cls) is the number of clauses in the theory, (#Lit) is the number of literals in the theory, and (T) is the execution time taken to execute the systems.

Table 2. Results for Aleph, EDA-ILP, and REDA-ILP

	Aleph				**EDA-ILP**				**REDA-ILP**			
	Acc.	#Cls.	#Lit.	T(s).	Acc.	#Cls.	#Lit.	T(s).	Acc.	#Cls.	#Lit.	T(s).
Alz.	69.7	7	3.4	56.6	73.5	3	7.2	45.7	71.8	3	4.2	16.6
Carc.	62.7	4.7	1.8	5.8	68.5	3	2.3	14	67.0	3	1.45	5.36

In our previous work [1] we concluded that the EDA-ILP is very competitive when compared to Aleph. In fact, for the both datasets, EDA-ILP obtained statistical significant accuracies results. The number of clauses in the theories obtained by EDA-ILP is smaller when compared to the Aleph, but the number of literals in EDA-ILP's theories is higher. In relation to execution time, EDA-ILP needs a little less time in Alz, but needs much more time in the Carc dataset. The accuracies obtained by REDA-ILP are equivalent to the accuracies of EDA-ILP for both datasets tested. Note, however, that REDA-ILP obtains simpler theories in both datasets. For Alz, the number of literals was reduced by 41% and for Carc by 37%. The execution time of REDA-ILP is also better when compared to the EDA-ILP. For Alz, the execution time reduced 63%, and for Carc, the reduction in time achieved 62%. By the presented results, one can see that REDA-ILP is more competitive with Aleph than EDA-ILP.

[1] Due to the stochastic nature of EDA-ILP, and REDA-ILP, the experimental results for each one of the 10 folds were obtained using the average of 10 runs in each fold.

[2] Preliminary tests showed that the results obtained by the EDA-ILP are considerably worse if its parameters are set as in REDA-ILP.

[3] Use the cross-validation procedure in order to set Aleph's parameters is hard task due to the large number of parameters of this system, this way, we opted to use the parameters suggested in these works, since the systems achieve good results when set as suggested.

5 Conclusions and Future Work

This work introduced REDA-ILP, an extension of EDA-ILP that uses Reduce algorithm to reduce the search space. REDA-ILP applies the Reduce algorithm in the bottom clauses generated then these reduced clauses are used to construct the Bayesian Networks that will perform the search using only the literals present in these reduced clauses. Preliminary results show that REDA-ILP, when compared to EDA-ILP, achieves equivalent accuracies results while considerably reduces the search time in order to find simpler theories. In addition, REDA-ILP proved to be very competitive when compared to Aleph.

Considering future works, one natural way to follow is to let the system use other literals of the bottom clause (not only those presented in the reduced bottom clause). In this respect, our actual line of research is to allow the literals that are in the reduced bottom clause to appear in the searched clauses with higher probabilities than the other literals of the bottom clause. Also, we would like to evaluate REDA-ILP against other systems, such as QG/GA, for example.

References

1. Pitangui, C., Zaverucha, G., Inductive Logic Programming Through Estimation Distribution Algorithm. To appear in proceedings of IEEE Congress of Evolutionary Computation (CEC-2011).
2. Muggleton, S., De Raedt, L. (1994); "Inductive Logic Programming: Theory and Methods", Journal of Logic Programming, v. 19, n. 20.
3. Mühlenbein, H., & Paaß, G. (1996); From recombination of genes to the estimation of distributions I. Binary parameters. Parallel Problem Solving from Nature, eds. Voigt, H.-M and Ebeling, W. and Rechenberg, I. and Schwefel, H.-P., LNCS 1141, Springer:Berlin, (pp. 178-187).
4. Pearl. J. (1988); Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
5. Baluja, S. (1994); Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Pittsburgh, PA: Carnegie Mellon University (Technical Report: CMU-CS-94-163).
6. Holland, J. (1975); *Adaptation in natural and artificial systems. MIT Press, Cambridge.*
7. Muggleton, S. H., & Feng, C. (1990). Efficient induction of logic programs. In Proceedings of the first conference on algorithmic learning theory (pp. 368–381). Tokyo: Ohmsha.
8. Srinivasan. A, The Aleph Manual, last access: 07/05/2011 http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/
9. Alphonse, E., Rouveirol, C., (2000), Lazy propositionalisation for Relational Learning, 14th European Conference on Artificial Intelligence 2000 (ECAI'00), pages 256-260, IOS Press.
10. Muggleton, S.,Tamaddoni-Nezhad A. (2006); QG/GA: A stochastic search approach for Progol. *Machine Learning*, 70(2-3):123-133, 2007. DOI: 10.1007/s10994-007-5029-3.
11. Muggleton, S. (1995); Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Progra mming, 13(3-4):245–286.
12. Oliphant, L., & Shavlik, J. (2007); Using Bayesian Networks to Direct Stochastic Search in Inductive Logic Programming. Proceedings of the 17th International Conference on Inductive Logic Programming, pages 191-199.
13. Srinivasan, A., King R.D. S.H. Muggleton S, and Sternberg M.. (1997); Carcinogenesis predictions using ILP. In Proceedings of the Seventh International Workshop on ILP, pages 273–287. Springer-Verlag, Berlin, LNAI 1297.
14. King R.D., Srinivasan A., and Sternberg M.J.E.. (1995) Relating chemical activity to structure: an examination of ILP successes. New Gen. Comp., 13:411–433, 1995.
15. Nadeau C., Bengio Y., (2003) "Inference for the Generalization Error", Machine Learning 52(3) pp. 239-281.
16. Huynh T., Mooney R. (2008); Discriminative Structure and Parameter Learning for Markov Logic Networks. In Proceedings of the 25th International Conference on Machine Learning (ICML-2008), Helsinki, Finland, pages 416-423.
17. Muggleton, S., Santos, J., Tamaddoni-Nezhad, A. TopLog: ILP using a logic program declarative bias. In *Proceedings of the International Conference on Logic Programming 2008*, LNCS 5366, pages 687-692. Springer-Verlag, 2010.
18. Bratko, I., (1999), Refining complete hypotheses in ILP. Proc. ILP'99 (9th Int. Workshop on Inductive logic programming), Bled, Slovenia, June 1999 (Lecture notes in computer science, Lecture notes in artificial inteligence, 1634). Berlin: Springer, 1999, pp. 44-55.