Inductive Logic Programming Through Estimation of Distribution Algorithm

Cristiano Grijó Pitangui
Federal University of Rio de Janeiro / COPPE
Federal University of Vales do Jequitinhonha e Mucuri / Decom
Rio de Janeiro – RJ – Brazil, Diamantina – MG – Brazil
cris_pi@cos.ufrj.br

Gerson Zaverucha*
Federal University of Rio de Janeiro / COPPE
Rio de Janeiro – RJ – Brazil
gerson@cos.ufrj.br

***Abstract* - Genetic Algorithms (GAs) are known for their capacity to explore large search spaces and due to this ability, they were to some extent applied to Inductive Logic Programming (ILP) problem. Although Estimation of Distribution Algorithms (EDAs) perform better in most problems when compared to standard GAs, this kind of algorithm have not been applied to ILP. This work presents an ILP system based on EDA. Preliminary results show that the proposed system is superior when compared to a "standard" GA and it is very competitive when compared to the state of the art ILP system Aleph.**

Keywords - Inductive Logic Programming; Estimation of Distribution Algorithms; Probabilistic Models.

I. INTRODUCTION

Inductive Logic Programming (ILP) [6, 7] is an area of artificial intelligence, which investigates the inductive construction of theories of Horn clauses of first-order logic from examples and a preliminary knowledge. In general, the goal of an ILP system is to induce concepts from examples and from a preliminary knowledge, both expressed by Horn clauses. ILP is applied to several important domains [8, 9], as protein structure prediction, detection of traffic problems, natural language processing, and many others. Due to the importance of the problem, a wide variety of ILP systems were proposed; some important ones are: Progol [10, 11], FOIL [12], Tilde [5] and Aleph [13].

Learning hypothesis in first-order logic is a highly complex task due to the extensive size of the search space, this way, several ILP systems based on Genetic Algorithms (GAs) [14, 15], [16], [17] [18] were designed to be applied to this problem, since such algorithms are known for their ability to explore large search spaces. The basic principles of GAs were presented rigorously by Holland [1] and subsequently described in many studies, such as [2] and [3]. In short, GAs are stochastic optimization methods inspired by natural evolution and genetics.

The intrinsic difficulty of the ILP problem has claimed attention from the GA community. This fact can be verified in several works such as Regal (Relational Learner Genetic Algorithm) [14, 15], G-NET (Genetic Network) [17], DOGMA (Domain Oriented Genetic Machine) [16] SIA01 (Supervised Inductive Algorithm version 01) [18], Evolutionary Concept Learner (ECL) [19] and QG/GA (Quick Generalization/Genetic Algorithm) [20], since all these works present some form of GA applied to ILP.

* The authors would like to thank Brazilian research agencies CNPq (Federal), FAPERJ (Rio de Janeiro), and FACEPE (Pernambuco) for the financial support.

Estimation of Distribution Algorithms (EDAs) [22], or Probabilistic Model Building Genetic Algorithms [21] are algorithms based on the explicit use of probability distributions. These algorithms completely (or partially) replace the traditional variation operators of the GAs, such as mutation and crossover, by building a probabilistic model of promising solutions and sampling the built model to generate new candidate solutions. The EDAs are conceived to capture the interactions between genes, which represent the internal structure of the solutions of the problem, and thereby directly estimate the distribution of good solutions instead of using genetic operators. One of the main differences between the various types of EDAs is related to the structure of the probabilistic model that is used to capture the interactions between the variables of the problem. These algorithms were applied to several optimization problems generating interesting results and surpassing, in most cases, the traditional method of GAs [23]. Despite demonstrating better results when compared to traditional GAs, the EDAs, to the best of our knowledge, were not yet used to develop an ILP system. Although [25, 31], henceforth called by the initials of their authors, i.e., OS, seems to be the only exception, this work uses only the motivation of the EDAs, i.e., it just models (using a probabilistic model) promising areas of the search space. Consequently, by only using this motivation, it cannot be said that OS is an application of EDAs to ILP; indeed, as can be seen in section III, the OS system has fundamental differences when compared to an EDA.

This work introduces EDA-ILP, an ILP system based on EDA. Next we present five guidelines adopted for developing the proposed system as well as brief discussion that justifies why these strategies were adopted. In conjunction, these guidelines differentiate the EDA-ILP from all others ILP systems.

I. Use of the bottom clause: EDA-ILP searches for clauses whose bodies are subsets of the literals in the bottom clauses.
Justification: The use of bottom clauses have proved to be a good rule of thumb and can be considered one of the main reasons of the success of important ILP systems, such as Progol [10, 11], Aleph [13], QG/GA, and OS [25, 31].
II. Use of an EDA as the search procedure: EDA-ILP uses an EDA as the only search algorithm.

Justification: First, there is the fact that there is no ILP system based on EDAs. Second, EDAs have already proved to be effective in complex problems.

III. Use of the Bayesian networks as probabilistic models: As in OS, EDA-ILP uses Bayesian networks whose structures captures the dependency between the literals in the bottom clauses.

Justification: OS demonstrated that the use of Bayesian networks for modeling the search space is a powerful strategy and can achieve interesting results.

IV. Use of the PBIL's [26] update rule to update the probabilistic models: EDA-ILP uses an "adaptation" of the PBIL's update rule.

Justification: PBIL's update rule is very simple (to perform and to implement) and thus can contribute to speeding up the execution of the system.

V. Evolve theories instead of single clauses: EDA-ILP searches for a whole theory (set of clauses) instead of a single clause, as occurs in QG/GA, Progol, Aleph, OS and others.

Justification: When the search is performed by isolated clauses, usually some kind of covering algorithm has to be employed to gradually build the theory. As pointed by [43], this kind of approach can generate theories that are unnecessarily large, i.e. theories with a large number of clauses. Thus, by evolving whole theories, we expect having smaller theories.

To evaluate EDA-ILP, first the system was compared to one of the state of the art of ILP systems, the Aleph. Later, we replaced the search done by the probabilistic model by the search used in the conventional GAs, i.e., by means of mutation and crossover operators. With the replacement of the probabilistic model, we have created another instance of the system, called GA-ILP, which was compared to EDA-ILP. To perform all experiments we have selected two datasets, namely carcinogenesis [34] and alzheimers-amine [35], mainly because they are well known to the ILP community and are good examples of practical problems where relational knowledge is important [36]. First results indicate a promising future to the use of EDA in ILP.

The work is organized as follows. Section II briefly presents the ILP problem; also, it discusses how one can use the bottom clauses to perform the exploration of the search space, and finally presents a quick overview of Aleph. Section III presents the general cycle of EDAs as well as the classes in which they are organized. Section IV presents an overview of OS, since this work is the closest one to the application of an EDA to ILP. Section V presents the details of EDA-ILP. Section VI presents the first experiments done to evaluate the proposed system. Finally, the conclusions and a list of future works are presented in Section VII.

II. Inductive Logic Programming

ILP can be viewed as the study of methods for learning theories that are expressed by means of a fragment of first-order logic. More formally, one can say that the basic problem of ILP is [8]:

Definition 1 (Inductive Logic Programming): Given: (a) an invariant preliminary knowledge denoted by BK, and (b) a set of positive and negative examples, denoted respectively by, E+ and E-, one must find a theory H that together with the BK logically implies (cover) all positive examples (H is complete), i.e., $BK \cup H \models E+$, and none of the negative examples (H is consistent), i.e., $\forall e^- \in E- : BK \cup H \not\models e^-$.

In practice, the requirements of completeness and consistency are relaxed; as a result, one must find a theory that logically implies the greatest number of positive examples while logically implies the smallest number of negative examples. As presented, it can be said that the goal of ILP is to induce a theory (a set of clauses) given two datasets (E+ and E-) and other logical relations expressed in BK.

A. The Search in Inductive Logic Programming

ILP can be seen as the problem of searching a theory space for a theory that matches the conditions mentioned in definition 1. However, a drawback of a first-order logic representation is that the theory space associated to this representation is usually much larger than the search space associated with a propositional representation [37]. This is because the number of first-order logic candidate solutions is much higher than the number of propositional logic candidate solutions. For this reason, the search in theory space is typically limited by a set of inductive biases.

One of the main inductive bias adopted to guide the search for a theory in an ILP problem is the use of the so called bottom clause; such bias is used in some of the most important ILP systems, such as Progol, Aleph, QG/GA, OS, and others. The formal definition and the algorithm used to construct the bottom clauses is beyond the scope of this work[1], but it is important to know that a bottom clause is constructed from a positive example in E+ by using a set of modes (usually present in BK). Modes indicate which arguments of the literal are input arguments and which are output arguments. As the bottom clause is being constructed, a literal can only be added if its input arguments are satisfied by the output arguments of literals already added to the bottom clause. This way, the modes create a dependency between the literals of a bottom clause, i.e., a literal may not be added to a bottom clause unless its input arguments appear as output arguments in some prior literal already in this clause [25]. Systems that adopt the bottom clause usually restrict its search for clauses in a lattice that is superiorly limited by the empty body clause (a clause that has no literals in its body) and inferiorly limited by the bottom clause. Consequently, these systems search for clauses whose bodies are a subset of the literals in the bottom clause. Perhaps, the most popular system that adopts the bottom clause to guide its search is the Aleph system. The next section briefly reviews this system.

B. The Aleph System: a brief review

Aleph [13] is a top-down ILP covering algorithm written completely in Prolog and based on Progol [10] [11]. The goal of Aleph is to generalize the specific examples from the

[1]Readers interested in such topic should consider consulting [10].

training set into a set of clauses (theory), H, that explains (covers) most of the examples found in E+ but few of the examples in E- [13]. Each loop through the covering algorithm adds an additional clause to H. The clauses cover a portion of the positive examples. Generally, Aleph removes the subset of positive examples explained by the new clause; then the algorithm iterates. Once Aleph explains all positive examples, the algorithm terminates and returns the set of clauses that have been found. Aleph follows a very simple procedure that can be described in four steps [13]:

1- Select a positive example: Select a positive example to be generalized. If none exist, stop; otherwise go to the next step.
2- Build the bottom clause: Construct the bottom clause that covers the example selected.
3- Search: Find a clause more general than the bottom clause. This is achieved by searching for some subset of the literals in the bottom clause that has the "best" score.
4- Remove redundant: The clause with the best score is added to the current theory, and all examples made redundant (that are covered by the clause) are removed. Return to Step 1.

What makes Aleph an important ILP system is its potential flexibility; since Aleph has various parameters to be tuned, the system can be adapted to run and, in fact, find high-quality solutions in a wide range of problems.

III. Estimation of Distribution Algorithms

The EDAs [22] differ from the traditional GAs by using a probabilistic model that is used to generate new solutions instead of using crossover and mutation operators for this purpose. Briefly, these probabilistic models are learned and refined during the iterations (generations) of the algorithm with the objective to generate, at each iteration, better (fitter) solutions. Thus, the search in an EDA can be viewed as an iterative procedure for constructing a probabilistic model that generates, at each generation, better solutions. Initially, the probabilistic model usually codifies a uniform distribution over all possible solutions; but at each of the iterations, this model is updated in order to generate better solutions. The general procedure of EDAs can be described as follows [23]:

```
EDA-Begin
1- t ← 0;
2- Generate initial population P(0);
3- While (not done)
     4- Select population of promising solutions
        S(t) from P(t);
     5- Build probabilistic model M(t) for S(t);
     6- Sample M(t) to generate new candidate
        solutions O(t);
     7- Incorporate O(t) into P(t);
     8- t ← t+1;
   End-While.
EDA-End.
```

The main procedure of EDAs is similar to that of traditional GAs. First, at step 2, the initial population is generated. Step 3 controls the execution of the EDA. As in the traditional GAs, although some EDAs do not have it, a selection criterion, as tournament, roulette wheel, or ranking, is performed at step 4. At step 5, the selected individuals will be used to build the probabilistic model. As shown in step 6, there is no need to use crossover and mutation operators, since this step is responsible for generating new individuals and thus to explore the search space by sampling the probabilistic model built. Step 7 incorporates individuals generated by the model to the current population; however, there are EDAs where the whole population is replaced by the new generated individuals. Note that, in general, at each iteration of the EDA, the probabilistic model must be constructed and/or updated. Depending on the complexity of the model, this step can require significant computational time.

One of the main differences among EDAs is related to the structure of the probabilistic model used to capture the interactions between the variables of the problem. Generally speaking, all EDAs can be categorized into one of three broad classes that consider different types of interactions between the variables of the problem [23]. These classes are:

The no Interactions Model: This model does not consider any dependencies among variables of the problem, in other words, the EDAs that fall into this category assume that the variables are all independent of each other. Examples of EDAs that fall into this category are: Population Based Incremental Learning (PBIL) [26] and the Compact Genetic Algorithm (cGA) [27].

The Pairwise Interactions Models: These models consider dependencies in the form of a chain, a tree, or a forest (a set of isolated trees) [23]. Examples of EDAs that fall into this category are: Mutual Information Maximization for Input Clustering (MIMIC) [28] and Bivariate Marginal Distribution Algorithm (BMDA) [29].

The Multivariate Interactions Models: These models consider multiple (more than two) interdependency between variables. Examples of EDAs that fall into this category are Extended Compact Genetic Algorithm (ECGA) [30] and Bayesian Optimization Algorithm (BOA) [23].

IV. Related Works

As stated, to the best of our knowledge, there is no ILP system based on EDAs. The work OS [25, 31] is the system that comes the closest to the application of an EDA to ILP, however, as we present in this section, this work uses only the motivation the EDAS and, thus, cannot be considered an ILP system based on EDA.

The OS can be considered a modification of the work [32] that proposed a method called Rapid Random Restart (RRR) and incorporated it into Aleph with the objective to reduce the search time to found a clause, and therefore, to build a theory. As seen, Aleph constructs a bottom clause BC from a positive example that has not been covered, then searches for a clause in the lattice that is inferiorly limited by BC and superiorly limited by the empty body clause. RRR is used to search for clauses in this lattice. Basically, the RRR selects an initial clause in the lattice using a uniform distribution and then performs a local search (typically the best-first search) by a fixed amount of time in order to find a clause. This search aims to find a clause that satisfies certain constraints imposed *a priori*. If this clause is not found within the stipulated time limit, the RRR abandons the actual search and choose, using a uniform distribution, another initial clause and restarts its

search. This whole process is repeated for a maximum number of tries, which is provided as an input parameter. The overall procedure of the RRR can be described as follows [32].

```
RRR-Begin:
1- Repeat N times
     2- Select an initial clause uniformly;
     3- Perform Local Search for S clauses;
   End-Repeat.
End-RRR.
```

OS modifies RRR by not using a uniform distribution to generate clauses[2]; instead, it uses a non-uniform probability distribution to generate clauses in order to bias the search towards more promising areas of the search space. To capture this non-uniform distribution, OS uses a pair of Bayesian networks whose structure captures the dependency between the literals in the bottom clause. As said in section II, the modes specify dependences between the literals of the bottom clause, and these dependences are used to construct the structure of the Bayesian networks. It is important to say that once a bottom clause is generated, two Bayesian networks with the same structure are built. In short, OS uses a pair of Bayesian networks with the objective to generate a clause that will be used as a seed to the local search procedure, i.e., once a clause is generated by sampling the Bayesian networks, the local search procedure is responsible to explore the search space. Additionally, it is important to realize that the structures of the Bayesian networks are not changed during the search for a clause in the lattice, but their conditional probabilities tables (CPTs) are updated at each of the iterations in order to generate better initial clauses (seeds). The CPTs of one of the Bayesian network are updated using all the clauses generated so far, while the CPTs of the other Bayesian network are updated using only the best clauses generated. Thus, using both Bayesian networks, the authors claim that they can balance exploration and exploitation. It is important to say that OS adopts the noisy-OR [33] assumption with the objective to reduce the memory used to store the CPTs. The idea to seed good clauses (using a couple of Bayesian networks) in the lattice and then perform a local search was implemented in a system called Gleaner [24], unlike the RRR, which was implemented directly into Aleph.

The main objective of the Gleaner is to quickly build an ensemble of theories that produce good recall-precision curves. Gleaner currently uses Aleph as its underlying engine for generating bottom clauses as well as their methods for finding good clauses in the lattice. Roughly speaking, one can say that the Gleaner system is a shell that wraps Aleph as Aleph searches for clauses with good recall-precision curves. Gleaner initially used the RRR as its search method, however, [25, 31] showed that the OS was capable to generate better results in comparison to the RRR. Thus, "nowadays", Gleaner reaches better recall-precision curves using the OS method.

From the presentation of OS, one can see that the only similarity that this system has with an EDA is the idea to model the promising areas of the search space. Actually, OS only uses the probabilistic model learned to seed a promising place in the search space, then applies a local search to explore it. In contrast to OS, in general, the EDAs use a different approach, since the probabilistic model learned is not only responsible for finding a place of promising solutions, but also to explore it, while a local search procedure can be used in conjunction with the probabilistic model to explore the promising place. Another fundamental difference is that EDAs use a population (a set of solutions) that, in general, competes with each other to explore the search space. This kind of exploration does not occur in OS, since a single clause is considered each time during the lattice search, unlike to what would happen if an EDA was used, since it would consider exploring not a single clause, but a set of clauses at once.

V. THE EDA-ILP

A. The Search space and Encoding

The first thing to note in relation to how EDA-ILP explores the search space is the fact that the system searches for a whole theory instead of a single clause. Thus, each individual in the population codifies a set of clauses (a theory) instead of a single clause[3]. In this way, we first present how EDA-ILP codifies its clauses and later we show how this codification can be used to codify theories.

EDA-ILP also adopts the bottom clauses as one of its search bias, i.e., EDA-ILP searches for clauses whose literals are subsets of the literals in a given bottom clause. To do so, EDA-ILP uses the Aleph system in order 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 EDA-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 was proposed in [42] and used in [20]. The next example illustrates this form of codification.

Example 1: Assume that the following bottom clause BC was generated h(A,B) :- p(A,C), q(B,C), r(C,D). Fig. 1 shows two possible clauses generated by the system using the binary encoding.

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)
S2_bin	1	1	1	0
C2	h(A,B) :-	p(A,C)	q(B,C)	

Figure 1: Two possible clauses generated by EDA-ILP

From fig. 1, one can see how the binary encoding is used. As expected, the system should transform a binary string into a clause. This 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

[2] OS authors also modify the criteria to apply the local search procedure. Readers interested in such topic should consult [25, 31].

[3] In the field of evolutionary classifier systems, there are two main approaches for evolving rules, namely, Pittsburgh and Michigan. In Pittsburgh, each individual in the population codifies a set of rules, in contrast to the Michigan approach, where each individual codifies only one rule. From this point of view, one can say that EDA-ILP uses Pittsburgh approach.

in h(A,B) :- r(C,D) and S2_bin (1110) transformed into a clause will result in h(A,B) :- p(A,C), q(B,C).

It is important to note that the probabilistic model (explained in next section) used by EDA-ILP is responsible for generating those binaries strings that subsequently will be mapped into clauses. In the current version of the system, the number of the clauses in a theory is provided by the user (as an input parameter), thus, given that the user has set the number of the clauses in a theory to k, the EDA-ILP will construct k bottom clauses and codifies each clause into a binary string. In addition, for each bottom clause constructed, EDA-ILP also builds a probabilistic model.

B. The probabilistic model

The probabilistic model used by EDA-ILP was inspired by the OS system, this way; EDA-ILP also uses Bayesian networks as the probabilistic models and adopts the noisy-OR assumption. As in OS, this network is built on the bottom clause using the modes. This way, a Bayesian network is responsible for storing the dependencies between the literals of a bottom clause. Note that differently from OS, our system builds only one network for each bottom clause created, in contrast to OS (see section IV), which builds two Bayesian networks for a bottom clause. As in OS, the structures of Bayesian networks in EDA-ILP do not change during the course of the search, however, the CPTs are updated in order to guide the Bayesian networks towards promising areas of the search space. Differently from OS, which updates the CPTs using several clauses, the actual version of EDA-ILP adopts a simple way to updates its CPTs. EDA-ILP updates its CPTs using an adaptation of the PBIL's update rule. Briefly, PBIL's update rule can be described as follows [23]:

$$pi \leftarrow pi + (xi - pi), \qquad (1)$$

whereas pi denotes the probability of a 1 in the *i*th position of solution strings, $\lambda \in (0, 1)$ is the learning rate, and xi is the *i*th bit of the best solution in the current generation.

It can be said that the PBIL's update rule makes the probability-vector entries move toward the best solution and, consequently, the probability of generating this solution increases. In our system, the intuition described above works exactly the same way, however, before updating a specific entry of a CPT relative to a node i, we must take into account the values (0 or 1) assumed by the parents of this node. To make clear how EDA-ILP updates its CPTs, consider the following example.

Example 2: Assume that by the mode declarations, the bottom clause h(A,B) :- p(A,C), q(B,C), r(C,D), from example 1, originated the Bayesian network expressed in Fig. 2. Table I represents the CPT[4] for the literal r(C,D) (we call h(A,B) as l_0 (literal 0), p(A,C) as l_1, q(B,C) as l_2, and r(C,D) as l_3). Since l_1 and l_2 have no parents, there is no CPT associated with this nodes, instead, there are only a marginal probability associated to each one, thus, assume that, $P(l_1 = T) = 0.5$ and $P(l_2 = T) = 0.5$. Note that the highlighted line in table I was computed using the noisy-OR, this way, this probability does not need to be stored in memory.

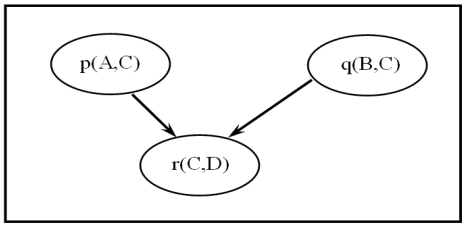

Figure 2: A Bayesian network for a bottom clause

TABLE I: CPT FOR THE LITERAL L_3

-	l_1	l_2	$P(l_3 = T \mid l_1, l_2)$
1	T	T	0.75
2	T	F	0.5
3	F	T	0.5
4	F	F	0.1

Supposing that the best individual in the current population is represented by the binary string {1, 1, 1, 0}, i.e., by the clause h(A,B) :- p(A,C), q(B,C), and that λ was set to 0.5, our use of PBIL's update rule will returns the following updates[5]:

- Updating nodes with no parents: In this case, we directly apply PBIL's update rule, thus:
  1- $P(l_1 = T) \leftarrow 0.5 + 0.5 * (1 - 0.5) = 0.75$
  2- $P(l_2 = T) \leftarrow 0.5 + 0.5 * (1 - 0.5) = 0.75$
- Updating nodes with parents: In this case we have to check for the parents of the node and also its values (0 or 1). After that, we have to update the lines from the CPT that holds the child and the parent's values (remember that we use noisy-OR, this way, not all probabilities values are stored in the table, since they can be calculated). Thus, to update $P(l_3 = T|l_1 = T, l_2 = T)$, we have to do the following modifications:
  3- $P(l_3 = T| l_1 = T, l_2 = F) \leftarrow 0.5 + 0.5 * (0 - 0.5) = 0.25$
  4- $P(l_3 = T| l_1 = F, l_2 = T) \leftarrow 0.5 + 0.5 * (0 - 0.5) = 0.25$

From 1 and 2 we can see that the $P(l_1 = T)$ and $P(l_2 = T)$ moved from 0.5 to 0.75. From 3 and 4, we can see that $P(l_3 = T|l_1 = T, l_2 = T)$, calculated using the noisy-OR, moved from 0.75 to 1 - (1 - 0.25) * (1 - 0.25) = 0.4375. This way, we can see that the probabilities entries moved toward generating the best solution, what is the intuition of the PBIL's update rule. As stated, we used this "adaptation" of PBIL's update rule in this first version of our system, since this rule is simple to be implemented and to be used, nevertheless, other updating rules will be tried in subsequent versions of EDA-ILP.

Before presenting the execution cycle of the EDA-ILP, it is important to note that the initial probabilities of the CPTs are input parameters of the system, thus, they might be provided by the user. These probabilities were called p_1, p_2, and p_3. p_1 denotes the probability of a T in a node that has no parent (we have one p_1 for each node that has no parent). p_2 denotes the probability of a T in a node that has parents given that one of its parents assumes the value T and all the others parents

[4] Note that the T and F values in a CPT can be interpreted, respectively, as 1 and 0. This fact is used while EDA-ILP samples the Bayesian network to generate the binary string that latter will be mapped into a clause.

[5] The bit that represents the first literal of the clause (or its head) will be always set to 1, since does not make sense to learn a clause without its head. In other words, it means that $P(l_0 = T) = 1$ and will be always 1, since this probability will never be updated.

assume the value F (we have one p_2 for each parent of a node). In the table I, lines 2 and 3 are examples of p_2. p_3 denotes the probability of a T in a node given that all its parents assume the value F (this probability is always low, because it breaks the restriction imposed by the modes)[6]. In table I, the line 4 is an example of p_3. All other probabilities can be calculated from p_2 using the noisy-OR. In table I, the line 1 is an example of the application of noisy-OR. As will be seen in section VI, p_1, p_2, and p_3 were initialized in a simple and very general way, nevertheless, as will be presented, the EDA-ILP obtained interesting results.

C. The main procedure of EDA-ILP

The main procedure of the EDA-ILP can be represented as:

```
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 to run the system.
EDA-ILP-Begin:
1 - Create k bottom clauses;
2 - Create k Bayesians networks (one for each
    bottom clause);
3- Repeat (Number_Generation) times
     4- Generate num_pop individuals (theories)
        by sampling all k Bayesians networks
        num_pop times;
     5- Evaluates all num_pop individuals with the
        fitness function;
     6- Select the fittest individual in the
        Population and updates the CPTs of all the
        k Bayesian networks;
   End-Repeat.
7- Generate num_pop individuals and returns the
   fittest one.
End-EDA-ILP.
```

The main procedure of EDA-ILP is very simple. First (steps 1 and 2) EDA-ILP creates k bottom clauses and k Bayesian networks (BNs). Step 4 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 5 evaluates the population with a predefined fitness function (accuracy, for example). The step 6 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 7 generates a new population and returns the fittest individual as the learned theory. Note that PBIL also inspired the execution cycle of the EDA-ILP, since our system, as occurs in PBIL, does not maintain a population between the generations and also, at each iteration, generates a completely new population by sampling the probabilistic model (in our case, sampling the BNs).

VI. EXPERIMENTS

A. Experimental Methodology

To evaluate EDA-ILP, we used two other systems, namely Aleph and GA-ILP. As said, Aleph can be considered a very powerful system due to its strong flexibility. To create GA-ILP we replaced the EDA (from EDA-ILP), by a "conventional" GA that uses the tournament selection, two points crossover and bit flip mutation. We used two datasets, namely, carcinogenesis (Carc) [34] and alzheimers-amine (Alz) [35], since they are good examples of practical ILP problems [36]. All the systems were evaluated in relation to the accuracy (using the corrected two-tailed paired t-test [38] 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[7].We used stratified 10-fold internal cross-validation to set the parameters [37] of the EDA-ILP.

Since the goal of the GA-ILP is to check the search capabilities of the EDA-ILP, the parameters gen_n (number of generations), pop_s (population size) and num_c (number of clauses in the theories) for GA-ILP were set the same as in EDA-ILP. The other parameters for GA-ILP, such as cros_p (crossover probability) mut_p (mutation probability) and tor_s (tournament size) were set using cross-validation as done for the EDA-ILP. The fitness function for EDA-ILP and GA-ILP was set to accuracy. The parameters for EDA-ILP and GA-ILP are presented in table II (note that p_1, p_2 and p_3 were initialized in a general way). All the experiments were performed in a Dual Core Pentium 2.0 GHz with 2.0 GB of RAM. All systems used Yap [39] as the Prolog engine. The configuration for Aleph was taken from [40] since this work suggests good values of parameters for Aleph[8].

TABLE II: CONFIGURATION OF EDA-ILP AND GA-ILP

-	**Alzheimers-amine**		**Carcinogenesis**	
	EDA-ILP	GA-ILP	EDA-ILP	GA-ILP
gen_num	500	500	100	100
pop_size	20	20	10	10
num_cls	3	3	1	1
λ	0.005	doesn't apply	0.01	doesn't apply
p1\p2\p3	0.5\0.5\0.1 *	doesn't apply	0.1\0.1\0.1 *	doesn't apply
mut_prob	doesn't apply	0.75	doesn't apply	0.6
cross_prob	doesn't apply	0.05	doesn't apply	0.05
tour_size	doesn't apply	2	doesn't apply	2

* p1, p2, and p3 for all nodes of the Bayesian networks were respectively initialized with these values.

[6] The OS system always respects the restrictions imposed by the modes, i.e., p_3 is always 0. Contrarily, we opted to give more flexibility to EDA-ILP using small values for p_3, however, more studies are needed to evaluate the best value for p3.

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

[8] Use the cross-validation 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 they achieved excellent results.

The experiments performed showed that the average number of literals in a bottom clause was, respectively, 49.7 and 67.3 for Alz and Carc. Considering the number of parents of a node in a Bayesian network, the experiments showed that a node have at most 10 parents for both datasets.

B. EDA-ILP x Aleph

The table III presents the results for EDA-ILP and Aleph. For this table as well as for all others, a highlighted cell (for accuracy) indicates that the result is statistically significant.

TABLE III: EDA-ILP X ALEPH

	EDA-ILP				**Aleph**			
	Acc.	#Cls.	#Lit.	T(s).	Acc.	#Cls.	#Lit.	T(s).
Alz.	73.5	3	7.2	45.7	69.7	7	3.4	56.6
Carc.	61.8	1	3.6	9.5	62.7	4.7	1.8	5.8

Considering the table III, one can see that both systems are competitive.

For Alz, the EDA-ILP obtained a statistically significant result in relation to the Aleph considering the accuracy (Acc) in the test set. Note that the EDA-ILP finds theories with a smaller number of clauses (#Cls), which confirms the problem noticed in [43] about using a large numbers of clauses when a covering algorithm is being used. The number of literals (#Lit) in the theories founded by EDA-ILP is higher when compared to the Aleph system. This result was expected and can be justified due to the fact that the EDA-ILP uses a more "global" search when compared to Aleph; remember that Aleph focuses the search in single clauses, while EDA-ILP develops entire theories. In relation to this topic, preliminary studies showed that some literals in the theories developed by EDA-ILP are redundant and can be removed without affecting the accuracy of the theory. This way, these initial studies point to the fact that the theories from EDA-ILP can be simplified by removing these literals. Considering the execution time (T(s)) for Alz, EDA-ILP takes about 19% less time.

For Carc dataset, the accuracies in the test set are equivalent. Note, however, that the EDA-ILP achieves an equivalent accuracy using only a single clause, while the Aleph uses, on average, 4.7 clauses. In this dataset, the Aleph used less time (39 % less) when compared to the EDA-ILP.

Comparing Alz and Carc results, one can see that EDA-ILP had more difficulty to deal with the Carc dataset since our system used more time and found an equivalent accuracy result in relation to Aleph. This fact motivated us to investigate the performance of the system during the cross-validation done to set its parameters. In short, we found that theories with a large number of literals and clauses demand considerable time to be evaluated but, generally, achieve very low accuracy. This way, we realized that theories that need high execution time to be evaluated should be ignored by EDA-ILP. With this objective, we implemented a method to cut off the evaluation of a theory after a certain time limit (given as an input parameter). This cutoff limit is given in seconds and it is relative to an example, thus, if EDA-ILP is trying to prove a positive example and the time limit exceeds, EDA-ILP assumes that this example has not been proved. This idea is also applied to the negative examples, however, if the time limit exceeds while proving a negative example, EDA-ILP assumes that this negative example was proved. By definition 1, we see that this policy penalizes theories (individuals) that demand considerable time to be evaluated.

In order to verify the cutoff limit, the EDA-ILP was executed again in Carc since EDA-ILP had more difficulty to deal with this dataset. The cutoff limit was set to 0.1s and the number of clauses in the theory was set to 3 (both parameters were set using cross-validation), while all other parameters were kept as described in table II. EDA-ILP using the cutoff limit was called tEDA-ILP. Table IV presents the results for the tEDA-ILP in Carc dataset.

TABLE IV: TEDA-ILP FOR CARCINOGENESIS

tEDA-ILP				
-	Acc.	#Cls.	#Lit.	T(s).
Carc.	68.5	3	2.3	14.0

The tEDA-ILP achieves statistically significant results in relation to EDA-ILP and Aleph (see table III). Note that the number of literals tEDA-ILP is lower when compared to the EDA-ILP. In relation to the Aleph, the number of literals in tEDA-ILP is higher, but tEDA-ILP uses fewer clauses in relation to Aleph system. Regarding execution time, the tEDA-ILP required 14s compared with 9.5s used by EDA-ILP; however, it is important to note that tEDA-ILP develops theories with 3 clauses while EDA-ILP develops theories with a single clause. This way, the difference in execution times is relatively low, what shows that the cutoff limit is an interesting mechanism. The success of the tEDA-ILP can be justified due the fact that the cutoff limit imposes a restriction to the theories explored, i.e., theories that demand considerable time to be evaluated are ignored, and since those clauses, especially for Carc, achieve low accuracy, the system does not waste time exploring them and thus concentrates efforts to explore other theories.

C. EDA-ILP x GA-ILP

In executing GA-ILP, we tried two forms to generate the initial population. The first way was the classical one, i.e., the initial population was generated with a uniform probability. The second way generated the initial population exact the same way in EDA-ILP, i.e., using the Bayesian networks constructed over the bottom clauses. As the second form to generate the initial population achieved superior results, this section presents the results obtained by GA-ILP using this kind of initialization. Table V presents the results of GA-ILP.

TABLE V: EDA-ILP X GA-ILP

	EDA-ILP				**GA-ILP**			
	Acc.	#Cls.	#Lit.	T(s).	Acc.	#Cls.	#Lit.	T(s).
Alz.	73.5	3	7.2	45.7	71.3	3	9.8	50.8
Carc.	61.8	1	3.6	9.5	57.75	1	6.8	28.8

In general, one can see that EDA-ILP is superior to GA-ILP. For Alz, despite that the accuracies are equivalent; GA-ILP used more time to obtain theories that use more literals when compared to EDA-ILP. For Carc, the results obtained by GA-ILP are even worse: the accuracy of EDA-ILP is statistically significant when compared to GA-ILP and this last

system used much more time to develop theories that use more literals.

As occurred with EDA-ILP when we created the cutoff limit, we also used this mechanism in GA-ILP creating, therefore, tGA-ILP. As one might expect, tGA-ILP was executed in Carc using the same cutoff limit for tEDA-ILP (0.1s) and the same number of clauses in a theory (3 clauses). Table VI presents the results of tGA-ILP for Carc.

TABLE VI: TEDA-ILP X TGA-ILP

	tEDA-ILP				**tGA-ILP**			
	Acc.	#Cls.	#Lit.	T(s).	Acc.	#Cls.	#Lit.	T(s).
Carc.	68.5	3	2.3	14.0	63.8	3	6.0	42.6

The accuracy obtained by tEDA-ILP is statistically significant when compared to the tGA-ILP. Note that the theories founded by tEDA-ILP use less literals when compared to the theories founded by tGA-ILP. Additionally, tEDA-ILP uses less time to obtain its results. As occurred for the other cases, one can say that the Bayesian networks explore the search space more efficiently when compared to the "traditional" search present in the standard GAs.

VII. CONCLUSIONS

This work presented EDA-ILP, an ILP system based on an EDA. EDA-ILP uses a set of Bayesian networks as probabilistic models to search the theory space. At each generation, the CPTs of these Bayesian networks are updated using PBIL's update rule in order to generate better solutions. EDA-ILP was executed in two classical datasets and proved to be very competitive in relation to Aleph, a popular and flexible ILP system, and also showed to overcome the "traditional" GA search. Future works involve: (1) execute EDA-ILP in other complex datasets [32] and compare the results with other ILP systems such as QG/GA, OS, and others; (2) use another form to update the CPTs; (3) allow the system to determine the number of clauses in a theory, (4) use a local search procedure, as hBOA [23] or as in [41] (specifically for rule learning) to improve EDA-ILP search.

VIII. REFERENCES

[1] Holland, J. (1975); *Adaptation in natural and artificial systems. MIT Press, Cambridge.*

[2] Davis, L. (1991); *Handbook of Genetic Algorithms. Van Nostrand Reinhold.*

[3] Goldberg, D. (1989); *Genetic Algorithms in search, optimization and machine learning. Addison-Wesley, 1989.*

[4] Lloyd, J. (1987); Foundations of Logic Programming, 2 ed., Springer Verlag.

[5] Blockeel, H., De Raedt, L. (1998); Top-down induction of first order logical decision trees. Artificial Intelligence, 101(1-2):285-297.

[6] Muggleton, S (1991); "Inductive Logic Programming", New Generation Computing, v. 13, n. 4, pp. 245-286.

[7] Muggleton, S., De Raedt, L. (1994); "Inductive Logic Programming: Theory and Methods", Journal of Logic Programming, v. 19, n. 20.

[8] Lavrac, N.; Dzeroski S. (1994); *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York.

[9] http://www.doc.ic.ac.uk/~shm/applications.html, last access 25/01/2011.

[10] Muggleton, S. (1995); Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Progra mming, 13(3-4):245–286.

[11] Muggleton, S. (1996); Learning from positive data. In Muggleton, S., editor, Proceedings of the 6th International Workshop on Inductive Logic Programming, pages 225–244. Stockholm University, Royal Institute of Technology.

[12] Quinlan, J. (1990); Learning logical definition from relations. Machine Learning, 5:239–266.

[13] Srinivasan. A, The Aleph Manual, last access: 25/01/2011 http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/

[14] Giordana, A. and Neri, F. (1996); "*Search-intensive concept induction"*. Evolutionary Computation, 3(4):375.416.

[15] Neri, F. and Saitta, L. (1995); "*Analysis of genetic algorithms evolution under pure election"*. In Proceedings of the Sixth International Conference on Genetic Algorithms, pages 32.39. Morgan Kaufmann, San Francisco, CA.

[16] Hekanaho, J. (1996); "*Background knowledge in GA-based concept learning"*. In International Conference on Machine Learning, pages 234.242

[17] Anglano, C., Giordana, A., Bello, G. L., and Saitta, L. (1998); "*An experimental evaluation of co-evolutive concept learning"*. In Proc. 15th International Conf. on Machine Learning, pages 19.27. Morgan Kaufmann, San Francisco, CA.

[18] Augier, S., Venturini, G., and Kodratoff, Y. (1995); "*Learning First order logic rules with a genetic algorithm"*. In Fayyad, U. M. and Uthurusamy, R., editors, The First International Conference on Knowledge Discovery and Data Mining, pages 21.26, Montreal, Canada. AAAI Press.

[19] Divina, F. and Marchiori, E. (2002); "*Evolutionary concept learning"*. In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pages 343.350, New York. Morgan Kaufmann Publishers.

[20] Muggleton, S.,; Tamaddoni-Nezhad A. (2006); QG/GA: A stochastic search approach for Progol. In Proceedings of the 16th International Conference on Inductive Logic Programming, LNAI 4455, pages 37-39. Springer-Verlag.

[21] Pelikan, M., Goldberg, D. E., & Lobo, F. G. (1999); "A *survey of optimization by building and using probabilistic models"*. Urbana, IL: University of Illinois Genetic Algorithms Laboratory (IlliGAL Report No. 99018).

[22] 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).

[23] Pelikan, M, (2005); *Hierarchical Bayesian Optimization Algorithm Toward a New Generation of Evolutionary Algorithms*, Series: Studies in Fuzziness and Soft Computing, Vol. 170, Springer; 1 edition.

[24] Goadrich M., Oliphant L., and Shavlik. J. (2006); Gleaner: Creating Ensembles of First-Order Clauses to Improve Recall-Precision Curves. Machine Learning, 64(1-3):231–261.

[25] 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.

[26] 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).

[27] Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1998); The compact genetic algorithm. In Proceedings of the IEEE Conference on Evolutionary Computation 1998 (ICEG'98) (pp. 523-528). Piscataway, NJ: IEEE Service Centre.

[28] De Bonet, J. S., Isbell, C. L., & Viola, P. (1997); MIMC: Finding optima by estimating probability densities. In Mozer, M. C., Jordan, M. I., & Petsche, T. (editors), Advances in Neural Information Processing Systems, Volume 9 (pp. 424). The MIT Press, Cambridge.

[29] Pelikan, M., & Mühlenbein, H. (1999); The bivariate marginal distribution algorithm. In Roy, R., Furuhashi, T., & Chawdhry, P. K. (Eds.), Advances in Soft Computing - Engineering Design and Manufacturing (pp. 521--535). London: Springer-Verlag.

[30] Harik, G. (1999); Linkage learning via probabilistic modeling in the ECGA. Urbana, IL: University of Illinois Genetic Algorithms Laboratory (IlliGAL Report No. 99010).

[31] Olhiphant, L. (2009); Adaptively Finding and Combining First-Order Rules for Large, Skewed Data Sets. PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison.

[32] Zelezn F., Srinivasan A., and Page D. (2003); Lattice-Search Runtime Distributions may be Heavy-Tailed. In Proceedings of the 12th International Conference on Inductive Logic Programming 2002, volume 2583 of Lecture Notes in Artificial Intelligence, pages 333–345, Syndey, Australia.

[33] Pearl. J. (1988); Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[34]. 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.

[35] 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.

[36] Muggleton S., Santos J., Tamaddoni-Nezhad A., (2008) TopLog: ILP Using a Logic Program Declarative Bias. ICLP 2008: 687-692.

[37] Mitchell, T. (1997). Machine learning. New York: McGraw-Hill.

[38] Nadeau C., Bengio Y., (2003) "Inference for the Generalization Error", Machine Learning 52(3) pp. 239-281.

[39] Yap Prolog: http://www.dcc.fc.up.pt/~vsc/Yap/, last access: 25/01/2011.

[40] 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.

[41] Pitangui, C., Zaverucha, G., (2008), Genetic local search for rule learning**,** Genetic And Evolutionary Computation Conference (GECCO) Atlanta, GA, USA, pp. 1427-1428, 2008.

[42] Alphonse, E., Rouveirol, C., (2000), Lazy propositionalisation for Relational Learning, 14th European Conference on Artificial Intelligence 2000 (ECAI'00), pages 256-260, IOS Press

[43] 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.