Appears in *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*. Corvallis, Oregon, USA. June, 2007.

Using Bayesian Networks to Direct Stochastic Search in Inductive Logic Programming

Louis Oliphant and Jude Shavlik

Computer Sciences Department, University of Wisconsin-Madison

Abstract. Stochastically searching the space of candidate clauses is an appealing way to scale up ILP to large datasets. We address an approach that uses a Bayesian network model to adaptively guide search in this space. We examine guiding search towards areas that previously performed well and towards areas that ILP has not yet thoroughly explored. We show improvement in area under the curve for recall-precision curves using these modifications.

1 Introduction

Inductive Logic Programming (ILP) [3] algorithms search for explanations written in first-order logic that discriminate between positive and negative examples. Two open challenges for scaling ILP to larger domains include slow evaluation times for candidate clauses and large search spaces in which to find those clauses. Our work addresses this second challenge using adaptive stochastic search.

Algorithms such as Progol [6] and Aleph [12] also address the second challenge by constraining the size of the search space using a bottom clause constructed from a positive seed example. A bottom clause is constructed from a positive seed by using a set of user-provided modes. The user creates a mode for each literal in the background knowledge. Modes indicate which arguments of the literal are input arguments and which are output arguments. As the bottom clause is being constructed, only literals whose input arguments are satisfied by the output arguments of literals already in the bottom clause may be added. The modes create a dependency between the literals of a bottom clause. A literal may not be added to a candidate clause unless its input arguments appear as output arguments in some prior literal already in the candidate clause.

Even with these constraints the size of the search space usually is much larger than can be exhaustively searched. Zelezny *et al.* [13] have incorporated a randomized search algorithm into Aleph in order to reduce the average search time. Their rapid random restart (RRR) algorithm selects an initial clause using a pseudo-uniform distribution and performs local search for a fixed amount of time. This process is repeated for a fixed number of tries.

Our work builds on top of the RRR algorithm and bottom-clause generation. We construct a non-uniform probability distribution over the search space that biases search towards more promising areas of the space and away from areas which have already been explored or that look unpromising. We use a pair of

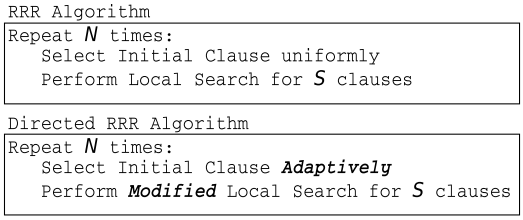

Fig. 1. Pseudo-code showing the Rapid Random Restart (RRR) algorithm and our modified version of RRR.

Bayesian networks to capture this skewed distribution. The structure of the networks is determined by the bottom clause and the parameters are trained as ILP's search progresses. The clauses that are evaluated by the ILP system become the positive and negative examples for training the Bayesian networks. The trained networks are then used to select the next initial clause and to modify the local search portion of RRR.

2 Directed Stochastic Search Algorithm

Zelezny's RRR algorithm appears in Figure 1 on the top. We have incorporated a non-uniform distribution into this algorithm in order to bias search towards more promising areas of the search space. Our modifications appear in Figure 1 on the bottom. In the following subsections we explain our method of modeling a probability distribution over ILP's search space, training the parameters of the model, and using this trained model to modify RRR's search process.

2.1 Modeling ILP's Search Space with Bayesian Networks

A Bayesian network [5] is a directed acyclic graphical model that captures a full joint probability distribution over a set of variables. Each node in the graphical model represents a random variable. Arcs represent dependencies between variables. Each node has a probability distribution showing the probability of the variable given its parents.

ILP's search space consists of subsets of literals from the bottom clause. A sample bottom clause appears in Figure 2 on the left. Literals' arguments have been annotated with +/- marks to indicate input/output arguments taken from the user-provided modes. Not all subsets of literals are legal clauses. A clause is legal and in the search space if the input arguments of each literal in a clause first appear as output arguments for some literal earlier in the clause.

We capture these dependencies created by the user-provided modes in a graphical model. Figure 2 on the right shows graphically the dependencies found in the bottom clause. Each node in the network represents a Boolean variable that is true if the corresponding literal from the bottom clause is included in

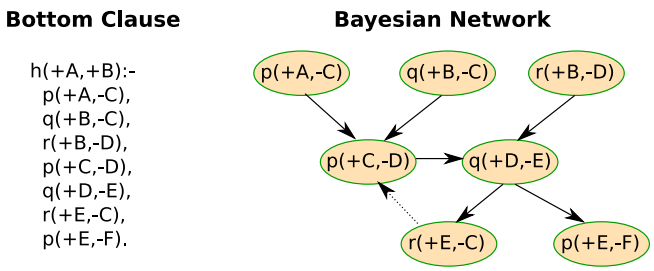

Fig. 2. A portion of a Bayesian network built from the literals in a bottom clause. The + marks indicate input arguments and the - marks indicate output arguments taken from the user-provided modes. The head literal is not part of the Bayes net. Arcs indicate dependencies between the output variables of one literal to the input variables of another literal. Dotted arcs are dropped to maintain the acyclic nature needed for Bayesian networks.

a candidate clause. Each arc represents a connection between the output arguments of one literal to the input arguments of another literal. Dotted arcs indicate dependencies that are dropped in order to maintain the acyclic nature needed for Bayesian networks. The structure of the Bayesian network is determined by the bottom clause while the parameters are learned as ILP's search progresses.

The algorithm to create the Bayesian network structure from a bottom clause appears in Figure 3. The algorithm constructs the Bayes net in a top-down approach. Variable `group` contains all literals whose input variables are satisfied by the `Head` literal or any literal already in the network. The literals in `group` are added one at a time to the Bayes net. As the literal is added into the Bayes net the algorithm connects it to all literals that contain some input variable that is not contained by the `Head` literal. Creating a group of literals and adding them to the network repeats until all literals have been added.

2.2 Training the Model

After creating the Bayesian network structure, we still need to learn the parameters of the model. Each node contains a conditional probability table (CPT) that predicts the probability the node is true given its parents. We estimate these probabilities from training data collected during ILP's search.

We construct two networks that have the same graphical structure. The parameters of the first network are trained using "good" clauses seen during search, while the parameters of the second are trained on all clauses evaluated. These two networks provide probabilities that indicate, respectively, how good a candidate clause is and how similar the clause is to past clauses. These distributions are density estimators indicating the promising areas of the search space and which areas have already been explored.

```
function CONSTRUCT_NETWORK(⊥): returns a Bayesian network
input: ⊥, a bottom clause consisting of a Head and Body
bayes_net=empty
reached=input variables from Head
while Body is not empty do:
    group={l|l ∈ Body and l's input variables are in reached}
    for each lit ∈ group do:
        ADD_NODE(lit,bayes_net) /* connects to lit all nodes
                                   in bayes_net that satisfy an input
                                   variable of lit*/
    Body = Body - group
    reached = reached + output variables from group
return bayes_net
```

Fig. 3. Pseudo-code showing the construction of a Bayesian network from a bottom clause.

The parameters of a node are estimated using ratios of weighted counts between clauses that contain the literal and those that do not for the various settings of the parents. We update the parameters during search when a clause is evaluated on the training data.

The first network, which estimates the probability that a clause is "good," is trained using high-scoring clauses. We have tried several methods for deciding which clauses to use. Our current approach involves using the Gleaner algorithm [4]. Gleaner retains a set of high-scoring clauses across a range of recall values. All clauses that are retained in Gleaner's database are used, along with those clauses in the trajectory from the initial clause to the one retained in the database. We use weighted counts (a clause's F1 score is its weight) with higher-scoring clauses receiving higher weights. This allows better clauses to have more influence on the network.

The second network, which estimates the probability that a new clause is similar to past clauses, is trained on all clauses considered, using a uniform weight on the clauses. The combination of the probabilities from these two networks will allow us to trade off exploration of unvisited portions of the hypothesis space for exploitation of the promising portions.

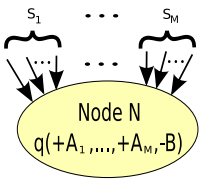

Fig. 4. Node with many arguments.

We have found that some nodes in our networks have 20 or more parents. In order to reduce the size of the conditional probability tables, we utilize a noisy-OR assumption [7]. The noisy-OR model assumes that all parents of a node are independent of each other in their ability to influence the node. This reduces the size of the CPT to be linear in the number of parents.

Figure 4 shows a single node whose corresponding literal has several input arguments. Each argument may be satisfied by one of many parents. We calculate the probability that a node, N, is true using the formula

$$P(N = t|\Pi(N)) = \prod_{j=1}^{M} P(N = t|S_j(N)) = \prod_{j=1}^{M} \left(1 - \prod_{R \in S_j(N)} P(N = f|R)\right)$$

where $\Pi(N)$ are the parents of N and S_j is the subset of $\Pi(N)$ that satisfy input argument j. The outer product ranges over all M input variables of the node. This reduces the conditional probability to a product of simpler conditional probabilities, one for each input argument. The simpler conditional probabilities are modeled as noisy-ORs. $P(N = f|R = f)$ is set to equal 1 so if any input argument is not satisfied by at least one parent then that portion of the product, $P(N|S_j(N))$, will be zero, making the entire product zero. This limits the clauses that have a non-zero probability to those that are legal.

2.3 Using the Model to Guide Search

The probabilities provided by the Bayesian networks are incorporated into a weight that we can attach to clauses. Recall that two networks are created. We call the probability from the network trained on "good" clauses $EXPLOIT$ and the probability from a second network trained on all clauses $EXPLORED$. We combine these two estimates into a weight for a candidate clause using the formula

$$W = \alpha * EXPLOIT + (1 - \alpha) * (1 - EXPLORED)$$

where $0 \leq \alpha \leq 1$. We can then set parameter α to trade-off exploration for exploitation. In order to interleave exploration and exploitation we select α from a range of values each time an initial clause is selected.

We use the clause weight to modify the RRR algorithm in two ways. The original RRR algorithm selects an initial clause uniformly. Our modified version of RRR performs K hill-climbing runs using the weights generated by the Bayesian network to guide search. We then select a single initial clause from the K local peaks by selecting a clause found at one of these peaks. We sample proportional to the weights of the clauses, with the idea that the search will begin in a higher-scoring and more diverse area of the space. Assigning a weight to a clause is relatively fast compared to evaluating the clause on the training data.

Next the original RRR algorithm performs a local search around this initial clause, expanding the highest-scoring clause on the open list and evaluating *all* of its neighbors on the training set. Our modified version of RRR attaches a weight to the neighboring clauses before they are evaluated on the training set and only a *high-weighted subset* of size L are retained and evaluated. This reduces the number of clauses that are evaluated on the training data which are close to any one initial clause, thus broadening the search and guiding it to areas of the search space which are more likely to contain high-scoring, unique clauses.

Our algorithm interleaves optimizing using our model and optimizing using real data. Assigning a weight to a clause using our model is much faster than evaluating a clause on real data when the dataset is large. We hypothesize our approach will outperform standard RRR search in terms of area under the recall-precision curve, when we allow RRR and our modified version to each evaluate the same number of clauses on real training data.

3 Directed-Search Experiments

We compare our search modifications using the Gleaner algorithm [4] of Goadrich *et al.* Following their methodology we compare area under the recall-precision curves (AURPC) using Gleaner with the standard RRR search algorithm and with our modified RRR search algorithm.

We evaluated our modifications to the RRR search algorithm on three datasets. The protein-localization biomedical information extraction dataset [4] contains 7,245 sentences from 871 abstracts taken from the Medline database. There are over 1,200 positive phrase-phrase relations and a positive:negative ratio of over 1:750. The relations were marked by hand using inter-expert agreement to reduce the number of incorrectly marked relations.

The gene-disorder dataset also comes from the biomedical information extraction domain. The dataset originally comes from work by Ray and Craven [9]. Due to memory limitations we scale the dataset by downsampling the abstracts. The relations in the dataset are marked by a computer algorithm. Our scaled down version contains 239 positive and 103,959 negative examples.

Finally, the University of Washington dataset contains 113 positive examples and 2,711 negative examples [10]. The task is to learn the advisor-student relationship. Background information includes facts about papers published and classes taught.

We assigned the α parameter to be between 0.01 and 0.75 in order to encourage exploration. The K parameter controlling the number of hill-climbing runs for each initial clause was set to ten, and the L parameter controlling how many neighbors of a clause are retained was set to twenty. The internal parameters of the Bayesian networks were updated as clauses were evaluated. We ran our experiment using 100 seeds for the two information extraction task and 50 seeds for the advisor-student task. We evaluated performance after one thousand, ten thousand, and twenty-five thousand clauses per seed.

Figure 5 shows the AURPC versus the number of clauses evaluated averaged over all five folds of each dataset. Although the improvement is small in the protein-localization task, it is significant for the first two points at the 95% confidence level using a paired t test. We also show improvement in the advisor-student task, however this improvement is not signification. Perhaps this is because the dataset is smaller, which may cause larger variance between folds. On the gene-disorder task no improvement is found. One possible reason for this may stem from the fact that the dataset has a considerable amount of noise due to the automatic labeling of the data as documented by Ray and Craven [9].

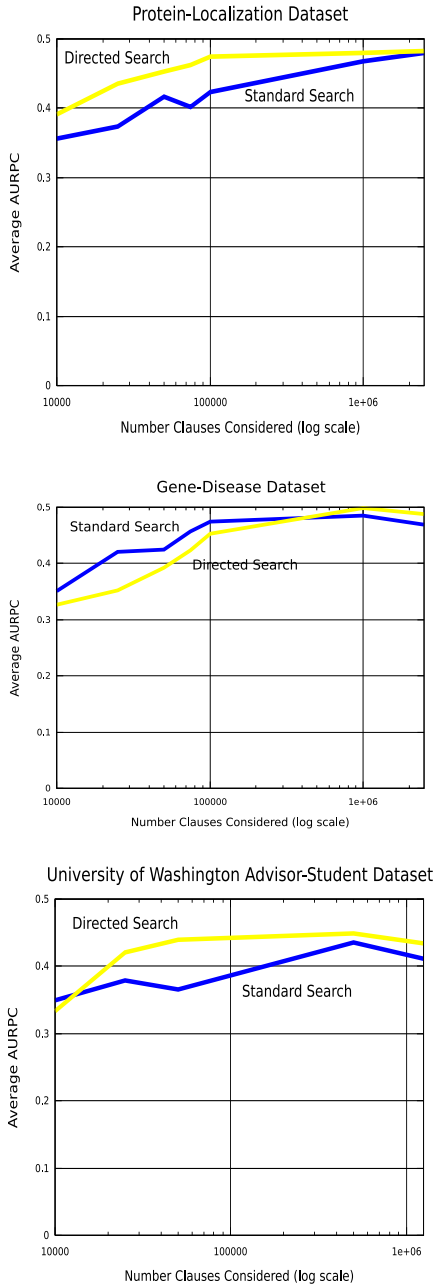

Fig. 5. Comparison on three datasets of AURPC for varying number of clauses considered using Gleaner with and without a directed RRR search algorithm.

One additional area that may improve performance would be to use a tuning set for setting the algorithms parameters.

4 Related Work

Several other researchers have used models to guide the search process. Pelikan *et al.* [8] developed the Bayesian Optimization Algorithm (BOA) which learns the structure and parameters of a Bayesian network from sampling the search space and uses the learned model to select a higher-scoring sample. Rubinstein's cross-entropy (CE) algorithm [11] comes from the rare-event modeling domain. It begins by uniformly sampling the search space and then the sample is sorted and a model is built using the highest ρ-percentile of the sample.

The STAGE algorithm by Boyan and Moore [1] learns an evaluation function that predicts the outcome of some type of local search, such as hill-climbing or simulated annealing. Their algorithm iterates between optimizing on real data and optimizing on the learned model. One final example is taken from the ILP literature. DiMaio and Shavlik [2] built a neural network to predict a clause's score in ILP. They use the clause's predicated score to modify selection of the initial clause, as well as to order clauses on the open list.

5 Conclusions and Future Work

Stochastic search of the space of clauses provides a means for ILP to scale to larger datasets. Our basic approach is to convert the dependency structure found in Aleph's modes into two Bayesian networks, whose parameters are trained as search progresses. These networks are used to influence where in the space of clauses will be searched next. We have shown improvement on area under the recall-precision curve experiments on two of three datasets by using this adaptive stochastic approach.

We plan on improving this approach by refining the structure of the networks as search progresses. An alternative approach, designing undirected graphical models that do not need to drop dependencies between literals, will also be considered. Fitting the parameters of either type of model quickly is important to reduce the amount of search. Allowing transfer of parameters from a model trained using one seed to models trained on some other seed will reduce the amount of training time needed for the new model. We plan on designing a mechanism for transferring these parameters. Finally we plan on including mechanisms to search for diverse clauses more directly, only allowing a clause to be retained when it is different enough from previously retained clauses. This should improve the diversity of the set of clauses, viewing the set more as an ensemble than as a single theory.

6 Acknowledgements

We would like to thank Mark Goadrich, Jesse Davis, Trevor Walker, and the anonymous reviewers for comments and suggestions on this work. This project is funded by DARPA IPTO under contract FA8650-06-C-7606 and DARPA grant HR0011-04-1-0007.

References

[1] J. Boyan and A. Moore. Learning Evaluation Functions for Global Optimization and Boolean Satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 3–10, 1998.

[2] F. DiMaio and J. Shavlik. Learning an Approximation to Inductive Logic Programming Clause Evaluation. In *Proceedings of the 14th International Conference on Inductive Logic Programming*, pages 80–97, Porto, Portugal, 2004.

[3] S. Džeroski and N. Lavrac. An Introduction to Inductive Logic Programming. In S. Džeroski and N. Lavrac, editors, *Proceedings of Relational Data Mining*, pages 48–66. Springer-Verlag, 2001.

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

[5] D. Heckerman. A Tutorial on Learning with Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, Washington, 1995. Revised June 1996.

[6] S. Muggleton. Inverse Entailment and Progol. *New Generation Computing Journal*, 13:245–286, 1995.

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

[8] M. Pelikan, D. Goldberg, and E. Cantú-Paz. BOA: The Bayesian Optimization Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume I, pages 525–532, Orlando, FL, 1999. Morgan Kaufmann Publishers, San Francisco, CA.

[9] S. Ray and M. Craven. Representing Sentence Structure in Hidden Markov Models for Information Extraction. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 2001.

[10] M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107–136, 2006.

[11] R. Rubinstein and D. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer-Verlag, Secaucus, NJ, USA, 2004.

[12] A. Srinivasan. The Aleph Manual Version 4. *http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/*, 2003.

[13] F. Železný, A. Srinivasan, and D. Page. 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, 2003.