A Comparative Study on Methods for Reducing Myopia of Hill-Climbing Search in Multirelational Learning

Lourdes Peña Castillo PENA@IWS.CS.UNI-MAGDEBURG.DE
Otto-von-Guericke-University Magdeburg, Germany

Stefan Wrobel WROBEL@AIS.FRAUNHOFER.DE
Fraunhofer AIS and University Bonn, Germany

Abstract

Hill-climbing search is the most commonly used search algorithm in ILP systems because it permits the generation of theories in short running times. However, a well known drawback of this greedy search strategy is its *myopia*. Macro-operators (or macros for short), a recently proposed technique to reduce the search space explored by exhaustive search, can also be argued to reduce the myopia of hill-climbing search by automatically performing a variable-depth look-ahead in the search space. Surprisingly, macros have not been employed in a greedy learner. In this paper, we integrate macros into a hill-climbing learner. In a detailed comparative study in several domains, we show that indeed a hill-climbing learner using macros performs significantly better than current state-of-the-art systems involving other techniques for reducing myopia, such as fixed-depth look-ahead, template-based look-ahead, beam-search, or determinate literals. In addition, macros, in contrast to some of the other approaches, can be computed fully automatically and do not require user involvement nor special domain properties such as determinacy.

1. Introduction

Multirelational learning or inductive logic programming (ILP) (Džeroski & Lavrač, 2001) is a subfield of machine learning concerned with inducing concept definitions using a first-order representation. Typically, ILP systems take as input background knowledge B, positive examples E^+ and negative examples E^-, and have to find a clausal theory (set of rules) T which is used to classify future examples. The systems' task is to find a theory T which minimizes the classification error on future instances.

In multirelational learning, performing exhaustive search can be too inefficient because the hypothesis space of most multirelational learning problems can be very large. Consequently, search strategies which only consider a limited number of alternatives at each level of the search are typically applied. Among these search strategies, hill-climbing search (often also called greedy search), which takes only one (the best) alternative at each level, is the most commonly used search algorithm in ILP (Fürnkranz, 1999). However, a well known problem of hill-climbing search is its *myopia*; i.e., the search algorithm might be unable to correctly assess the quality of a refinement and end up with a non-optimal clause.

Hill-climbing's shortsightedness occurs because hill-climbing search does not consider the existence of *non-discriminating* relations and the inability of the evaluation function to deal properly with literals denoting this kind of relations. For example, consider the eastward trains domain where the learning task is to find a theory to classify the trains according to their traveling direction (east or west). One of the relations in this domain is the structural relation between one train and its cars denoted by the predicate **hasCar**/2. Since **hasCar** is a non-discriminating relation (i.e., every train has one or more cars), literals having **hasCar** as predicate are not useful to distinguish between examples of trains belonging to different classes. However, they introduce a new reference to a car (i.e., a new variable) in a clause. The problem with these literals is that the evaluation value of clauses containing **hasCar** as their last literal is low, and thus they

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004. Copyright 2004 by the authors.

are not selected by hill-climbing. In this case, hill-climbing search may end up with a non-optimal clause or hit a dead end. The fact that hill-climbing cannot "see" that **hasCar** combined with other literals may yield a solution is called *myopia*.

In this paper, we integrate macro-operators, which have recently been shown to reduce the search space explored by exhaustive search (Peña Castillo & Wrobel, 2002), in a hill-climbing learner and empirically show that a greedy system using macros performs significantly better than current systems involving other techniques for reducing myopia.

Macros are based on provider-consumer iterations of existential variables between the literals; a macro-based refinement operator refines a clause by adding macros instead of single literals. The benefit derived from macros in hill-climbing is that the evaluation function is only applied to *admissible* clauses, which has the advantage that the quality of these clauses can be more accurately assessed since they have discriminative power and can be a solution. For example, in the case of the eastward trains domain, literals with the predicate **hasCar** would never be added to a clause without a consumer of **hasCar**'s output variables. Since a consumer of **hasCar** is always added, the evaluation function is able to appropriately estimate the relevance of the refinement. Using macros, the number of literals added at once to a clause are automatically adjusted so that only admissible clauses are generated.

In addition, we present the first detailed comparative study on approaches to reduce the myopia problem of hill-climbing search in multirelational learning. This study involves fixed-depth look-ahead, template-based look-ahead, beam-search, determinate literals and macros. Our results show that, with the exception of beam-search, a hill-climbing learner using macros reports significantly lower classification error than systems using other approaches.

The next section describes a hill-climbing search algorithm and illustrates its shortsightedness. In Section 3, we briefly explain look-ahead, beam-search and determinate literals. Section 4 explains macros and how they reduce hill-climbing's myopia problem. Section 5 describes our experiments, work related to macros is surveyed in Section 6, and Section 7 concludes.

2. Myopia of Hill-Climbing Search

With the purpose of having a search algorithm able to perform various search strategies, we formulate Algorithm 1. This algorithm performs a top-down search

```
Top_Down_Search
Input: The top of the lattice $\top$, $B$, $E$, $b$, $s$
Output: Either a clause $\mathcal{C}$ or $\emptyset$

1. $\mathbf{S} = \{\top\}$ /* Ordered set of clauses to consider*/
2. While a clause $\mathcal{D} \in \mathbf{S}$ can be refined and search resources are not exhausted
    (a) $\mathbf{R} = \emptyset$ /* Set of refinements */
    (b) For every clause $\mathcal{D} \in \mathbf{S}$ that can be refined
        i. $\mathbf{W} = \bigcup_{d=1}^{s} \rho^d(\mathcal{D})$ /*Refining $\mathcal{D}$ */
        ii. $\mathbf{R} = \mathbf{R} \cup \mathbf{W}$
    (c) Sort R according to $eval$
    (d) Let $\mathbf{R}_b$ be the best $b$ refinements in R
    (e) $\mathbf{S} = \mathbf{R}_b$
3. Let $\mathcal{C}$ be the best evaluated clause in S
4. If $\mathcal{C}$ covers enough positive and few enough negative examples in $E$
    (a) Then Return $\mathcal{C}$
    (b) Else Return $\emptyset$
```

Algorithm 1: A top-down search algorithm

and uses an example to guide the search for hypothesis. It receives two user-defined parameters, s and b, which indicate the amount of look-ahead and the beam width, respectively. The default values, $s = 1$ and $b = 1$, correspond to hill-climbing search. In Algorithm 1, $\rho^d(\mathcal{C})$ is the set of clauses obtained after d-step refinements of some clause $\mathcal{C}$.

To evaluate the quality of a clause $\mathcal{C}_i$, Algorithm 1 uses the evaluation function shown in Equation 1 which is based on the information gain heuristic (Fürnkranz & Flach, 2003). In Equation 1, pos_i and neg_i are the number of positive and negative examples covered by $\mathcal{C}_i$; $|E^+|$ and $|E^-|$ are the total number of positive and negative examples, respectively; $|\mathcal{C}_i|$ the number of body literals in $\mathcal{C}_i$ (or 1 if $\mathcal{C}_i$'s body is empty); $\top$ refers to the unit clause at the top of the refinement lattice, and $IC(\mathcal{C})$ returns the information content of a clause, which is given by $IC(\mathcal{C}_i) = -log_2(\frac{pos_i}{pos_i+neg_i})$.

$$eval(\mathcal{C}_i) = \frac{\frac{pos_i+(|E^-|-neg_i)}{|E^+|+|E^-|} * (IC(\top) - IC(\mathcal{C}_i))}{|\mathcal{C}_i|}. \quad (1)$$

To illustrate the myopia problem of hill-climbing using Algorithm 1, consider the student loan domain (Pazzani & Brunk, 1991) where the learning task is to find a theory to classify individuals into those who are not required to pay back an educational loan and those who must pay. Assume that we set the maximum length of the clauses to four body literals and that the bottom

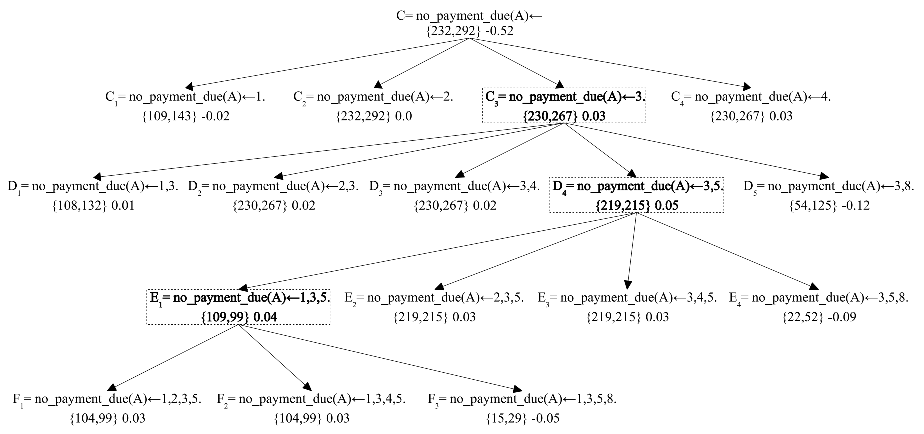


Figure 1. Search space explored by hill-climbing. Below each clause $\mathcal{C}_i$, pos_i and neg_i are shown between braces followed by $\mathcal{C}_i$'s heuristic value obtained by Equation 1. A square encloses the clause chosen by hill-climbing at each iteration.

```
no_payment_due(A) <--  -1- male(A),
   -2- longest_absence_from_school(A,I0),
   -3- enrolled(A,U1,I1), -4- enrolled(A,U2,I2),
   -5- gte(I1,3), -6- gte(I0,4), -7- gte(I2,9),
   -8- lte(I1,3), -9- lte(I0,4), -10- lte(I2,9).
```

Figure 2. Bottom clause $\perp$ for the student loan example

clause (see Figure 2) is derived from a given example e. The bottom clause $\perp$ is used to guide the search for a solution and as lower bound of the search space.

In the bottom clause shown in Figure 2, the numbers which precede the literals indicate the position of each literal in $\perp$ and we use them to refer to the literals.

Let us explain how Algorithm 1 works. In the first iteration, after step 2b, the set **R** contains four refinements of the clause $\mathcal{C}$ =**no_payment_due(A)**← (these refinements are $\mathcal{C}_1, \ldots, \mathcal{C}_4$ in Figure 1). There are only four refinements at the first level because the hypothesis language does not allow clauses with unbound input variables. From these refinements, $\mathcal{C}_3$ and $\mathcal{C}_4$ are the best evaluated clauses by Equation 1. Assume that, in step 2d, $\mathcal{C}_3$ is chosen. Then, after three more iterations, hill-climbing hits a dead end with three clauses in **S** which cannot be refined ($\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ in Figure 1), and since $\mathcal{F}_1$ does not satisfy the criterion in step 4, the algorithm returns $\emptyset$. Hill-climbing fails to find a solution because the literals 2, 3 and 4 do not have discriminative power and the evaluation function is unable to correctly assess their quality when added by themselves to a clause. The refinements which lead to a solution in this example are $\mathcal{C}_2$ and $\mathcal{C}_4$ and the solution is the clause **no_payment_due(A)** ← 2,4,7,9.

3. Reducing Hill-Climbing's Myopia

Before reviewing the various approaches used in ILP to alleviate the myopia of greedy systems, we introduce some terminology to describe these techniques.

As proposed by Peña Castillo and Wrobel (2002), literals can be classified in providers and consumers. A literal p is a *consumer* of literal q if p has at least one input variable bound to an output argument value of q; conversely, q is a *provider* of p. Notice that these relations apply as well to the head literal. A literal providing the value for an output argument of the head literal is a *head provider*, and a body literal is a *head consumer* if it consumes an input argument value of the head. A literal is a *dependent provider* if it introduces existential variables to a clause and if for every combination of ground bindings of its input variables there is at least one ground binding for each of its output variables. In this sense, dependent providers represent non-discriminating relations (e.g., **hasCar**).

3.1. Beam-search

Some ILP systems, such as m-FOIL (Džeroski & Bratko, 1992) and ICL (De Raedt & Van Laer, 1995), mitigate the myopia problem of hill-climbing search by performing beam-search. Beam-search considers the b best refinements at each level, where b ($1 < b \ll \infty$) is the beam width. With a value of b greater than one in Algorithm 1, beam-search is obtained. Because beam-search considers the best b refinements, the probability of including the refinement that yields a solution increases with respect to hill-climbing search; however, beam-search also disregards the existence of dependent providers. Beam-search relies on domain-dependent tuning of the beam width, which could be time-consuming and might not yield optimal results.

3.2. Fixed-depth Look-ahead

Fixed-depth look-ahead is used in several ILP systems and consists in allowing the system to evaluate clauses which are s or less refinement steps further. By setting the parameter s with a value x ($x \geq 1$) in Algorithm 1, x-step look-ahead is performed. That means that step 2(b)ii adds to $\mathbf{R}$ the set of all refinements $\mathbf{W}$ obtained by x or less applications of ρ to $\mathcal{D}$.

Employing x-step look-ahead has the drawback that non-admissible clauses (e.g. clauses which contain a dependent provider as the last literal such as $\mathcal{C}_3$ in Figure 1) are considered. As described in the introduction, the evaluation function cannot adequately estimate the value of such clauses. Furthermore, as shown in Section 5.2, fixed-depth look-ahead may incur in significantly longer run times without gain in accuracy.

3.3. Template-based Look-ahead

Template-based look-ahead relies on user-defined templates to perform a selective look-ahead (e.g., relational clichés by Silverstein and Pazzani (1991) and TILDE's look-ahead by Blockeel and De Raedt (1997)). A drawback of this approach is that the templates have to be hand-written by the user; basically, the user has to provide a template for every provider-consumer match. In addition, the system can, but does not have to, use the templates provided to refine a clause and by doing this, non-admissible clauses are considered.

3.4. Determinate Literals

Determinate literals (Quinlan, 1991) work as follows. In a first step, all determinate literals are added to the clause to be refined. In a second step, refinements containing consumers of these literals are evaluated, and the best one is selected. Finally, all determinate literals without a consumer are removed from the clause.

Determinate literals are the subset of dependent providers which are uniquely satisfied by all the examples. Determinate literals' approach requires determinacy in the application domain. This domain property implies that there must be a one to one mapping between input and output argument values in the literals. However, determinacy is not a property present in many application domains. For example, in the eastward trains domain, a train can have multiple cars, or in the mesh design domain, a node has multiple neighbours.

4. Macros in Hill-Climbing Search

In this section, we first review macros and then explain how they are used with hill-climbing search.

4.1. Macro-operators

Macros were introduced by Peña Castillo and Wrobel (2002) as a formal approach to reduce the search space (when using exhaustive search) defined by a downward refinement operator without further restricting the hypothesis space. Macros are based on provider-consumer relations between the literals. In addition to dependent providers, macros introduce the concept of dependent consumers, where a *dependent consumer* is a literal which has at least one input argument value provided exclusively by dependent providers. A *macro* is then either a literal which is neither a dependent provider nor a dependent consumer, or a subsequence of literals with at least one consumer for every dependent provider in the subsequence. A subsequence of literals contains a subset of the literals in the bottom clause.

A macro-based refinement operator refines a clause by adding macros instead of single literals and only generates admissible clauses. Admissible clauses are clauses which do not contain unneeded literals[1] and are also called *legal subsequences of literals*. Macros are automatically created based on a list of (user declared or automatically determined) dependent providers. The macros construction algorithm we use differs from the one presented in (Peña Castillo & Wrobel, 2002) in the generation of macros with more than one dependent provider.[2] By employing a macro-based refinement operator (ρ_M) in step 2(b)i of Algorithm 1, we obtain macro-based hill-climbing.

4.2. Hill-climbing with Macros

Let us illustrate how macros alleviate the myopia problem of hill-climbing using again the student loan domain. In this domain, literals having as predicate either **longest_absence_from_school** or **enrolled** are dependent providers. Based on these dependent providers, seven macros are obtained using the bottom clause listed in Section 2. These seven macros are $\mathbf{MSet} = \{1, [2,6], [2,9], [3,5], [3,8], [4,7], [4,10]\}$.[3]

As depicted in Figure 3, in the first iteration, macro-based hill-climbing evaluates all the refinements al-

[1] An unneeded literal can be removed from a clause without affecting the clause's coverage and consistency.

[2] This complete and correct algorithm for constructing macros is given in Peña Castillo (2004).

[3] Recall that the literals are referred with the number corresponding to their position in $\perp$ (see Figure 2).

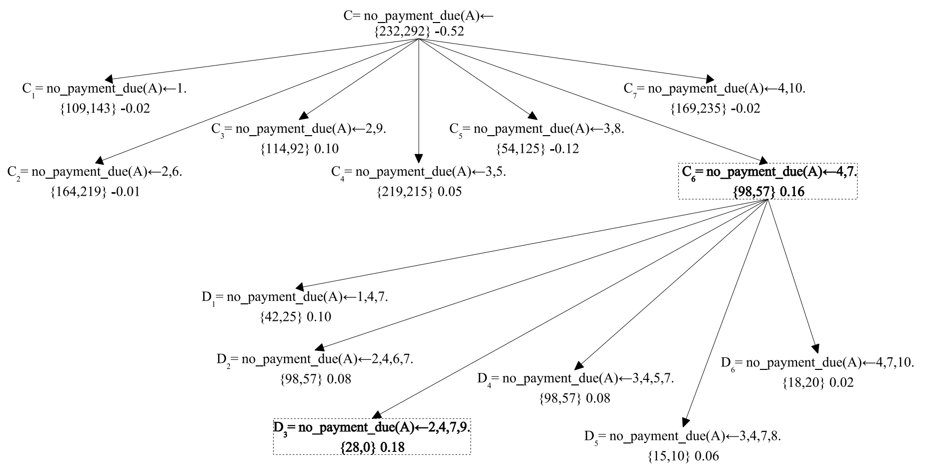

C= no_payment_due(A)←
{232,292} -0.52
C1= no_payment_due(A)←1.
{109,143} -0.02
C2= no_payment_due(A)←2,6.
{164,219} -0.01
C3= no_payment_due(A)←2,9.
{114,92} 0.10
C4= no_payment_due(A)←3,5.
{219,215} 0.05
C5= no_payment_due(A)←3,8.
{54,125} -0.12
C6= no_payment_due(A)←4,7.
{98,57} 0.16
C7= no_payment_due(A)←4,10.
{169,235} -0.02
D1= no_payment_due(A)←1,4,7.
{42,25} 0.10
D2= no_payment_due(A)←2,4,6,7.
{98,57} 0.08
D3= no_payment_due(A)←2,4,7,9.
{28,0} 0.18
D4= no_payment_due(A)←3,4,5,7.
{98,57} 0.08
D5= no_payment_due(A)←3,4,7,8.
{15,10} 0.06
D6= no_payment_due(A)←4,7,10.
{18,20} 0.02

Figure 3. Search space explored by macro-based hill-climbing. A square encloses the clause chosen at each iteration.

lowed by the hypothesis language which are obtained by adding to $\mathcal{C}$ =**no_payment_due(A)**← the generated macros. From those refinements, $\mathcal{C}_6$ obtains the highest heuristic value and is selected in step 2d. In the second iteration, in step 2b, $\mathcal{C}_6$ is refined by adding the available macros, and in step 2d, $\mathcal{D}_3$ is selected. Since $\mathcal{D}_3$ cannot be further refined, the algorithm terminates and returns $\mathcal{D}_3$, which is a solution and is then added to the final theory.

Macros alleviate hill-climbing's myopia problem because Equation 1 is applied to clauses whose quality can be more accurately assessed because they have discriminative power and can be a solution. By using macros, an automatically adjusted variable-depth look-ahead is performed where only admissible clauses are considered.

5. Empirical Evaluation

In this section we empirically analyze the performance of the approaches described above in terms of classification error and run time. We carried out experiments on the following application domains.

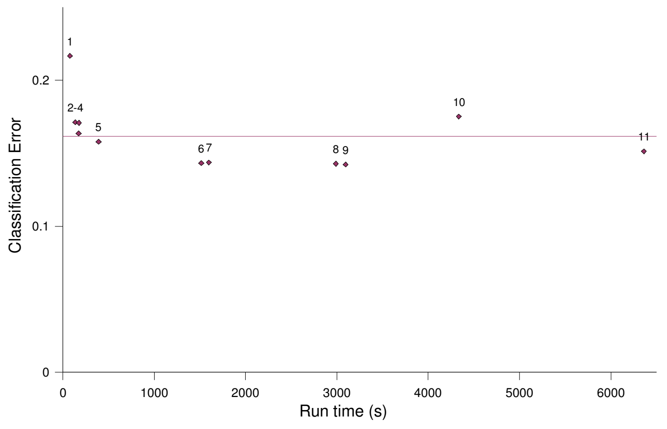
1. Chess moves. This dataset contains 180 positive and 17 negative examples of valid chess moves for five pieces, and the learning task is to learn what are the correct moves for those pieces.
2. Eastbound trains. The system has to determine the direction (east or west) of trains based on their attributes. This dataset contains 42 positive and 19 negative examples.
3. Student loan (Pazzani & Brunk, 1991). This dataset consists of 643 positive and 357 negative examples. The learning task is to discriminate between individuals who are not required to pay back an educational loan and those who must pay.
4. Mutagenesis (Srinivasan et al., 1996). This is an ILP benchmark dataset with 188 compounds (125 positive and 63 negative examples). The mutagenesis problem deals with the prediction of the mutagenic activity of small, heterogeneous molecules.
5. Mesh design (Dolšak et al., 1998). Stresses in physical structures are analyzed by approximating the structures with a mesh model. The task is to determine the appropriate number of elements N on an edge. The data contains information about the edges of ten different structures.
6. Traffic problem detection (Džeroski et al., 1998). This dataset has 256 examples of traffic situations in road sections, which have to be classified as accident, congestion or non-critical section.

We used 5-fold-cross-validation for the first three datasets, 10-fold-cross-validation for mutagenesis and traffic, and cross-validation-leave-one-out for mesh.

5.1. Results Summary

In our experiments, hill-climbing with macros (MioGM), hill-climbing, fixed-depth look-ahead and beam-search were performed using Algorithm 1. For determinate literals and template-based look-ahead, we carried out experiments with FOIL (Quinlan & Cameron-Jones, 1995) version 6.4 and with TILDE (Blockeel & De Raedt, 1998) contained in ACE 1.1.15. To provide a comparison with exhaustive search, we also included Progol (Muggleton, 1995) in the experiments.[4] We defined the templates needed by TILDE to perform look-ahead based on the macros

[4]The settings and mode declarations used for MioGM, TILDE and Progol are available on request; for FOIL the defaults values were used.



Point	Approach		Run Time	Error
1	Hill-climbing	-	76s	0.217
2	Beam-search	$(b = 5)$	135s	0.171
3	Fixed-depth L-ahead	$(s = 2)$	171s	0.164
4	**MioGM**	-	175s	0.171
5	Beam-search	$(b = 20)$	390s	0.158
6	Beam-search	$(b = 80)$	1514s	0.143
7	**MioGM**	$(s = 2)$	1598s	0.144
8	Beam-search	$(b = 160)$	2990s	0.143
9	**MioGM**	$(s = 3)$	3096s	0.142
10	Fixed-depth L-ahead	$(s = 3)$	4336s	0.175
11	Fixed-depth L-ahead	$(s = 4)$	6362s	0.151

Figure 4. Average classification error and running times across all six datasets using MioGM, hill-climbing, fixed-depth look-ahead and beam-search.

Table 1. Win-loss-tie comparison between MioGM and the other approaches in terms of the no. of domains with a significant ($\alpha = 0.05$) difference in accuracy

MioGM vs	**Win-loss-tie**
TILDE (template-based)	3-0-3
Progol (exhaustive search)	4-0-2
FOIL (det. literals)	5-0-1
Hill-climbing	5-0-1
Fixed-depth L-ahead ($s = 2$)	2-0-4
Beam-search ($b = 80$)	0-0-6

created by MioGM. This was done for every dataset except for mutagenesis and mesh where the templates recommended by the author of TILDE were used. MioGM was run with look-ahead ($s = 2$).

Table 1 shows a win-loss-tie accuracy comparison between MioGM and all the other approaches[5]. This Table shows that MioGM is more accurate than all the other approaches but beam-search, with 0.95% statistical significance. Next we analyze in detail our empirical results.

5.2. Comparing Macros, Fixed-depth Look-ahead and Beam-Search

To compare macros, fixed-depth look-ahead and beam-search, we ran MioGM and Algorithm 1 with different values for the parameters b and s. Figure 4 reports the results obtained by MioGM with s set to 1, 2 and 3, and b to 1; and Algorithm 1 with eight different parameter settings: using hill-climbing ($s = 1$ and $b = 1$), with 2, 3 and 4-step look-ahead ($s = 2 \dots 4$ and $b = 1$), and using beam-search with various beam widths ($s = 1$ and $b = 5, 20, 80, 160$).

Figure 4 shows the average classification error across all six datasets obtained by each parameter setting. The horizontal line at 0.162 represents the average classification error of all settings reported. The table on the right of Figure 4 shows the values of the data points.

In the experiments, hill-climbing (see data point 1 in Figure 4) has, as we expected, the highest classification error. The lowest classification error is obtained using macros with look-ahead (data points 7 and 9) and beam-search with the beam width set to 80 and 160 (data points 6 and 8). However, beam-search has the drawback that the beam width has to be tuned by trial-and-error. In our case, we tried in total six different beam width values for every application domain (only four are reported in Figure 4).

Increasing the amount of fixed-depth look-ahead beyond two (points 10 and 11) does not pay off because of the long running times and the marginal decrease in classification error. In addition, the behaviour of beam-search and macros is more stable than that of fixed-depth look-ahead since there is no descent in accuracy and no oversearching occurs when the beam width or the amount of look-ahead is increased.

5.3. Comparing MioGM to Existing Systems

As Figure 5 and Table 2 show, the classification error of MioGM is lower than that of TILDE, FOIL and Progol, with mesh being an exception, where TILDE obtains the lowest error. According to a t-significance value of $\alpha = 0.005$, MioGM's accuracy is statistically

[5]This table contains the results obtained using fixed-depth look-ahead ($s = 2$) and beam-search ($b = 80$), which are the best settings according to Section 5.2.

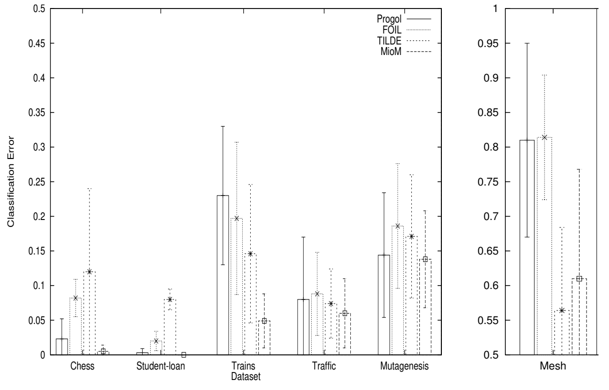


Figure 5. Classification error of Progol (left), FOIL (2nd bar), TILDE (3rd bar), and MioGM (right) per dataset

Table 2. Average error per system per dataset

Dataset	**Progol**	**FOIL**	**TILDE**	**MioGM**
Chess	0.023	0.082	0.118	0.005
S.Loan	0.003	0.018	0.084	0.000
Trains	0.231	0.197	0.146	0.049
Traffic	0.077	0.088	0.074	0.058
Mutag.	0.144	0.186	0.171	0.138
Mesh	0.810	0.814	0.564	0.612

Table 3. Average run time per system per dataset

Dataset	**Progol**	**MioGM**	**TILDE**	**FOIL**
Chess	1.2s	4.6s	1.1s	0.8s
S.Loan	17s	7.1s	1.6s	24.7s
Trains	54s	100s	0.2s	0.1s
Traffic	182s	113s	2.8s	1.7s
Mutag.	1h18m	48m	18.6s	2.2s
Mesh	13h55m	1h48m	35.3s	9.1s

higher than that of FOIL in the chess, student loan, trains, and mesh design domains. MioGM's accuracy is statistically higher than that of TILDE in the chess, student loan, and trains datasets. Finally, MioGM's accuracy is statistically higher than that of Progol in the trains, student loan, and mesh datasets. Thus, macros significantly improve accuracy compared with template-based look-ahead (TILDE) and determinate literals (FOIL), and obtain an average accuracy which positively compares with that obtained by exhaustive search (Progol).

Table 3 shows the average run time on a 500 MHz Sun Blade 100 (128MB of RAM) of Progol, MioGM, TILDE and FOIL per dataset. One can see that macros represent a significant improvement in accuracy with respect to the other systems, and they attain a middle place in terms of running time between the exhaustive system Progol and systems such as TILDE or FOIL. There is reason to believe that MioGM's longer running times are in part due to implementation issues: MioGM is implemented in Java and uses a Java-SICStus Progol interface (Jasper) to execute all Prolog code; while FOIL is implemented in C and TILDE is implemented in ilProlog which is a built-in high performance Prolog system with special features for ILP. To determine the impact of the implementation, we executed a single query to obtain the coverage of the same clause using TILDE and MioGM. TILDE takes on average 0.01s to execute this query, while MioGM takes 0.10s. Generating the macros is a minor time overhead since, on average, it takes 0.1 milliseconds to automatically construct a macro and 130 macros are generated per covering iteration. Hence, macros are not the efficiency bottleneck of the system.

6. Work Related to Macros

The definition of macros may seem similar to k-local clauses, a language bias proposed by Cohen (1995). In this bias, only clauses of locality k or less are considered, where the locality of a clause is the size of the largest set of literals which contain either a free (local) variable X or some free variable influenced by X (this set is called the *locale* of X). However, contrary to the clauses generated by a macro-based refinement operator, a k-local clause might not be a legal subsequence of literals, and thus it does not need to be considered because it cannot be a solution. For example, suppose we search for 3-local clauses. In this case, a clause whose body consists of one dependent provider is considered. However, such a clause is not admissible. Macros, on the other side, does not generates these clauses.

Another work reminiscent of macros is first-order feature construction for individual-centered domains by Lavrač and Flach (2001). This approach is used in 1BC (Flach & Lachiche, 1999) to restrict the search space, and in LINUS and RSD (Lavrač et al., 2003) to propositionalize input data with nondeterminate literals. In both approaches, feature construction and macros, the key idea is to use provider-consumer relations of existential variables among the literals as basis to construct the features or, respectively, the macros. *Structural predicates* in first-order features are binary predicates similar to *structural literals* defined by Zucker and Ganascia (1998) but structural predicates relax the condition of transitivity. Macros' dependent providers relax in addition the requirement of antisymmetry. Macros extend first-order feature construction by allowing head providers and n-ary dependent providers. In addition, macros, contrary to first-order features, are also suitable for non-individual-centered domains or for program synthesis tasks.

7. Conclusions

In this paper, we showed that a learner using hill-climbing search with macro-operators (MioGM) exhibits significantly lower classification error than other greedy systems using other techniques such as fixed-depth look-ahead, template-based look-ahead, or determinate literals. In addition, macros are automatically computed, are less sensitive to domain-dependent tuning of parameters, and do not require determinacy in the application domain.

As future work, we plan to compare hill-climbing using macros with stochastic hill-climbing which is also used to reduce the myopia problem of greedy learners (Fürnkranz, 1999).

Acknowledgments

We thank Oscar Meruvia for many valuable suggestions. The second author was partially supported by the German Science Foundation, projects WR40/1–3 and WR40/2–1.

References

Blockeel, H., & De Raedt, L. (1997). Lookahead and discretization in ILP. *Proc. of the 7th Int. Workshop on ILP* (pp. 77–84).

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence*, *101*, 285–297.

Cohen, W. W. (1995). Pac-learning non-recursive Prolog clauses. *Artificial Intelligence*, *79*, 1–38.

De Raedt, L., & Van Laer, W. (1995). Inductive constraint logic. *Proc. of the 6th Conf. on Algorithmic Learning Theory* (pp. 80–94).

Dolšak, B., Bratko, I., & Jezernik, A. (1998). Application of machine learning in finite element computation. *Machine learning and data mining: methods and applications*, 147–171.

Džeroski, S., & Bratko, I. (1992). Handling noise in inductive logic programming. *Proc. of the 2nd Int. Workshop on ILP*.

Džeroski, S., Jacobs, N., Molina, M., Moure, C., Muggleton, S., & Laer, W. V. (1998). Detecting traffic problems with ILP. *Proc. of the 8th Int. Conf. on ILP* (pp. 281–290).

Džeroski, S., & Lavrač, N. (2001). Introduction to inductive logic programming. In S. Džeroski and N. Lavrač (Eds.), *Relational data mining*, 48–73.

Flach, P., & Lachiche, N. (1999). 1BC: A first-order Bayesian classifier. *Proc. of the 9th Int. Workshop on ILP* (pp. 92–103).

Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review*, *13*, 3–54.

Fürnkranz, J., & Flach, P. (2003). An analysis of rule evaluation metrics. *Proc. of the 20th ICML* (pp. 202–209).

Lavrač, N., & Flach, P. (2001). An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, *2*, 458–494.

Lavrač, N., Železný, F., & Flach, P. (2003). RSD: Relational subgroup discovery through first-order feature construction. *Proc. of the 12th Int. Conf. on ILP* (pp. 149–165).

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*, 245–286.

Pazzani, M. J., & Brunk, C. A. (1991). Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning. *Knowledge Acquisition*, *3*, 157–173.

Peña Castillo, L. (2004). *Search improvements in multirelational learning*. Doctoral dissertation, Otto-von-Guericke-University Magdeburg.

Peña Castillo, L., & Wrobel, S. (2002). Macro-operators in multirelational learning: a search-space reduction technique. *Proc. of the 13th ECML* (pp. 357– 368).

Quinlan, J. R. (1991). Determinate literals in inductive logic programming. *Proc. of the 12th IJCAI* (pp. 746–750).

Quinlan, J. R., & Cameron-Jones, R. M. (1995). Induction of logic programs: FOIL and related systems. *New Generation Computing*, *13*, 287–312.

Silverstein, G., & Pazzani, M. J. (1991). Relational clichés: constraining constructive induction during relational learning. *Proc. of the 8th Int. Workshop on Machine Learning* (pp. 203–207).

Srinivasan, A., Muggleton, S., King, R. D., & Sternberg, M. J. E. (1996). Theories for mutagenicity: a study of first-order and feature based induction. *Artificial Intelligence*, *85*, 277–299.

Zucker, J.-D., & Ganascia, J.-G. (1998). Learning structurally indeterminate clauses. *Proc. of the 8th Int. Conf. on ILP* (pp. 235–244).