CLASSIC'CL: An Integrated ILP System

Christian Stolle, Andreas Karwath, and Luc De Raedt

Albert-Ludwigs Universität Freiburg, Institut für Informatik,
Georges Köhler Allee 79, D-79110 Freiburg, Germany
{stolle, karwath, deraedt}@informatik.uni-freiburg.de

Abstract. A novel inductive logic programming system, called *Classic'cl* is presented. *Classic'cl* integrates several settings for learning, in particular learning from interpretations and learning from satisfiability. Within these settings, it addresses descriptive and probabilistic modeling tasks. As such, *Classic'cl* (C-armr, cLAudien, icl-S(S)at, ICl, and CLl-pad) integrates several well-known inductive logic programming systems such as Claudien, Warmr (and its extension C-armr), ICL, ICL-SAT, and LLPAD. We report on the implementation, the integration issues as well as on some experiments that compare *Classic'cl* with some of its predecessors.

1 Introduction

Over the last decade, a variety of ILP systems have been developed. At the same time, some of the most advanced systems such as Progol [12, 16] and ACE [3] can solve several different types of problems or problem settings. ACE induces rules (as in ICL [7]), decision trees (as in TILDE [1]) and frequent patterns and association rules (as in Warmr [8]). However, most of the present ILP techniques focus on predictive data mining setting and also deal with the traditional learning from entailment setting [4]. The key contribution of this paper is the introduction of the system *Classic'cl*, which learns from interpretations in a descriptive setting. The key novelty is that it tightly integrates several descriptive ILP, such as Claudien [5], Warmr [8], C-armr [6], and LLPADs [15]. This is realized using a generalized descriptive ILP algorithm that employs conjunctive constraints for specifying the clauses of interest. A wide variety of constraints is incorporated, including minimum frequency, exclusive disjunctions, and condensed representations [6]. By combining constraints in different ways, *Classic'cl* can emulate Warmr, Claudien, C-armr and LLPADS as well as some novel variations. *Classic'cl* is derived from the implementation of C-armr [6]. The performance of *Classic'cl* is experimentally compared with some of its predecessors, such as ACE and Claudien. In addition to the descriptive setting, *Classic'cl* also includes a predictive learning setting that emulates the ICL system [7]. This setting is not covered in this paper.

This paper relies on some (inductive) logic programming concepts. The reader unfamiliar with this terminology is referred to [13] for more details.

In the following section we introduce general constraints for the descriptive ILP problem and show how known algorithms can be expressed in this formalism.

A. Hoffmann, H. Motoda, and T. Scheffer (Eds.): DS 2005, LNAI 3735, pp. 354–362, 2005.

A general algorithm to tackle this problem is presented in 3, some implementational issues are described in section 4 and experiments are presented in section 5. We conclude in section 6.

2 The Descriptive ILP Problem

2.1 Constraint Based Mining Problem

Mannila and Toivonen [11] formalized the task of data mining as that of finding the set $Th(Q, D, \mathcal{L})$, where Q is a constraint or query, D a data set and $\mathcal{L}$ a set of patterns. $Th(Q, D, \mathcal{L})$ then contains all patterns h in $\mathcal{L}$ that satisfy the constraint Q w.r.t. the data set D, i.e. $Th(Q, D, \mathcal{L}) = \{h \in \mathcal{L} | Q(h, D) = true\}$. When applying this definition of descriptive data mining to ILP, the language $\mathcal{L}$ will be a set of clauses, the data set D a set of examples and Q can be a complex constraint. Clauses are expressions of the form $h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m$ where the h_i and b_j are logical atoms and all variables are universally quantified (cf. appendix in [13]). The learning from interpretations setting is incorporated by many well-known systems such as Claudien, Warmr, C-armr, Farmr, and LLPADS. We therefore choose interpretations as examples. In this paper, an interpretation is a set of ground facts. The above leads to the descriptive ILP problem, which is tackled in this paper:

Given:
- a language $\mathcal{L}_h$ (i.e., a set of clauses)
- a set of interpretations E
- a constraint $cons(h, E) \in \{true, false\}$ where $h \in \mathcal{L}_h$

Find:
- $Th(cons, E, \mathcal{L}_h)$, i.e., the set of clauses $c \in \mathcal{L}_h$ for which $cons(c, E) = true$

Using this generic formulation of descriptive ILP, we can now consider various constraints *cons* as a conjunction of constraints $c_1 \wedge \cdots \wedge c_k$ (e.g frequency, covers, cf. below). Some of the constraints can be monotonic or anti-monotonic, which can be used to prune the search space. A constraint $cons_m$ is monotonic if all specializations of a clause c will satisfy $cons_m$ whenever c does, and a constraint $cons_a$ is anti-monotonic if all generalizations of a clause c will satisfy $cons_m$ whenever c does. As framework for generality we employ Plotkin's θ-subsumption, which is the standard in ILP. It states that a clause c is more general than a clause c' if and only if there exists a substitution θ such that $c\theta \subset C'$.

2.2 Constraints for ILP

Motivated by constraints used in Claudien, Warmr, C-armr, and LLPAD, *Classic'cl* employs constraints defined on clauses of the form $h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m$:

1. *query* is *true* iff the head of the clause is empty, i.e., if $n = 0$. This constraint is built-in in systems searching for frequent queries such as Warmr and C-armr.

2. $covers(e)$ is $true$ for an interpretation $e \in E$ iff $\leftarrow b_1 \wedge \cdots \wedge b_m$ succeeds in e, i.e. if there is a substitution θ s.t. $\{b_1\theta, \ldots, b_m\theta\} \subseteq e$. E.g., $\leftarrow drinks(X), beer(X)$ covers $\{drinks(vodka)$, $liquor(vodka)$, $drinks(duvel)$, $beer(duvel)\}$. This constraint is often used in the case of queries (i.e., where $n = 0$).
3. $satisfies(e)$ is $true$ iff $h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m$ satisfies $e \in E$, i.e., iff $\forall\theta$: $\{b_1\theta, \ldots, b_m\theta\} \subseteq e \rightarrow \{h_1\theta, \ldots, h_n\theta\} \cap e \neq \emptyset$, e.g. the clause $beer(X) \leftarrow drinks(X)$ does not satisfy the interpretation $\{drinks(vodka)$, $liquor(vodka)$, $drinks(duvel)$, $beer(duvel)\}$ but does satisfy $\{drinks(duvel)$, $beer(duvel)\}$.
4. $xor(e)$ is $true$ iff for any two $h_i \neq h_j$ there exist no substitutions θ_1 and θ_2 such that $\{b_1\theta_1, \ldots, b_m\theta_1, h_i\theta_1\} \subseteq e$ and $\{b_1\theta_2, \ldots, b_m\theta_2, h_j\theta_2\} \subseteq e$. The xor constraint specifies that at most one literal in the head of the clause can be $true$ within the interpretation e.
5. $freq(cons, E) = |\{e \in E | cons(e)\}|$ specifies the number of examples e in E for which the constraint $cons(e)$ is $true$. This is typically used in combination with the constraints $satisfies$ or $covers$.
6. $maxgen$ is $true$ iff $h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m$ satisfies the monotonic part of the rest of the constraint $cons$ and no clause $h_1 \vee \cdots \vee h_{i-1} \vee h_{i+1} \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m$ satisfies $cons$. This constraint is needed as there may be an infinite number of refinements of such clauses that satisfy a monotonic constraint.
7. *s-free(T)* is $true$, where T is a set of horn clauses, iff there is no range-restricted clause $p \leftarrow b'_1 \wedge \cdots \wedge b'_k$ where all $b'_i \in \{b_1, \ldots, b_m\}$ and $p \in \{b_1, \ldots, b_m\} - \{b'_1 \wedge \cdots \wedge b'_k\}$ for which $T \models p \leftarrow b'_1 \wedge \cdots \wedge b'_k$. So no redundancies are induced w.r.t. a background theory T that specifies properties of the predicates (cf. [6]). E.g. $T = \{leq(X, Z) \leftarrow leq(X, Y), leq(Y, Z)\}$ (transitivity) averts clauses such as ($\leftarrow leq(X, Y)$, $leq(Y, Z), leq(X, Z)$) as the last literal is redundant.
8. $free(E)$ is $true$ iff there is no range-restricted clause $p \leftarrow b'_1 \wedge \cdots \wedge b'_k$ where all $b'_i \in \{b_1, \ldots, b_m\}$ and $p \in \{b_1, \ldots, b_m\}$ and $p \neq b_i$ for which $freq(p \leftarrow b'_1 \wedge \cdots \wedge b'_k, satisfies, E) = |E|$. This assures that there are no redundant literals given the data. E.g., given the interpretation $I := \{beer(duvel)$, $alcohol(duvel)$, $alcohol(vodka)\}$, the clause $\leftarrow beer(X)$ is free while $\leftarrow beer(X) \wedge alcohol(X)$ is not free, as the clause $alcohol(X) \leftarrow beer(X)$ is satisfied by I (cf. [6]).
9. *δ-free(E)* is $true$, where δ is a natural number, iff there is no range-restricted clause $p \leftarrow b'_1 \wedge \cdots \wedge b'_k$ where all $b'_i \in \{b_1, \ldots, b_m\}$ and $p \in \{b_1, \ldots, b_m\} - \{b'_1 \wedge \cdots \wedge b'_k\}$ for which $freq(p \leftarrow b'_1 \wedge \cdots \wedge b'_k, satisfies, E) \geq |E| - \delta$. It is not required that the rule perfectly holds on the data, but only that it holds approximately, as δ exceptions are allowed (cf. [6]).
10. $consistent(T)$ is $true$, where T is a set of horn clauses, if and only if $T \cup \{h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m\} \not\models \Box$, i.e., if it is satisfiable. E.g., consider the theory $T = \{\leftarrow parent(X, X)\}$ which specifies that no one is its own parent. Any clause containing this literal is not consistent with respect to T.

The above specified constraints have the following properties: $freq(h, cons_m, E) > t$ and $satisfies$ are monotonic, while $covers$, $query$,

$consistent$, $s-free$, $free$, δ-free, and $freq(h, cons_a, E) > t$ are anti-monotonic. xor is anti-monotonic w.r.t. the head only, i.e., xor is anti-monotonic w.r.t. a fixed body. Clauses with an empty head always satisfy the xor constraint. Therefore, this constraint only applies when refining the heads of clauses. The $maxgen$ constraint is is neither monotonic nor anti-monotonic. Therefore, it will require special attention in our algorithm.

2.3 Existing Descriptive ILP Systems

Claudien [5] essentially searches for all maximally general clauses that satisfy a set of interpretations. This corresponds to using the constraint $cons = maxgen \wedge freq(satifies, E) = |E|$. E.g., given the interpretation $I = \{vodka(smirnov)$, $beer(duvel)$,$alcohol(smirnov)$, $alcohol(duvel)\}$ and a language bias over the literals in I, one would find the following clauses: $\{beer(X) \vee vodka(X) \leftarrow alcohol(X); \leftarrow beer(X) \wedge vodka(X); alcohol(X) \leftarrow vodka(X); alcohol(X) \leftarrow beer(X)\}$.

Warmr [8] extends the well-known Apriori system to a relational data mining setting. It employs essentially the constraints $cons = query \wedge freq(covers, E) > t$. In the example above $(t = 1)$ these queries would be generated: $\{\leftarrow beer(X); \leftarrow vodka(X); \leftarrow alcohol(X); \leftarrow beer(X) \wedge alcohol(X); \leftarrow vodka(X) \wedge alcohol(X)\}$.

C-armr [6] is a variant of Warmr that extends Warmr with condensed representations. Additional constraints that can be imposed include $free$, $s - free$, $consistent$ and $\delta - free$. On the same example, and having the additional constraint $free$, the following queries would be generated. $\{\leftarrow beer(X); \leftarrow vodka(X); \leftarrow alcohol(X)\}$.

CLLPAD combines ideas from Claudien with probabilistic ILP. It essentially mines for LPADS, [17]. These consists of annotated clauses of the form $(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow b_1 \wedge \cdots \wedge b_m$. The $\alpha_i \in [0, 1]$ are real-valued numbers, s.t. $\sum_{i=1}^{n} \alpha_i = 1$. The head atoms h_i of the clauses fulfill the xor constraint, such that for each interpretation at most one h_i is $true$ with a certain probability. This ensures that the clauses c_i of an LPAD P can be considered independently as in traditional inductive logical programs.

$$cons = maxgen \wedge \bigwedge_{e \in E} xor(e) \wedge freq(satisfies, E) = |E| \wedge freq(covers, E) \geq 1$$

Notice that the xor constraint together with $satisfies$ actually implies $maxgen$, so that the CLLPAD can be considered a specialization of the Claudien setting. This constraint is imposed in an early system inducing LPADs, LLPAD [15]. The annotated clauses satisfying $cons$ are then composed to LPADs in a post-processing step (cf. [15]). E.g., consider the following interpretations $\{beer(duvel)$, $alcohol(duvel)\}$ and $\{vodka(smirnov)$, $alcohol(smirnov)\}$. The clauses $\{0.5 : vodka(X) \vee 0.5 : beer(X) \leftarrow alcohol(X); 1.0 : alcohol(X) \leftarrow vodka(X); 1.0 : alcohol(X) \leftarrow beer(X)\}$ would satisfy the constraints. As in [15] the rules get annotated using the equation $\alpha_i = \frac{\sum_{e \in E, satisfies(h_i \leftarrow b_1 \wedge \cdots \wedge b_n, e)} \pi_P^*(e)}{\sum_{e \in E, covers(\leftarrow b_1 \wedge \cdots \wedge b_n, e)} \pi_P^*(e)}$, where the $\pi^*_P(E)$ denotes the probabilities of the interpretations specified in the data set. So the probability of h_i is the sum of probabilities of the interpretations which are covered by $h_i \wedge b$ divided by the sum of probabilities of the interpretations which are covered by b.

The usage of these constraints opens the possibility for several new combinations:

- introduction of condensed representations within the Claudien and CLLPAD setting. The effect of constraints as $free$, $\delta - free$, and $s - free$ is that less patterns are found, that they are typically found more efficiently, and also that (for $free$ and $s - free$) only redundant and undesirable clauses are pruned away, without affecting the semantics of the solution set.
- the original implementation of LLPAD, as described in [15], does not seem to allow for the use of variables in clauses, which essentially corresponds to a propositional version of LLPAD. In contrast, the version in*Classic'cl* does allow for variabelized clauses.
- new combinations, combining, e.g., $freq(satisfies, E)$, $freq(covers, E)$ and δ-free, now become possible.

3 The Descriptive ILP Algorithm

By now we are able to specify the algorithm. We will first discover all bodies that satisfy the constraints, and then expand these into those clauses that satisfy also the head. The algorithm employs two different phases for realizing that. The first phase employs a body refinement operator ρ_b, which merely refines the body of a clause whereas the second phase employs a head refinement operator ρ_o, which merely refines the head by adding literals to the conclusion part of clauses.

Algorithm 1 The generic function **body**$(cons, E)$.

$C_0 := \{false \leftarrow true\}$; $i := 0$; $F_0 := I_0 := \emptyset$
while $C_i \neq \emptyset$ **do**
  $F_i := \{c \in C_i | cons_a(c, E)\}$
  if $cons$ does not contain the constraint $query$ **then**
    call head$(cons, F_i)$
  else
    output $\{f \in F_i | cons_m(f, E)\}$
  end if
  $I_i := C_i - F_i$
  $C_{i+1} := \{b' \mid b \in F_i \text{ and } b' \in \rho_b(b) \text{ and } \neg\exists s \in \bigcup_j I_j : s \preceq b'\}$
  $i := i + 1$
end while

The *body* function (algorithm 1) is very similar to a traditional level wise search algorithm such as Warmr. It starts from the empty query and repeatedly refines it – in a level wise fashion – until the anti-monotonic $cons_a$ part of the constraint $cons$ no longer holds on candidate clauses. The algorithm does not only keep track of the clauses satisfying the anti-monotonic constraint $cons_a$ (on the F_i) but also of the negative border (using the I_i). This is useful for pruning because – when working with a language bias specified using rmodes (cf. below) – not all clauses in the θ-subsumption lattice are within the language $\mathcal{L}_h$, i.e. the language $\mathcal{L}_h$ is not anti-monotonic. Consider for instance the clause $p(K) \leftarrow benzene(K, S) \wedge member(A, S) \wedge atom(K, A, c)$. Even though this clause will typically satisfy the syntactic constraints, its generalization $p(K) \leftarrow member(A, S)$ will typically not be mode-conform. Furthermore, when a new candidate is generated, it is tested whether the candidate is not subsumed by an already known infrequent one.

Algorithm 2 The generic function **head**($cons, F$).

$C_0 := F$; $i := 0$; $S_0 := I_0 := \emptyset$
while $C_i \neq \emptyset$ **do**
  $S_i := \{c \in C_i | cons_m(c, E)\}$
  if *cons* does contain the constraint *maxgen* **then**
    $I_i := C_i - S_i$
    $S_i := \{c \in S_i | \neg\exists s \in \bigcup_j S_j : s \preceq c\}$
  else
    $I_i := C_i$
  end if
  $C_{i+1} := \{c' \ | c \in I_i \text{ and } c' \in \rho_h(c) \text{ and } cons_a(c', E), \}$
  $i := i + 1$
end while
output $filter(\cup_i S_i)$

The interesting and new part of the algorithm is concerned with the function *head* (algorithm 2). This part is used if $query \notin cons$, and one searches for proper clauses, not just queries. The algorithm then proceeds as follows. The *head* function is invoked using the call $head(cons, F)$ for every body. Within the procedure only the head is changed using a head refinement operator ρ_h (which adds literals to the head). Within this context, the algorithm *head* is similar in spirit to the level wise algorithm, except that if the constraint *maxgen* is included in *cons*, those clauses that satisfy *cons* are no longer refined. The algorithm employs a list of candidate clauses on C_i. Those candidates satisfying the constraint are put on S_i, the set of solutions. Depending on *maxgen* all candidates on C_i or only those not satisfying *cons* are refined. The algorithm then outputs, according to some output filter (e.g. a filter that annotates the clauses for CLLPAD), all solutions $\cup S_i$.

4 Implementation Issues

Language Bias. Within ILP, $\mathcal{L}_h$ typically imposes syntactic restrictions on the clauses to be used as patterns. Whereas some of the original implementations (such as Claudien [5]) employed complex formalisms such as DLAB, *Classic'cl* uses the now standard mode and type restrictions (rmodes) of ILP.

Optimizations and Optimal Refinement Operators. In order to search efficiently for solutions, it is important that each relevant pattern is generated at most once. For this, optimal refinement operators (using some canonical form) are employed. As *Classic'cl* is based on the original C-armr implementation of [6], it employs the same optimal refinement operator. In a similar way, we have used a canonical form and optimal refinement operator defined for disjunctive head literals with a fixed body. As computing constraints like frequency are computationally expensive, we have employed the same optimizations as in [6], the system is equally designed as a *light* Prolog implementation that is small but still reasonably efficient.

5 Experiments

The aim was to 1) investigate the performance of *Classic'cl* w.r.t the original implementations, and 2) show that we can tackle some new problem settings.

Datasets. We used artificial and real-world datasets. As artificial datasets, we used the Bongard 300 and 6013 datasets. As real world datasets, we have chosen the Mutagenesis data set [10], the secondary structure prediction dataset from [14], and the SCOP-fold dataset [9].

Warmr and C-armr. First, we compared ACE-Warmr with *Classic'cl*. ACE-Warmr is the original Warmr algorithm in the ACE toolkit [3]. ACE is implemented in a custom build Prolog (iProlog), and can be used with a number of optimizations, like query packs [2]. The results of the comparison can be seen in table 1. The different number of frequent patterns is due to a slightly different language bias and operators. If one takes as criterion time per pattern, then ACE-Warmr and *Classic'cl* are more or less comparable in this experiment.

As a second test, we investigated searching for disjunctive clauses versus searching for horn clauses. This compares to the settings $cons_1 = freq(h, covers, E) > t \wedge query(h) \wedge freq(h, satisfies, E) > t$ to $cons_2 = query(h) \wedge freq(h, covers, E) > t$.

Claudien. We evaluated Classic'cl Claudien compared to the original Claudien implementation using the Mutagenisis and Bongard datasets. All tests we ran on a SUN Blade 1550, as we only had a compiled version for the original Claudien version available. We only mined for horn clauses with a maximum of 5 literals in the Mutagenesis case. This was necessary, as the computational costs proved to be too expensive for the original Claudien. In the case of the Bongard 300 experiment we also restricted the search to definite clauses, as the language bias definition languages rmodes and DLAB are too different to generate comparable results. The results can be found in table 3.

CLLPAD. We employed the LPAD setting and applied it to the SCOP dataset. The test was to evaluate the applicability of the CLLPAD setting to a real world

Table 1. Comparison between the ACE WARMR and *Classic'cl* in the Warmr and C-armr setting on mutagenesis. For the C-armr setting, we chose to employ $\delta - free, s - free$, *consistent* (with $\delta = 0$, t = 2 and maxlevel = 4). ACE-Warmr (packs) denotes the setting for ACE with the option 'use_packs(ilp)'.

	Runtime [secs].	# freq. Patterns
ACE-Warmr(no packs)	12960	91053
ACE-Warmr(packs)	1816	91053
Classic'cl-**Warmr**	5301	194737
Classic'cl-**Carmr()**	4622	124169

Table 2. Comparison between the run times and number of rules for the definite ($cons = query(h) \wedge freq(h, covers, E) > t$) and disjunctive ($cons = query(h) \wedge freq(h, satisfies, E) > t$) search

Data set	**Subset**	**Runtime [s] Horn**	**Disj.**	**# Rules Horn**	**Disj.**	**Factor Horn**	**Disj**
Mutagenesis	188	2602.62	4098.26	893	9099	1.57	10.19
	42	1454.52	1839.45	996	6291	1.26	6.32
	230	3484.94	5339.67	1002	9904	1.53	9.88
Bongard	300	4.78	12.52	54	1628	2.62	30.15
	6013	212.02	1597.97	114	2610	7.54	22.89
Sec. Structure	alpha	75414.4	76950.51	1188	18145	1.02	15.27
	beta	162.79	188.11	111	16768	1.16	151.06
	coil	55102.04	55827.35	1186	18146	1.01	15.3

Table 3. Comparison between the original Claudien and the Classic'cl in the Claudien setting. The differences in the number of rules found is due to the different language bias used (DLAB vs. rmodes). To avoid the comparison between the different setting we also present the time spent by the two implementations producing a rule in seconds per rule. Classic'cl clearly outperforms the original algorithm.

Dataset	**Subset**	**Level**	**Runtime [s] Orig.**	**Classic**	**# Rules Orig.**	**Classic**	**Sec. p. rule Orig.**	**Classic**	**Factor**
Mutagenesis	188	4	66631.9	3290.6	262	308	254.32	10.68	23.8
	42	4	12964.3	1214.41	123	303	105.40	4.01	26.3
	230	4	86022.3	4490.62	279	418	308.32	10.74	28.7
Bongard	300	5	71.53	14.44	32	51	2.24	0.28	7.89

database. The initial set of clauses, *Classi'cl* took 5,714 seconds to construct. Applying the post processing filter solving the CSP took 5,742 seconds and resulted in 33 LPADs build from 18 horn clauses and 7 annotated disjunctive clauses. The disjunctive clauses produced, all center around three folds, name fold1, fold37, and fold55. For space limitations, detailed results are omitted from this paper. This application was impossible with the previous implementation of LLPADs which only employes propositional examples.

To summarize, the experiments clearly show that Classic'cl can indeed simulate its predecessors, that its performance is much better of that of Claudien and despite the light Prolog implementation realistic enough to be applied to real-world data.

6 Conclusions

A novel descriptive data mining approach within the ILP setting of learning from interpretations has been presented. The approach incorporates ideas from constraint based mining in that a rich variety of constraints on target hypotheses can be specified. The algorithm is also incorporated in the system*Classic'cl*, which is able to emulate many of its predecessors such as Claudien, Warmr, c-Armr, CLLPad, as well as ICL and ICL-SAT, as well as some new settings. Classic'cl is implemented in Prolog and it is available from the authors.

References

[1] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *AI*, 101(1-2):285–297, 1998.
[2] H. Blockeel, B. Dehaspe, L.and Demoen, and G. Janssens. Improving the efficiency of inductive logic programming through the use of query packs. *JAIR*, 16:135–166, 2002.
[3] H. Blockeel, L. Dehaspe, J. Ramon, and J.Struyf. Ace - a combined system.
[4] L. De Raedt. Logical settings for concept-learning. *AI*, 95(1):187–201, 1997.
[5] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
[6] L. De Raedt and J. Ramon. Condensed representations for inductive logic programming. In *KR '04*, pages 438–446, 2004.
[7] L. De Raedt and W. Van Laer. Inductive constraint logic. In *ALT95*, volume 997 of *LNAI*. SV, 1995.
[8] L. Dehaspe. *Frequent Pattern Discovery in First-Order Logic.* K. U. Leuven, Dec. 1998.
[9] K. K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards discovering structural signatures of protein folds based on logical hidden markov models. In *PSB 2003*, pages 192–203, 2003.
[10] R. D. King, S. Muggleton, A. Srinivasan, and M. J. E. Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *PNAS*, 93:438–442, 1996.
[11] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. In *Data Mining and Knowledge Discovery*, volume 1, 1997.
[12] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
[13] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic programming*, 17(20):629–679, 1994.
[14] S. Muggleton, R. D. King, and M. J. E. Sternberg. Protein secondary structure prediction using logic. In S. Muggleton, editor, *ILP'92*, Report ICOT TM-1182, pages 228–259, 1992.
[15] F. Riguzzi. Learning logic programs with annotated disjunctions. In *ILP'04*, 2004.
[16] A. Srinivasan. The aleph manual.
[17] J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. Technical report cw386, K.U. Leuven, 2003.