4

An overview of empirical ILP systems

This chapter gives an overview of several empirical ILP systems. Special attention is paid to FOIL, which has been the basis for a substantial part of the work in this book. The systems GOLEM and MOBAL are also briefly described. Some other empirical ILP systems are touched upon for completeness.

4.1 An overview of FOIL

FOIL [Quinlan 1990] extends some ideas from the attribute-value learning paradigm to the ILP paradigm. In particular, it uses a covering approach similar to AQ [Michalski 1983] and an information based search heuristic similar to ID3 [Quinlan 1986]. From MIS [Shapiro 1983] it inherits the idea of top-down search of refinement graphs.

The hypothesis language $\mathcal{L}$ in FOIL is restricted to function-free program clauses. Constants and compound terms may not appear in induced clauses. The body of a clause is a conjunction of literals A or *not* A, where A is an atom. Literals in the body have either a predicate symbol q_i from the background knowledge $\mathcal{B}$, or the target predicate symbol, which means that recursive clauses can be induced. At least one of the variables in the arguments of a body literal must appear in the head of the clause or in one of the literals to its left. No literals that bind a variable to a constant are allowed.[1]

Training examples are function-free ground facts (constant tuples). Background knowledge $\mathcal{B}$ consists of extensional predicate definitions

[1]Literals that bind a variable to a constant value have the form $X = value$. The lack of this feature could be circumvented in FOIL by introducing additional predicates into $\mathcal{B}$ which have the form $is\_a\_value(X)$ for each possible *value*. This feature has been recently added to FOIL4 [Cameron-Jones and Quinlan 1993].

given by a finite set of function-free ground facts, i.e., $\mathcal{B} = \mathcal{M}$ where $\mathcal{M}$ denotes a ground model.

The FOIL algorithm is basically the same as the generic empirical ILP algorithm from Section 3.2.2. It consists of three basic steps: pre-processing of training examples, hypothesis construction and hypothesis post-processing.

Negative examples are not necessarily given to FOIL. They may be generated in pre-processing, based on the closed-world assumption. Hypothesis construction in FOIL is basically the same as the *covering* loop of Algorithm 3.2. Post-processing of the induced hypothesis is a form of reduced error pruning, briefly mentioned in Section 3.2.2 and described in some more detail in Sections 8.3 and 8.4.

The *covering* loop of the FOIL algorithm (Algorithm 4.1) assumes the given (pre-processed) set of training examples $\mathcal{E}$, the language bias $\mathcal{L}$ of function-free program clauses, the extensional background knowledge $\mathcal{B}$, as well as the search heuristics and the heuristics used as the stopping criteria.

Algorithm 4.1 (FOIL – the covering algorithm)

Initialize $\mathcal{E}_{cur} := \mathcal{E}$.
Initialize $\mathcal{H} := \emptyset$.
repeat {covering}
  Initialize clause $c := T \leftarrow$.
  Call the *SpecializationAlgorithm*$(c, \mathcal{E}_{cur})$
    to find a clause c_{best}.
  Assign $c := c_{best}$.
  Post-process c by removing irrelevant literals to get c'.
  Add c' to $\mathcal{H}$ to get a new hypothesis $\mathcal{H}' := \mathcal{H} \cup \{c'\}$.
  Remove positive examples covered by c' from $\mathcal{E}_{cur}$ to get
    a new training set $\mathcal{E}'_{cur} := \mathcal{E}_{cur} - covers_{ext}(\mathcal{B}, \{c'\}, \mathcal{E}^+_{cur})$.
  Assign $\mathcal{E}_{cur} := \mathcal{E}'_{cur}, \mathcal{H} := \mathcal{H}'$.
until $\mathcal{E}^+_{cur} = \emptyset$ or encoding constraints violated.

Output: Hypothesis $\mathcal{H}$.

In the inner *specialization* loop, FOIL seeks a clause of the form

$$p(X_1, X_2, \ldots, X_n) \leftarrow L_1, L_2, ..., L_m$$

Finding a clause consists of a number of refinement steps. Search starts with the clause with an empty body. At each step, the clause $c_i = p(X_1, X_2, \ldots, X_n) \leftarrow L_1, L_2, \ldots, L_{i-1}$ built so far is refined by adding a literal L_i to its body. Literals are atoms A (or negated atoms *not* A) where A is of the form $X_j = X_s$ or $q_k(Y_1, Y_2, ..., Y_{n_k})$, where the X variables appear in c_i, the Y variables either appear in c_i or are new existentially quantified variables, and q_k is a predicate symbol from the background knowledge $\mathcal{B}$. At least one of the Y variables has to appear in c_i. Recursive calls, i.e., atoms of the form $p(Z_1, Z_2, ..., Z_n)$ may also be added to the body of c_i. This requires at least one irreflexive partial ordering $X_s < Z_s$, $1 \leq s \leq n$, to hold in order to prevent the clause from causing infinite recursion (see Chapter 7). A more sophisticated approach to preventing infinite recursion developed along these lines has been recently implemented within FOIL4 [Cameron-Jones and Quinlan 1993].

In the specialization loop, FOIL makes use of a *local training set* which is initially set to the current training set $\mathcal{E}_1 = \mathcal{E}_{cur}$. While $\mathcal{E}_{cur}$ consists of n-tuples, the local training set consists of m-tuples of constants, each of them being a value assignment to the m variables in the current clause. Let $\mathcal{E}_i$ denote the local training set of tuples that satisfy the current clause $c_i = p(X_1, X_2, \ldots, X_n) \leftarrow L_1, L_2, \ldots, L_{i-1}$. Let n_i denote the number of tuples in $\mathcal{E}_i$. Elements of $\mathcal{E}_i^+$ are positive tuples and elements of $\mathcal{E}_i^-$ are negative tuples; the numbers of tuples in these sets are denoted by $n_i^{\oplus}$ and $n_i^{\ominus}$, respectively.[2]

The FOIL specialization loop, implementing the function $SpecializationAlgorithm(c, \mathcal{E}_{cur})$, is given in Algorithm 4.2. In each refinement step, clause c_{i+1} is obtained by adding a literal L_i to the body of the clause $c_i = p(X_1, X_2, \ldots, X_n) \leftarrow L_1, L_2, \ldots, L_{i-1}$, which

[2]In the case that $m = n$, the m-tuples of constants in the local training set $\mathcal{E}_i$ consist of those tuples from $\mathcal{E}_{cur}$ that are covered by c_i. Therefore, the number of examples from $\mathcal{E}_{cur}$ covered by c_i could be denoted by $n(c_i)$. However, since the set contains 'extended' tuples (m can be greater than n), the distinction between the local training set $\mathcal{E}_i$ and the set of examples from $\mathcal{E}_{cur}$ covered by c_i needs to be kept and a different notation used.

Algorithm 4.2 (FOIL – the specialization algorithm)

Initialize local training set $\mathcal{E}_i := \mathcal{E}_{cur}$.
Initialize current clause $c_i := c$.
Initialize $i := 1$.
while $\mathcal{E}_i^- \neq \emptyset$ or *encoding constraints violated* **do**
  Find the best literal L_i to add to the body of $c_i = T \leftarrow Q$
    and construct $c_{i+1} := T \leftarrow Q, L_i$.
  Form a new local training set $\mathcal{E}_{i+1}$ as a set of extensions of
    the tuples in $\mathcal{E}_i$ that satisfy L_i.
  Assign $c := c_{i+1}$.
  Increment i.
endwhile

Output: Clause c.

covers the set of tuples $\mathcal{E}_i$. The literal L_i is either an atom A of the form $q_k(Y_1, Y_2, \ldots, Y_{n_k})$ or $X_j = X_s$, or *not* A. Some of the variables $Y_1, Y_2, \ldots, Y_{n_k}$ belong to the 'old' variables already occurring in c_i, $\{OV_1, \ldots, OV_{Old}\}$, while some are 'new', $\{NV_1, \ldots, NV_{New}\}$, i.e., introduced by the literal L_i. The set of tuples $\mathcal{E}_{i+1}$ covered by clause c_{i+1} is the set of ground $(Old + New)$-tuples (instantiations of $< OV_1, \ldots, OV_{Old}, NV_1, \ldots, NV_{New} >$) for which the body $L_1, L_2, \ldots, L_{i-1}, L_i$ is true. Producing a new training set $\mathcal{E}_{i+1}$ as a set of extensions of the tuples in $\mathcal{E}_i$ that satisfy L_i can be expressed in the relational algebra terminology by saying that $\mathcal{E}_{i+1}$ is the natural join of $\mathcal{E}_i$ with the relation corresponding to literal L_i.

Table 4.1 illustrates the FOIL specialization algorithm on the family relations problem from Section 1.4. The search starts with clause $c_1 = daughter(X, Y) \leftarrow$. The initial local training set $\mathcal{E}_1$ contains all training examples, two of which are positive ($n_1^{\oplus} = 2$) and two negative ($n_1^{\ominus} = 2$). Choosing literal $L_1 = female(X)$ gives rise to the new local training set $\mathcal{E}_2$ of tuples covered by c_2, which contains two positive ($n_2^{\oplus} = 2$) and one negative tuple ($n_2^{\ominus} = 1$). Adding the second literal $L_2 = parent(Y, X)$ to the body of clause c_2 produces clause c_3 with a local training set $\mathcal{E}_3$ containing only positive tuples. Thus, a consistent

Current clause c_1 :	$daughter(X,Y) \leftarrow$				
$\mathcal{E}_1$	$(mary, ann)$	$\oplus$		$n_1^{\oplus} = 2$	$I(c_1) = 1.00$
	(eve, tom)	$\oplus$		$n_1^{\ominus} = 2$	
	(tom, ann)	$\ominus$	$L_1 = female(X)$		
	(eve, ann)	$\ominus$	$Gain(L_1) = 0.84$	$n_1^{\oplus\oplus} = 2$	
Current clause c_2 :	$daughter(X,Y) \leftarrow female(X)$				
$\mathcal{E}_2$	$(mary, ann)$	$\oplus$		$n_2^{\oplus} = 2$	$I(c_2) = 0.58$
	(eve, tom)	$\oplus$		$n_2^{\ominus} = 1$	
	(eve, ann)	$\ominus$	$L_2 = parent(Y,X)$		
			$Gain(L_2) = 1.16$	$n_2^{\oplus\oplus} = 2$	
Current clause c_3 :	$daughter(X,Y) \leftarrow female(X), parent(Y,X)$				
$\mathcal{E}_3$	$(mary, ann)$	$\oplus$		$n_3^{\oplus} = 2$	$I(c_3) = 0.00$
	(eve, tom)	$\oplus$		$n_3^{\ominus} = 0$	

Table 4.1: FOIL trace for the family relation problem.

clause c_3 (which covers no negative tuples) is generated.

If a positive literal with new (existentially quantified) variables is added to the body of the clause, the size (arity) of the tuples in the local training set increases. A tuple from $\mathcal{E}_i$ may also give rise to more than one tuple in $\mathcal{E}_{i+1}$. Consider again the problem of learning family relations, with the first literal $L_1 = parent(Y,Z)$ introducing a new variable Z (see Table 4.2). All of the tuples in $\mathcal{E}_1$ have two successors in $\mathcal{E}_2$, which contains eight 3-tuples. The tuples in $\mathcal{E}_{i+1}$ inherit the labels $\oplus$ and $\ominus$ from their parents in $\mathcal{E}_i$.

The choice of literals is directed by an entropy-based search heuristic, called *weighted information gain*, which can be described as follows. If the local set of tuples $\mathcal{E}_i$ contains n_i tuples, $n_i^{\oplus}$ of which are positive and $n_i^{\ominus}$ negative, the information needed to signal that a tuple is positive is $I(c_i) = -log_2(\frac{n_i^{\oplus}}{n_i}) = -log_2(\frac{n_i^{\oplus}}{n_i^{\oplus}+n_i^{\ominus}})$ (see Section 8.4 which explains the informativity and the information gain heuristics in more detail). Let the choice of the next literal L_i give rise to a new tuple set $\mathcal{E}_{i+1}$; the information needed for the same signal is then $I(c_{i+1}) = -log_2(\frac{n_{i+1}^{\oplus}}{n_{i+1}})$. Note that if L_i does not introduce new variables,

Current clause c_1 : $daughter(X,Y) \leftarrow$				
$\mathcal{E}_1$	$(mary, ann)$	$\oplus$		$n_1^{\oplus} = 2$
	(eve, tom)	$\oplus$		$n_1^{\ominus} = 2$
	(tom, ann)	$\ominus$	$L_1 = parent(Y,Z)$	
	(eve, ann)	$\ominus$		$n_1^{\oplus\oplus} = 2$
Current clause c_2 : $daughter(X,Y) \leftarrow parent(Y,Z)$				
$\mathcal{E}_2$	$(mary, ann, mary)$	$\oplus$		$n_2^{\oplus} = 4$
	$(mary, ann, tom)$	$\oplus$		
	(eve, tom, eve)	$\oplus$		
	(eve, tom, ian)	$\oplus$		
	$(tom, ann, mary)$	$\ominus$		$n_2^{\ominus} = 4$
	(tom, ann, tom)	$\ominus$		
	$(eve, ann, mary)$	$\ominus$		
	(eve, ann, tom)	$\ominus$		

Table 4.2: The effect of new variables on the current training set of tuples in FOIL.

then $\mathcal{E}_{i+1} \subseteq \mathcal{E}_i$, $n_{i+1}^{\oplus} \leq n_i^{\oplus}$ and $n_{i+1}^{\ominus} \leq n_i^{\ominus}$. However, if new variables are introduced, it can happen that $n_{i+1}^{\oplus} > n_i^{\oplus}$ and/or $n_{i+1}^{\ominus} > n_i^{\ominus}$. If $n_i^{\oplus\oplus}$ of the positive tuples in $\mathcal{E}_i$ are represented by one or more tuples in $\mathcal{E}_{i+1}$, the information gained by selecting literal L_i amounts to $Gain(L_i) = WIG(c_{i+1}, c_i) = n_i^{\oplus\oplus} \times (I(c_i) - I(c_{i+1}))$. The literal with the highest gain is selected at each step. $WIG(c_{i+1}, c_i)$ stands for weighted information gain, i.e., the *information gain* $I(c_i) - I(c_{i+1})$ *weighted* by the factor $n_i^{\oplus\oplus}$.

For the example, illustrated in Table 4.1, $I(c_1) = -log_2(\frac{2}{2+2}) = 1.00$, $I(c_2) = -log_2(\frac{2}{2+1}) = 0.58$, and $I(c_3) = -log_2(\frac{2}{2+0}) = 0.00$. As $n_1^{\oplus\oplus} = n_2^{\oplus\oplus} = 2$, we have $Gain(L_1) = 2 \times (I(c_1) - I(c_2)) = 2 \times 0.42 = 0.84$, and similarly $Gain(L_2) = 2 \times 0.58 = 1.16$.

The heuristic allows for efficient pruning, which is responsible for the efficiency of FOIL. Let L_i be a positive literal with $Gain(L_i) = n_i^{\oplus\oplus} \times (I(c_i) - I(c_{i+1}))$. L_i may contain new variables and their replacement with old variables can never increase $n_i^{\oplus\oplus}$. Moreover, such replacement can at best produce a set $\mathcal{E}_{i+1}$ containing only $\oplus$ tuples,

i.e., with $I(c_{i+1}) = 0$. The maximum gain that can be achieved by any literal obtained by substituting new variables in L_i with old is $MaximumGain(L_i) = n_i^{\oplus\oplus} \times I(c_i)$. If this maximum gain is less than the best gain achieved by a literal so far, no literals that can be obtained with such replacement are considered.

To implement noise-handling, FOIL employs stopping criteria based on the *encoding length restriction*, which limits the number of bits used to encode a clause to the number of bits needed to explicitly indicate the positive examples covered by it. If a clause c_i covers $n^{\oplus}(c_i)$ positive examples out of n_{cur} examples in the current training set $\mathcal{E}_{cur}$, the number of bits used to encode the clause must not exceed $ExplicitBits(c_i, \mathcal{E}_{cur}) = log_2(n_{cur}) + log_2(\binom{n_{cur}}{n^{\oplus}(c_i)})$ bits. The construction of a clause is stopped (the *necessity stopping criterion*) when it covers no negative examples (is consistent) or when no more bits are available for adding literals to its body (see Section 8.4). In the latter case, the clause is retained in the hypothesis if it is more than 85% accurate or is discarded otherwise. The search for clauses stops (the *sufficiency stopping criterion*) when no new clause can be constructed under the encoding length restriction, or alternatively, when all positive examples are covered.

Improvements of FOIL, implemented in FOIL2.0 [Quinlan 1991], include *determinate* literals (an idea borrowed from GOLEM), types and mode declarations of predicates, as well as post-processing of clauses. Consider a partially developed clause which has some old variables and a corresponding set of tuples. Determinate literals [Quinlan 1991] are literals that introduce new variables for which each $\oplus$ tuple has exactly one extension and each $\ominus$ tuple has at most one extension in the new training set (see Section 5.6 for the definition of determinacy). Unless a literal is found with gain close to the maximum, all determinate literals are added to the body of the clause under development. Upon completion of the clause, irrelevant literals are eliminated. Determinate literals alleviate some of the problems in FOIL, which are due to its short-sighted hill-climbing search strategy.