SAYPHI-RULES. On learning control knowledge for forward chaining planners and use them stochastically

Daniel Borrajo and **Susana Fernández**
Departamento de Informática
Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
dborrajo@ia.uc3m.es, sfarregu@inf.uc3m.es

Abstract

This paper describes the learning system SAYPHI-RULES for participating in the Learning Track of the 2008 International Planning Competition. Learning is based on the combination of two paradigms: explanation-based learning (EBL) and inductive logic programming (ILP). The learning system is built on a forward state-space planner, SAYPHI, that uses heuristic search. The input to the learning system is the search trees generated by SAYPHI when it solves training problems. And the output, is a set of relational decision trees (heuristics) than guide the planner during the search of new plans, in the same domain, in future problems. SAYPHI-RULES first learns EBL rules. Then, it induces over them by using relational decision trees (ACE software, TILDE component) independently for action and bindings selection. The learned knowledge is used stochastically during the SAYPHI EHC search algorithm with restarts.

Introduction

This paper describes the learning system SAYPHI-RULES. The learning is based on the combination of two well known machine learning paradigms: explanation-based learning (EBL) (Minton *et al.* 1989) and inductive logic programming (ILP) (Muggleton 1992). In particular, we learn relational decision trees using the TILDE system from the ACE software (Blockeel & Raedt 1998). EBL is a deductive learning approach that generates control rules. This learning scheme usually suffers the commonly referred as utility problem, i.e. it learns too many, overly specific rules, which result in the planner spending too much time simply evaluating the rules at the cost of reducing the number of search nodes considered (Minton 1988). Therefore, we induces over the EBL rules by using relational decision trees to alleviate the utility problem. The learned trees are used as control knowledge or heuristics for guiding the planner in the search for solution of new planning problems in the same domain.

The learning system is built on a forward state-space planner, SAYPHI (De la Rosa, García-Olaya, & Borrajo 2007), that uses heuristic search. It re-implements FF (Hoffmann 2001) planner in Lisp. SAYPHI incorporates several search algorithms as enforced-hill-climbing (EHC), A* and hill-climbing branch and bound (DfBnB). The main heuristic is the well-known *optimal relaxed distances* function developed by Bonet and Geffner that accounts for the number of actions to the goal in a relaxed problem where actions' deletes are ignored (Bonet & Geffner 2001). This heuristic performs good in some problems but in others it misleads the search or makes the planner to fall into plateaux. On plateaux, the heuristic value of all successor states is the same as, or worse than, the heuristic value of the current state. The nature of the plateaux encountered, and whether EHC is able to find a path to escape from them, is influenced by the properties of the planning domain (Hoffmann 2005).

The input to the learning system is the search trees generated by SAYPHI when solves training problems using DfBnB, i.e. first, it finds a solution using EHC and then that solution is improved by searching with a backtracking hill-climbing algorithm pruned by the first solution upper-cost bound. It generates positive and negative examples from the search trees. The positive examples are the nodes that lead to the solution and the negative examples are the nodes that do not belong to the solution path. Using EBL the learner generates a control rule from each example for selecting (positive example) or rejecting (negative example) an instantiated operator. The meta-predicates used in the rules are the unsolved goals, the current state and the type of objects. Afterwards, these rules are provided to TILDE to induce decision trees for generating the final knowledge. The learned knowledge is used stochastically with an EHC algorithm with restarts.

In the following sections, we first describe the ILP system TILDE and ACE software. Then, we describe the learning system we have implemented. And finally, last section explains how we have integrated the learned rules into the planner.

The ILP system: TILDE

ACE is a data mining system that provides a common interface to a number of relational data mining algorithms. Relational data mining is the process of finding patterns in a relational database possibly consisting of multiple tables, and extends classical data mining in the sense that in the latter case only patterns within single tuples are found, whereas patterns found by relational data mining systems may ex-

tend over different tuples of different relations. Currently ACE encompasses several algorithms as TILDE that is an upgrade of the decision tree learner C4.5 (Quinlan 1993) towards relational data mining; it builds decision trees that allow to predict the value of a certain attribute in a relation from other information in the database.

TILDE induces logical decision trees that are a first-order logic upgrade of the classical decision trees used by propositional learners (Blockeel & Raedt 1998). It incorporates many features of Quinlan's C4.5, which is a state-of-the-art decision tree learner for attribute-value problems, including the top-down induction of decision trees (TDIDT) algorithm. TDIDT starts from a set of examples and considers all possible tests in the root of the tree and selects the test that performs best according to a certain heuristic (e.g. information gain). It then splits the data set according to the outcome of the test in the examples and it propagates the examples to the resulting subtrees. For each subtree, it then decides whether to turn the subtree into a leaf or to recursively call the induction procedure. This process continues until the tree is completed. Next to these, a number of techniques are used that are specific to ILP: a language bias can be specified (types and modes of predicates), a form of lookahead is incorporated, and dynamic generation of literals (DGL) is possible. The latter is a technique that allows, among other things, to fill in constants in a literal. For learning in numerical domains, a discretization procedure is available that can be used by the DGL procedure to find interesting constants.

The learning system

The SAYPHI-RULES learner is an incremental learning method based on EBL and inductive refinement of relational formulae (control rules). The input to the learner is a task domain (domain ($\mathcal{D}$) and a set of training problems ($\mathcal{P}$).The output is a set of relational decision trees ($\mathcal{K}$) that can be used as control knowledge to steer a forward chaining planner.

The SAYPHI-RULES learner receives a set of training problems. For each training problem in $\mathcal{P}$, it is solved with EHC, and solutions are refined with DfBnB. First, it computes the partial plan for the generated total-order plan, plus the weakest preconditions for each action in the best solution found. For each decision made in the search tree corresponding to the best solution found, one positive example of a rule is generated that contains the regressed state on that node (of the partial plan starting from the action in the node), the target goals and the types of objects. Also, all other nodes are considered negative examples (if they could not be executed in parallel to that action in the partial plan). Then, all examples generated from solving all training problems are translated into the input of TILDE. It calls ACE once to learn which action to select, and one more for each action to learn which bindings to use for that action. All those are separate learning tasks.

Generating control rules with EBL

The module that generates control rules receives as input the solution path of a solved problem. There are two kinds of rules learned from them:

1. SELECT INSTANTIATED-OPERATOR rules: it generates one of these rules for each instantiated operator in the best solution found.
2. REJECT INSTANTIATED-OPERATOR rules: it generates one of these rules for each instantiated operator not in the best solution that cannot be executed in parallel to the previous operator in the partial plan.

The condition part is made of a conjunction of meta-predicates which check for local features of the search process:

- TARGET-GOAL to identify the goals the planner tries to achieve. There is one of this meta-predicate for each goal the planner has not yet achieved.
- TRUE-IN-STATE to test that a literal is true in the planning current state. In order to make the control rules more general and reduce the number of TRUE-IN-STATE meta-predicates, a goal regression is carried out as in most EBL techniques (DeJong & Mooney 1986). Only those literals in the current state which are required, directly or indirectly, by the preconditions of the operator involved in the rule are included.
- TYPE-OF-OBJECT to determine the type of each variable in the rule.

Figures 1 and 2 show examples of SELECT and REJECT rules, respectively, learned in the *matching-bw* domain. The first rule recommends to pick up a negative block with the negative arm and the second rule rejects to pick up a negative block with the positive arm.

```
(IF (and (target-goal (on b4 b1)) (target-goal (on b3 b2))
         (target-goal (on b5 b3)) (target-goal (on b2 b4))
         (true-in-state (hand-positive h1))
         (true-in-state (hand-negative h2))
         (true-in-state (block-positive b2))
         (true-in-state (clear b5))
         (true-in-state (on-table b3))
         (true-in-state (empty h2))
         (true-in-state (on-table b4))
         (true-in-state (clear b4))
         (true-in-state (block-negative b4))
         (true-in-state (block-negative b3))
         (true-in-state (block-negative b5))
         (true-in-state (solid b1)) (true-in-state (solid b4))
         (true-in-state (solid b2)) (true-in-state (solid b3))
         (type-of-object h2 hand) (type-of-object b5 block)
         (type-of-object b3 block) (type-of-object b2 block)
         (type-of-object h1 hand) (type-of-object b4 block)
         (type-of-object b1 block)))
(THEN (select instantiated-operator (pickup h2 b4)))
```

Figure 1: Example of select instantiated-operator rule.

Inductive refinement with TILDE

The examples generated from solving all training problems (EBL rules) are translated into the input of TILDE to find relational patterns of the actions performance. Particularly, to learn which action to select from the set of domain actions,

```
(IF (and (target-goal (on b4 b1)) (target-goal (on b3 b2))
         (target-goal (on b5 b3)) (target-goal (on b2 b4))
         (true-in-state (hand-negative h2))
         (true-in-state (hand-positive h1))
         (true-in-state (empty h1))
         (true-in-state (empty h2))
         (true-in-state (on-table b3))
         (true-in-state (on-table b4))
         (true-in-state (clear b4))
         (true-in-state (clear b5))
         (true-in-state (block-positive b1))
         (true-in-state (block-positive b2))
         (true-in-state (on b5 b1))
         (true-in-state (on b1 b2))
         (true-in-state (on b2 b3))
         (true-in-state (block-negative b3))
         (true-in-state (block-negative b4))
         (true-in-state (block-negative b5))
         (true-in-state (solid b1))
         (true-in-state (solid b2))
         (true-in-state (solid b3))
         (true-in-state (solid b4))
         (true-in-state (solid b5))
         (type-of-object h2 hand) (type-of-object h1 hand)
         (type-of-object b5 block)(type-of-object b4 block)
         (type-of-object b3 block)(type-of-object b2 block)
         (type-of-object b1 block)))
(THEN (reject instantiated-operator (pickup h1 b4)))
```

Figure 2: Example of reject instantiated-operator rule.

and with which bindings. TILDE induces relational decisions trees containing logic queries about the relational facts holding in them. In SAYPHI-RULES, two kind of relational decision trees are learned: one for selecting an action and others (one for action) for choosing their bindings. When building each tree, the learning component receives two inputs:

- The language bias, specifying the restrictions in the variables of the predicates to constrain their instantiation. Figure 3 shows the language bias specified for learning the patterns of the performance of the select action in the *matching-bw* domain. This bias is automatically extracted from the STRIPS domain definition: (1) the classes are the domain actions (2) the types of the target concept are fixed (`select_action(example,class)`) (3) the types of the `true_in_state` and `target_goal` literals are extracted from the predicate definitions and (4) the types of the `types_of_object` literals are extracted from the type definitions. The bias for the action-binding trees are the same except for the target concept. For example, in `pickup` action it would be:

  ```
  % The target concept
  predict(select_bindings_pickup(+Example,+h,+b,-Class)).
  classes([selected,rejected]).
  type(select_bindings_pickup(example,hand,block,class)).
  ```

  The symbols + and − specify whether at the time of testing the truth of the predicate the argument should be bound (or instantiated, it is of type +) or not (it is of type −). One can also combine these modes and write +− stating that all calling patterns are permitted.
- The knowledge base, specifying the set of examples of the target concept (there is no background knowledge). In SAYPHI-RULES, they are extracted from the EBL rules generated solving the training problems. Figure 4 shows a piece of the knowledge base for learning the patterns of performance of the select action. Particularly, this example captures an example with id e0 that resulted in selecting the `pickup` action. The select action execution (example of target concept) is linked with the state literals through an identifier that represents the execution instance, e0. The selected action is also augmented with the label that describes the class of the learning example (pickup, putdown_pos_pos, ...). The knowledge bases for learning the binding patterns are also similar except for the first line of each example that it could be, for example, `select_bindings_pickup(e0,h2,b4,selected).`

```
% The target concept
predict(select_action(+Example,-Class)).
classes([pickup,putdown_pos_pos,putdown_neg_neg,
putdown_pos_neg,putdown_neg_pos,stack_pos_pos,
stack_neg_neg,stack_pos_neg,stack_neg_pos,unstack]).
type(select_action(example,class)).

% The domain predicates
rmode(true_in_state_hand_positive(+Example,+-H)).
type(true_in_state_hand_positive(example,hand)).
rmode(target_goal_hand_positive(+Example,+-H)).
type(target_goal_hand_positive(example,hand)).
........
rmode(type_of_object_block(+Example,+Object)).
type(type_of_object_block(example,object)).
rmode(type_of_object_hand(+Example,+Object)).
type(type_of_object_hand(example,object)).
rmode(type_of_object_object(+Example,+Object)).
type(type_of_object_object(example,object)).
```

Figure 3: Example of language bias for selecting an action.

Each branch of the learned decision tree will represent a pattern of performance of the corresponding select action or select/reject bindings for an action:

- the internal nodes of the branch contain the set of conditions under which the pattern of performance is true.
- the leaf nodes contain the corresponding class; in this case, the performance of the select action or the select/reject bindings and the number of examples covered by the pattern.

Figure 5 shows part of the built decision tree where it is recommended to select action `unstack` with 34 examples and to reject some bindings with 101 examples. As we explained above, the types of the target concept for the `select_action` decision-tree specified in the language bias, are `select_action(example,class)` and the types of the rest of predicates are shown in Figure 3. So, in the example, variable `a` stands for the example, `b` for the action (`unstack`) and the rest of variables for the corresponding type fixed in the language bias.

```
%% Example 0
select_action(e0,pickup).
target_goal_on(e0,b4,b1).
target_goal_on(e0,b3,b2).
target_goal_on(e0,b5,b3).
target_goal_on(e0,b2,b4).
true_in_state_hand_positive(e0,h1).
true_in_state_hand_negative(e0,h2).
true_in_state_block_positive(e0,b2).
true_in_state_clear(e0,b5).
true_in_state_on_table(e0,b3).
true_in_state_empty(e0,h2).
true_in_state_on_table(e0,b4).
true_in_state_clear(e0,b4).
true_in_state_block_negative(e0,b4).
true_in_state_block_negative(e0,b3).
true_in_state_block_negative(e0,b5).
true_in_state_solid(e0,b1).
true_in_state_solid(e0,b4).
true_in_state_solid(e0,b2).
true_in_state_solid(e0,b3).
type_of_object_hand(e0,h2).
type_of_object_block(e0,b5).
type_of_object_block(e0,b3).
type_of_object_block(e0,b2).
type_of_object_hand(e0,h1).
type_of_object_block(e0,b4).
type_of_object_block(e0,b1).
```

Figure 4: Knowledge base corresponding to example e0 for selecting an action.

Using the learned control knowledge

The learned knowledge is used stochastically with an EHC algorithm with restarts. When SAYPHI solves a new problem from the same domain where control knowledge was previously learned, EHC decides which node to explore next according to the decision trees recommendation and the heuristic. We use a parameter p to be the probability of using EHC heuristic to decide which node to explore next from the children of a node (we set as 0.4 for the experiments). So, at each node, with probability p it will expand according to EHC, and with probability $1 - p$ it will decide according to the learned decision trees. If it decides according to the learned knowledge, then it will first consult the actions decision tree. It will follow the conditions of the tree until the leave nodes. Once in the leave node, it will decide again stochastically which action to select according to a roulette mechanism. Each action will have a probability of being selected proportional to the number of examples in that decision leave. Once the action has been selected the same procedure is executed for its corresponding bindings decision tree.

Acknowledgements

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05, and a grant from the Spanish MEC. We also thank the chairs of the Learning Track of IPC-08 for making this competition possible.

```
((actions ((select_action a b)) ((true_in_state_empty a c))
  (yes (true_in_state_on a d e)) (yes (true_in_state_clear a d))
  (yes (true_in_state_hand_negative a c))
  (yes (true_in_state_holding a f g))
  (yes (target_goal_on a g h))
  (yes (true_in_state_block_negative a d))
  (yes
   (unstack 34.0
    (pickup 4.0 putdown_pos_pos 0.0 putdown_neg_neg 0.0
     putdown_pos_neg 0.0 putdown_neg_pos 0.0 stack_pos_pos 0.0
     stack_neg_neg 0.0 stack_pos_neg 0.0
     stack_neg_pos 0.0 unstack 30.0)))
.........

 (unstack ((select_bindings_unstack a b c d e))
  ((true_in_state_solid a d))
  (yes (true_in_state_on_table a f))
  (yes (true_in_state_solid a f))
  (yes (true_in_state_on a g f)) (yes (true_in_state_solid a g))
  (yes (target_goal_on a h f)) (yes (true_in_state_solid a h))
  (yes (target_goal_on a i g)) (yes (true_in_state_solid a i))
  (yes (rejected 101.0 (selected 12.0 rejected 89.0)))
```

Figure 5: Examples of relational decision tree induced from *matching-bw* domain.

References

Blockeel, H., and Raedt, L. D. 1998. Top-down induction of first-order logical decision trees. *Artificial Intelligence* 101(1-2):285–297.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*.

De la Rosa, T.; García-Olaya, A.; and Borrajo, D. 2007. Using cases utility for heuristic planning improvement. In *Proceedings of the 7th International Conference on Case-Based Reasoning*, 137–148. Springer Verlag.

DeJong, G., and Mooney, R. J. 1986. Explanation-based learning: An alternative view. *Machine Learning* 1(2):145–176.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI Magazine* 22(3):57 – 62.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Minton, S.; Carbonell, J. G.; Knoblock, C. A.; Kuokka, D. R.; Etzioni, O.; and Gil, Y. 1989. Explanation-based learning: Optimizing problem solving performance through experience. *Artificial Intelligence* 40:63–118.

Minton, S. 1988. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Boston, MA: Kluwer Academic Publishers.

Muggleton, S. 1992. *Inductive Logic Programming*. London: Academic Press Limited.

Quinlan, J. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.