Learning Relational Decision Trees for Guiding Heuristic Planning

Tomás de la Rosa, Sergio Jiménez and **Daniel Borrajo**
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
trosa@inf.uc3m.es, sjimenez@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract

The current evaluation functions for heuristic planning are expensive to compute. In numerous domains these functions give good guidance on the solution, so it worths the computation effort. On the contrary, where this is not true, heuristics planners compute loads of useless node evaluations that make them scale-up poorly. In this paper we present a novel approach for boosting the scalability of heuristic planners based on automatically learning domain-specific search control knowledge in the form of relational decision trees. Particularly, we define the learning of planning search control as a standard classification process. Then, we use an off-the-shelf relational classifier to build domain-specific relational decision trees that capture the preferred action in the different planning contexts of a planning domain. These contexts are defined by the set of helpful actions extracted from the relaxed planning graph of a given state, the goals remaining to be achieved, and the static predicates of the planning task. Additionally, we show two methods for guiding the search of a heuristic planner with relational decision trees. The first one consists of using the resulting decision trees as an action policy. The second one consists of ordering the node evaluation of the Enforced Hill Climbing algorithm with the learned decision trees. Experiments over a variety of domains from the IPC test-benchmarks reveal that in both cases the use of the learned decision trees increase the number of problems solved together with a reduction of the time spent.

Introduction

During the last years, heuristic planning has achieved significant results and has become one of the most popular planning paradigms. However, the current domain-independent heuristics are still very expensive to compute. As a consequence, heuristic planners suffer from scalability problems because they spent most of the planning time computing useless node evaluations. This effect becomes more problematic in domains where the heuristic function gives poor guidance on the true solution (e.g. blocksworld) given that useless node evaluations happen more frequently.

Since STRIPS, Machine Learning has been a successful tool to automatically define domain-specific knowledge that improves the performance of automated planners. In the recent years, we are living a renewed interest in using Machine Learning to automatically extract knowledge that improves the performance of planners, specially targeted towards heuristic ones. Some examples are:

- Learning macro-actions (Botea *et al.* 2005; Coles & Smith 2007): macro-actions result from combining the actions that are more frequently used together. They have been used in heuristic planning to reduce the search tree depth. However, this benefit decreases with the number of new macro-actions added as they enlarge the branching factor of the search tree causing the *utility problem*. To overcome this problem, one can use different filters that decide on the applicability of the macro-actions (Newton *et al.* 2007). This approach does not consider the goals of the planning task, so macros that helped in a given problem may be useless for different ones.
- Learning cases: Cases consist of traces of past solved planning problems. But, unlike macro-actions, cases can memorize goals information. Recently, typed sequences cases have been used in heuristic planning for node ordering during the plan search. They have shown to reduce the number of heuristic evaluations (De la Rosa, García-Olaya, & Borrajo 2007). This technique still relies on the heuristic function, so it is not appropriate for domains where the heuristic is not accurate, such as domains where big plateaus appear, like the Blocksworld.
- Learning the heuristic function: In this approach (Yoon, Fern, & Givan 2006; Xu, Fern, & Yoon 2007), a state-generalized heuristic function is obtained through a regression process. The regression examples consist of observations of the true distance to the goal from diverse states, together with extra information from the relaxed planning graph. This approach is able to provide a more accurate heuristic function that captures domain specific regularities. However, the result of the regression is poorly understandable by humans making the verification of the correctness of the acquired knowledge difficult.
- Learning general policies (Khardon 1999; Martin & Geffner 2004; Yoon, Fern, & Givan 2007): A general policy is a mapping of the problem instances of a given domain (world state plus goals) into the preferred action to be executed in order to achieve the goals. Thereby, a good general policy is able to solve any possible problem instance of a given domain by simply executing the

general policy in every current state without any search process. Given the relational representation of the AI planning task, it is easier to represent, learn and understand relational policies than accurate evaluation functions. Moreover, the techniques for learning general policies also perform well in stochastic versions of the same problems (Fern, Yoon, & Givan 2006).

The work presented in this paper is included in the last group. Particularly, we present a new approach, which we have called ROLLER, for learning general policies for planning by building domain-dependent relational decision trees from the helpful actions of a forward-chaining heuristic planner. These decision trees are built with an off-the-shelf relational classifier and capture which is the best action to take for each possible decision of the planner in a given domain. The resulting decision trees can be used either as a policy to directly solve planing problems or as a guide for ordering node evaluations in a heuristic planning algorithm.

The paper is organized as follows. The first section describes the basic notions in heuristic planning and what is a *helpful context*. The second section explains the concept of relational decision trees and how to learn planning control knowledge with them. The third section shows different ways of how decision trees can be applied to assist heuristic planning. Then, the fourth section presents the experimental results obtained in some IPC benchmarks. The fifth section discusses the related work, and finally the last section discusses some conclusions and future work.

Helpful Contexts in Heuristic Planning

We follow the propositional STRIPS formalism to describe our approach. We define a planning task P as the tuple (L, A, I, G) with L the set of literals of the task, A the set of actions, where each action $a = (pre(a), add(a), del(a))$. I is the set of literals describing the initial state and G the set of literals describing the goals. Under this definition, solving a planning task implies finding a plan $\mathcal{P}$ as the sequence $(a_1, \ldots, a_n)$ that transforms the initial state into a state in which the goals have been achieved.

The concept of *helpful context* relies on some properties of the relaxed plan heuristic and the extraction of the set of *helpful actions*, both introduced in the FF planner (Hoffmann & Nebel 2001). FF heuristic returns the number of actions in the relaxed plan denoted by $\mathcal{P}^+$, which is a solution of the relaxed planning task P^+; a simplification of the original task in which the deletes of actions are ignored. The relaxed plan is extracted from the relaxed planning graph, which is built as a sequence of fact and action layers. This sequence, represented by (F_0, A_0, $\ldots$, A_t, F_t), describes a reachability graph of the applicable actions in the relaxed task. For each search state the length of the relaxed plan extracted from this graph is used as the heuristic estimation of the corresponding state. Moreover, the relaxed plan extraction algorithm marks a set of facts G_i in the planning graph, for each fact layer F_i, as a set of literals that are goals of the relaxed planning task or are preconditions of some actions in a subsequent layer in the graph. Additionally, the set of helpful actions are defined as

$$helpful(s) = \{a \in A \mid add(a) \cap G_1 \neq \emptyset\}$$

The helpful actions are used in the search as a pruning technique, because they are considered as the only candidates for being selected during the search.

Given that each state generates its own particular set of helpful actions, we argue that the helpful actions, together with the remaining goals and the static literals of the planning task, encode a useful context related to each state. Formally, we define the helpful context, $\mathcal{H}(s)$, of a state s as:

$$\mathcal{H}(s) = \{helpful(s), target(s), static(s)\}$$

where $target(s) \subseteq G$ describes the set of goals not achieved in state s ($target(s) = G - s$), and $static(s) = static(I)$ is the set of literals that remain true during the search, since they are not changed by any action in the problem. The aim of defining this context is to determine in later planning episodes, which action within the set of applicable ones should be selected to continue the search.

Learning General Policies with Decision Trees

ROLLER follows a three-step process for learning the general policies building relational decision trees:

1. Generation of learning examples. ROLLER solves a set of training problems and records the decisions made by the planner when solving them.
2. Actions Classification. ROLLER obtains a classification of the best operator to choose in the different helpful contexts of the search according to the learning examples.
3. Bindings Classification. For each operator in the domain, ROLLER obtains a classification of the best bindings (instantiated arguments) to choose in the different helpful contexts of the search according to the learning examples.

The process for the generation of the learning examples is shared by both classification steps. The learning is separated into two classification steps in order to build general policies with off-the-shelf classifiers. The fact that each planning action may have different arguments (in terms of arguments type and arguments number) makes unfeasible, for many classifiers, the definition of only one learning target concept. Besides, We believe that this two-step decision process is also clearer from the decision-making point of view, and helps users to understand the generated policy better by focusing on either the decision on which action to apply, or which bindings to use given a selected action.

Generation of Learning Examples

With the aim of providing the classifier with learning examples corresponding to good quality planning decisions, ROLLER solves small training problems first using the Enforced Hill Climbing algorithm (EHC) (Hoffmann & Nebel 2001), and then, it refines the found solution with a Depth-first Branch-and-Bound algorithm (DfBnB) that increasingly generates better solutions according to a given metric, the plan length in this particular work. The final search tree is traversed and all nodes belonging to one of the solutions with the best cost are tagged for generating training instances for the learner. Specifically, for each tagged node, ROLLER generates one learning example consisting of:

- the helpful context of the node, i.e., the helpful actions extracted from the node siblings plus the set of remaining goals and the static predicates of the planning problem.
- the class of the node. For the *Actions Classification* the class indicates the operator of the node applied action. For the *Bindings Classification* the class indicates whether the node and their siblings of the same operator are part (selected) or not (rejected) of one of the best solutions.

Finally, the gathered learning examples are saved in two different ways, one for each of the two classification processes (actions and bindings).

The Classification Algorithm

A classical approach to assist decision making consists of gathering a significant set of previous decisions and building a decision tree that generalizes them. The leaves of the resulting tree contain the decision to make and the internal nodes contain the conditions over the examples features that lead to those decisions. The common way to build these trees is following the *Top-Down Induction of Decision Trees* (TDIDT) algorithm (Quinlan 1986). This algorithm builds the decision tree repeatedly splitting the set of learning examples by the conditions that maximize the examples entropy. Traditionally, the learning examples are described in an attribute-value representation. Therefore, the conditions of the decision trees represent tests over the value of a given attribute of the examples. On the contrary, decisions in AI planning are described relationally: a given action is chosen to reach some goals in a given context all described in predicate logic.

Recently, new algorithms for building relational decision trees from examples described as logic facts have been developed. This new relational learning algorithms are similar to the propositional ones except that the conditions in the tree consist of logic queries about relational facts holding in the learning examples. Since the space of potential relational decision trees is normally huge, these relational learning algorithms are biased according to a specification of syntactic restrictions called *language bias*. This specification contains the predicates that can appear on the examples, the target concept, and some learning-specific knowledge as type information, or input and output variables of predicates. In our approach all this language bias is automatically extracted from the PDDL definition of the planning domain.

Along this work we used the tool TILDE (Blockeel & De Raedt 1998) for both the action and bindings classification. This tool implements a relational version of the TDIDT algorithm, though we could have used any other off-the-shelf tool for relational classification.

Learning the Actions Tree

The inputs to the actions classification are:

- *The language bias*, that specifies restrictions in the values of arguments of the learning examples. In our case, this bias is automatically extracted from the PDDL domain definition and consists of the predicates for representing the target concept, i.e., the action to select, and the background knowledge, i.e., the helpful context. Figure 1 shows the language bias specified for learning the action selection for the Satellite domain. The target concept is defined by the predicate `selected(+Example,+Exprob,-Class).` where `Example` is the identifier of the decision, `Exprob` is a training problem identifier for sharing static facts knowledge between examples, and `Class` is the name of the action selected. And the helpful context is specified by three types of predicates: The literals capturing the helpful actions `candidate_`A_i, the predicates that appear in the goals `target_goal_`G_i and the static predicates `static_fact_`S_i. All these predicates are extended with extra arguments: `Example` that links the literals belonging to the same example, and `Exprob` that links the static facts belonging to the same problem.

```
% The target concept
predict(selected(+Example,+Exprob,-Class)).
type(selected(example,exprob,class)).
classes([turn_to,switch_on,switch_off,calibrate,take_image]).

% The domain predicates
rmode(candidate_turn_to(+Example,+Exprob,+-A,+-B,+-C)).
type(candidate_turn_to(example,exprob,satellite,direction,
                                                direction)).

rmode(candidate_switch_on(+Example,+Exprob,+-A,+-B)).
type(candidate_switch_on(example,exprob,instrument,satellite)).

rmode(candidate_switch_off(+Example,+Exprob,+-A,+-B)).
type(candidate_switch_off(example,exprob,instrument,satellite)).

rmode(candidate_calibrate(+Example,+Exprob,+-A,+-B,+-C)).
type(candidate_calibrate(example,exprob,satellite,
                                        instrument,direction)).

rmode(candidate_take_image(+Example,+Exprob,+-A,+-B,+-C,+-D)).
type(candidate_take_image(example,exprob,satellite,direction,
                                                   instrument,mode)).

rmode(target_goal_pointing(+Example,+Exprob,+-A,+-B)).
type(target_goal_pointing(example,exprob,satellite,direction)).

rmode(target_goal_have_image(+Example,+Exprob,+-A,+-B)).
type(target_goal_have_image(example,exprob,direction,mode)).

rmode(static_fact_on_board(+Exprob,+-A,+-B)).
type(static_fact_on_board(exprob,instrument,satellite)).

rmode(static_fact_supports(+Exprob,+-A,+-B)).
type(static_fact_supports(exprob,instrument,mode)).

rmode(static_fact_calibration_target(+Exprob,+-A,+-B)).
type(static_fact_calibration_target(exprob,instrument,direction)).
```

Figure 1: Language bias for the satellite domain automatically generated from the PDDL domain definition.

- *The learning examples*. They are described by the set of examples of the target concept, and the background knowledge associated to these examples. As previously explained, the background knowledge describes the *helpful-context* of the action selection. Figure 2 shows one learning example with id `tr01_e1` resulted from a selection of the action `switch-on` and its associated background knowledge for building the actions tree for the satellite domain.

The resulting relational decision tree represents a set of disjoint patterns of action selection that can be used to pro-

```
% Example tr01_e1
selected(tr01_e1,tr01_prob,switch_on).
candidate_turn_to(tr01_e1,tr01_prob,satellite0,phenomenon2,star0).
candidate_turn_to(tr01_e1,tr01_prob,satellite0,phenomenon3,star0).
candidate_turn_to(tr01_e1,tr01_prob,satellite0,phenomenon4,star0).
candidate_switch_on(tr01_e1,tr01_prob,instrument0,satellite0).
target_goal_have_image(tr01_e1,tr01_prob,phenomenon3,infrared2).
target_goal_have_image(tr01_e1,tr01_prob,phenomenon4,infrared2).
target_goal_have_image(tr01_e1,tr01_prob,phenomenon2,spectrograph1).
```

Figure 2: Knowledge base corresponding to the example tr01_e1 obtained solving the training problem tr01_prob from the satellite domain.

vide advice to the planner: *the internal nodes* of the tree contain the set of conditions under which the decision can be made. The *leaf nodes* contain the corresponding class; in this case, the decision to be made and the number of examples covered by the pattern. Figure 3 shows the actions tree learned for the satellite domain. Regarding this tree, the first branch states that when there is a calibrate action in the set of helpful/candidate actions, it was correctly selected in 44 over 44 times, independently of the rest of helpful/candidate actions. The second branch says that if there is no calibrate candidate but there is a take_image one, the planner has finally chosen correctly to take_image in 110 over 110 times and so on for all the tree branches.

```
selected(-A,-B,-C)
candidate_calibrate(A,B,-D,-E,-F) ?
+--yes:[calibrate] 44.0 [[turn_to:0.0,switch_on:0.0,
|                         switch_off:0.0,calibrate:44.0,
|                         take_image:0.0]]
+--no: candidate_take_image(A,B,-G,-H,-I,-J) ?
       +--yes:[take_image] 110.0 [[turn_to:0.0,switch_on:0.0,
       |                            switch_off:0.0,calibrate:0.0,
       |                            take_image:110.0]]
       +--no: candidate_switch_on(A,B,-K,-L) ?
              +--yes:[switch_on] 59.0 [[turn_to:15.0,switch_on:44.0,
              |                         switch_off:0.0,calibrate:0.0,
              |                         take_image:0.0]]
              +--no: [turn_to] 149.0 [[turn_to:149.0,switch_on:0.0,
                                       switch_off:0.0,calibrate:0.0,
                                       take_image:0.0]]
```

Figure 3: Relational decision tree learned for the action selection in the satellite domain.

Learning the Bindings Tree

At this step, a relational decision tree is built for each action in the planning domain. These trees, called bindings trees, indicate the bindings to select for the action in the different planning contexts. The *language bias* for learning a bindings tree is also automatically extracted from the PDDL domain definition. But in this case, the target concept represents the action of the planning domain and its arguments, plus an extra argument indicating whether the set of bindings was selected or rejected by the planner in a given context; Figure 4 shows part of the language bias specified for learning the bindings tree for the action turn_to from the satellite domain.

The *learning examples* for learning a bindings tree consist of examples of the target concept, and their associated background knowledge. Figure 5 shows a piece of the knowledge base for building the bindings tree corresponding to the action turn_to from the satellite domain. This example, with id tr07_e63, resulted in the selection of the action turn_to(satellite0,groundstation0,planet4).

```
% The target concept
predict(turn_to(+Example,+Exprob,+Sat,+Dir1,+Dir2,-Class)).
type(turn_to(example,exprob,satellite,direction,direction,class)).
classes([selected,rejected]).
% The useful-context predicates, the same as in the Actions tree
...
```

Figure 4: Part of the language bias for learning the bindings tree for the turn_to action from the satellite domain.

```
% static predicates
static_fact_calibration_target(tr07_prob,instrument0,groundstation0).
static_fact_supports(tr07_prob,instrument0,thermograph2).
static_fact_on_board(tr07_prob,instrument0,satellite0).

% Example tr07_E63
turn_to(tr07_e63,tr07_prob,satellite0,groundstation0,planet4,selected).
turn_to(tr07_e63,tr07_prob,satellite0,phenomenon2,planet4,rejected).
turn_to(tr07_e63,tr07_prob,satellite0,phenomenon3,planet4,rejected).
candidate_turn_to(tr07_e63,tr07_prob,satellite0,groundstation0,planet4).
candidate_turn_to(tr07_e63,tr07_prob,satellite0,phenomenon2,planet4).
candidate_turn_to(tr07_e63,tr07_prob,satellite0,phenomenon3,planet4).
target_goal_have_image(tr07_e63,tr07_prob,phenomenon2,thermograph2).
target_goal_have_image(tr07_e63,tr07_prob,phenomenon3,thermograph2).
target_goal_have_image(tr07_e63,tr07_prob,planet4,spectrograph0).
```

Figure 5: Knowledge base corresponding to the example tr07_e63 obtained solving the training problem tr07_prob from the satellite domain.

Figure 6 shows a bindings tree built for the action turn-to from the satellite domain. According to this tree, the first branch says that when there is a sibling node that is a turn-to of a satellite C from a location E to a location D with the goal of goal_pointing D, another goal is having an image of E and we have to calibrate C in E, the turn-to has been selected by the planner in 12 over 12 times.

```
turn_to(-A,-B,-C,-D,-E,-F)
candidate_turn_to(A,B,C,D,E),target_goal_pointing(A,B,C,D)?
+-yes:target_goal_have_image(A,B,E,-G)?
|     +--yes: static_fact_calibration_target(B,-H,D)?
|     |       +--yes:[selected] 12.0 [[selected:12.0,rejected:0.0]]
|     |       +--no: [rejected] 8.0 [[selected:0.0,rejected:8.0]]
|     +--no:  [rejected] 40.0 [[selected:0.0,rejected:40.0]]
+-no: candidate_turn_to(A,B,C,D,E),target_goal_have_image(A,B,E,-I)?
      +--yes: target_goal_have_image(A,B,D,-J) ?
      |       +--yes:[rejected] 48.0 [[selected:0.0,rejected:48.0]]
      |       +--no: [selected] 18.0 [[selected:18.0,rejected:0.0]]
      +--no:  [selected] 222.0 [[selected:220.0,rejected:2.0]]
```

Figure 6: Relational decision tree learned for the bindings selection of the turn_to action from the satellite domain.

Planning with Decision Trees

In this section we explain how we use the learned decision trees directly as general action policies (the H-Context Policy algorithm) or as control knowledge for a heuristic planner (the sorted EHC algorithm).

H-Context General Policy

The *helpful context*-action policy is applied forward from the initial state as described in the pseudo-code of Figure 7. The algorithm goes as follows. For each state to be expanded, its helpful actions and target goals are computed. Then, the action decision tree determines which action should be applied using the current helpful context. The leaf of the decision tree returns *action-list*, a list of actions sorted by the number of examples matching the leaf during the training phase (see Figure 3). From this list we keep only actions that have at least one matching example. With the first action in *action-list*, we give its ground applicable actions to the corresponding bindings tree. The leaf of the binding tree returns the number of times that the decision was selected or rejected in the training phase.

```
Depth-First H-Context Policy $(I, G, T)$: plan
------------------------------------------------
$I$: initial state
$G$: goals
$T$: (Policy) Decision Trees
------------------------------------------------
open-list $= \emptyset$ ; delayed-list $= \emptyset$ ;$s = I$
while open-list $\neq \emptyset$ and delayed-list $\neq \emptyset$
                          and not solved$(s, G)$ do
  if $s$ not in path$(I, s)$ then
    $S'$ = helpful-successors$(s)$; candidates= $\emptyset$
    action-list = solve-op-tree$(s, T)$
    for each action in action-list
      $C'$ = nodes-of-operator($S'$,action);
      for each $c'$ in $C'$
        ratio$(c')$ = solve-binding-tree$(c', T)$
      candidates = candidates $\cup$ sort($C'$,ratio$(c')$)
    open-list = candidates $\cup$ open-list
    for each action not in action-list
      $C'$ = nodes-of-operator($S'$,action)
      delayed-list= $C'$ $\cup$ delayed-list
  if open-list $\neq \emptyset$ then
    $s$ = pop(open-list)
  else
    $s$ = pop(delayed-list)
if solved$(s, G)$ then
  return path$(I, s)$
```

Figure 7: A depth-first algorithm with a sorting strategy given by the helpful-context policy.

We use the selection ratio, selected/(selected + rejected), to sort the ground actions. Then, they are inserted in *candidates* list. We repeat the ground action sorting for each action in *action-list* and finally *candidates* list is placed at the beginning of *open-list*. As a result we apply a depth-first strategy in which a backtrack-free search is the exact policy execution. We keep track of the repeated states, so if the policy leads to a visited state the algorithm discards that node and takes the next one in the *open-list*. Ground actions whose action does not appear in *action-list* are placed in *delayed-list*. Nodes in *delayed-list* are used only if *open-list* becomes empty, to make the search complete in the helpful action search tree.

The main benefit of this algorithm is that it can handle the search process regardless of the robustness of the policy. A perfect policy will be directly applied. But an inaccurate one will be applied, recovering with backtracking when needed. States in the path are evaluated with the heuristic function just for computing the helpful context, but the heuristic values are not used.

Search Control Knowledge

We also have used learned decision trees as control knowledge for the heuristic search algorithm. We use them as a node ordering technique for guiding EHC. Given that EHC skips siblings evaluation once it finds a node that improves the heuristic value of its parent, evaluating the successor nodes in the right order (better to worse) can save a lot of computation. At any state in the search tree, we compute the helpful context, and then sort its successors (candidate actions) using the order by which they appear in the decision tree leaf that matches the current context. Ground actions of the same action are also sorted, evaluating them in the order given by their selected/rejected ratio obtained in the bindings tree match.

Experimental Results

We tested the two algorithms on seven IPC domains classified as different in the topology induced by the heuristic function (Hoffmann 2005). These domains are the Blocksworld, Miconic and Logistics from the IPC-2 set, Zenotravel from IPC-3, Satellite from IPC-4 and Rovers and TPP from IPC-5. For each domain we generated 30 random problems as training set using the IPC random problem generators. The training problems were solved with EHC, refined with DfBnB, and then the solutions obtained for each problem were compiled into learning examples, as explained in the *Generation of Learning Examples* section. From these examples ROLLER builds the corresponding decision trees with the TILDE system. The resulting trees are loaded in memory. Finally ROLLER attempts to solve each test problem of the corresponding STRIPS IPC problem set with a time bound of 1800 seconds.

Training

In order to evaluate the efficiency of our training process we measured the time needed for solving the training problems, the number of learning examples generated in this process and the time spent by TILDE for learning the decision trees. Table 1 shows the results obtained for each domain.

Table 1: Experimental Results of the training process.

Domain	Training Time	Learning Examples	Learning Time
BlocksWorld	21.30	250	6.05
Miconic	50.83	4101	21.94
Logistics	268.67	965	11.58
Zenotravel	530.13	1734	212.53
Satellite	13.18	2766	40.14
Rovers	850.68	384	21.71
TPP	277.48	644	5.29

ROLLER achieves shorter learning times than the state-of-the-art systems for learning general policies (Martin & Geffner 2004; Yoon, Fern, & Givan 2007). Particularly, while these systems implement ad-hoc learning algorithms that often need hours to obtain good policies, our approach only needs seconds to learn the decision trees for a given domain. This feature makes our approach more suitable for architectures that need on-line planning and learning processes. However, these learning times are not constant in the different domains because they depend on the number of learning examples (in our work this number is given by the amount of different solutions for the training problems) and on the size of the learning examples (in our work this size is given by the number of predicates and actions in the planning domain and their arity).

Testing

To evaluate the correctness of the learning process, we used the resulting decision trees first, as a general action policy and second, as a search control ordering technique for EHC. We compared the performance of both approaches with EHC. These three configurations are named as follows:

- **EHC**: an EHC implementation with the FF's heuristic and the helpful actions as a pruning technique (Hoffmann & Nebel 2001). This configuration serves as a baseline for comparison as used in (De la Rosa, García-Olaya, & Borrajo 2007).
- **H-Context Policy**: directly using the learned decision trees with the Depth-first H-Context Policy algorithm.
- **Sorted-EHC**: sorting-based EHC that sorts node successors based on the order suggested by the decision trees.

The evaluation of the learned decision trees is performed both in terms of problems solved and quality of the solutions found. Table 2 shows the number of problems solved for each of the three planning configurations in all the domains. Table 3 shows the average values of computation time (in seconds), plan length and evaluated nodes, only in the problems solved by the three configurations. Note that for the blocksworld domain we used the problems from the track 2 of IPC2 which was a harder test set not even tried by many competitors.

Table 2: Problems solved by the three configurations.

Domain (problems)	EHC	H-Context Policy	Sorted-EHC
Blocksworld (103)	20	103	21
Miconic (150)	150	150	150
Logistics (79)	72	79	73
Zenotravel (20)	19	20	19
Satellite (36)	23	28	26
Rovers (40)	31	40	33
TPP (30)	19	30	19

Regarding the experimental results displayed in Table 2 the planning configuration based on the direct application of the learned decision trees as a general policy solves all the IPC problems in all domains except in one, the Satellite domain. Though the policy learned in this domain is effective, the last 8 problems have a very large size (from 100 to 200 goals) so the computation of the *helpful actions* along the solution path is too expensive for a time bound of 1800 seconds. On the other hand, the ordering of node evaluation in EHC solves only a few more problems than the standard EHC. The reason is that this technique still relies on the accuracy of the heuristic function, so where the heuristic is uninformed, evaluation of useless nodes still happens like in the standard EHC.

In terms of plan length table 3 shows that the direct application of the learned policy has two different types of behavior. On one hand, in domains with resource management such as logistics or zenotravel, the quality of the plans found by these configuration got worse. In these domains our definition of *helpful context* is not enough to always capture the best action to make because the arguments of the best action do not always correspond to the goals of the problem or the static predicates. On the other hand, in domains with no resource selections, like the blocksworld or miconic, the direct application of the general policy improves the quality performance of the baseline EHC algorithm. Otherwise, the sorted-EHC configuration keeps good quality values in all the domains because the deficiencies of the policy in domains with resources are corrected by the heuristic function.

In terms of computation time, the *H-Context Policy* configuration scales much better than the standard EHC algorithm. Figures 8, 9 and 10 show, in a logarithmic scale, the evolution of the accumulated time used to solve the problems in the Miconic, Logistics and Rovers domain respectively. In these graphs we can also see that the simple EHC algorithms achieve better time values when solving problems that require computation times lower that one second. This is because the use of the learned tree for planning involves a fixed time cost for matching helpful contexts, and this process is not performed by the standard EHC. We omitted graphs in the rest of domains because the behavior is similar to the three ones taken as example.

In the BlocksWorld domain heuristic planners scale poorly because there is a strong interaction among the goals that current heuristics are unable to capture. Particularly in this domain achieving a goal separately may undo others therefore, it is crucial to achieve the goals in a particular order. The actions trees learned by our approach give a total order of the domain actions in the different contexts capturing this information. This fact makes our approach achieve impressive scalability results producing good quality solution plans.

Related Work

Our approach is strongly inspired by the way Prodigy (Veloso *et al.* 1995) models the search control knowledge. In the Prodigy architecture, the action selection is a two-step process: first, the best uninstantiated operator/action to apply is selected, and, second, the bindings for the operators are selected. Control knowledge could be specified or learned (Leckie & Zukerman 1998; Minton 1990) for guiding both selections. We have

Table 3: Average values of the planning time, plan length and evaluated nodes in the problems solved by all the configurations.

Domain	EHC			H-Context Policy			Sorted-EHC		
	Time	Quality	Evaluated	Time	Quality	Evaluated	Time	Quality	Evaluated
Blocksworld	50.29	41.16	4401.50	0.48	34.66	37.83	30.20	47.3	4293.83
Miconic	7.21	69.49	341.10	1.62	51.37	52.37	7.35	55.78	157.97
Logistics	172.23	110.00	1117.84	18.63	131.30	132.25	183.50	108.48	1140.05
Zenotravel	82.26	32.31	407.26	7.06	55.68	56.68	115.51	33.67	554.21
Satellite	48.69	42.60	398.47	3.66	43.95	44.91	9.58	42	112
Rovers	70.28	51.32	691.25	4.96	52.29	53.38	45.14	50.93	447.87
TPP	108.08	62.45	2631.77	1.88	46.00	46.86	104.09	62.68	2649.40

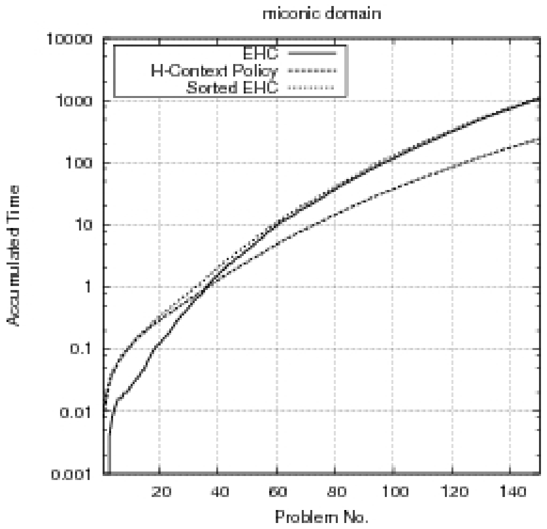

Figure 8: Accumulated time in the Miconic domain.

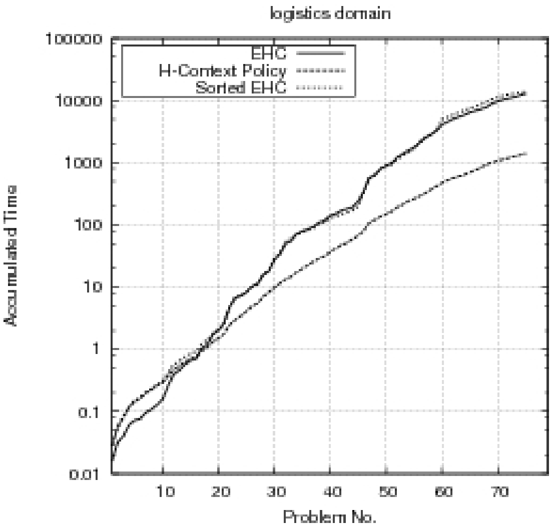

Figure 9: Accumulated time in the Logistics domain.

translated this idea of the two-step action selection into the heuristic planning paradigm, because it allows us to define the process of learning planning control knowledge as a standard classification process. Nevertheless, unlike Prodigy, our approach does not distinguish among different node classes in the search tree.

Relational decision trees have been previously used to learn action policies for the relational reinforcement learning task (Dzeroski, De Raedt, & Blockeel). In comparison to our work, this approach presented two limitations. First, the learning is targeted to a given set of goals, therefore they do not directly generalize the learned knowledge for different goals within a given domain. And, second, since they use an explicit representation of the states to build the learning examples they need to add extra background knowledge to learn effective policies in domains with recursive predicates such as the blocksworld.

Recent work on learning general policies (Martin & Geffner 2004; Yoon, Fern, & Givan 2007) overcome these two problems by introducing the planning goals to the background knowledge of the learning examples and changing the representation language for the examples from predicate logic to concept language. Our approach is an alternative to these works, because learning the action to choose among the *Helpful actions* allows us to obtain effective general policies without changing the representation language. As a consequence, we can directly use off-the-shelf relational classifiers that work in predicate logic so the resulting policies are easy readable and the learning times are shorter.

Conclusions and Future Work

We have presented a technique for reducing the number of node evaluations in heuristic planning based on learning *Helpful context*-action policies with relational decision trees. Our technique defines the process of learning general policies as a two-step classification task and builds domain-specific relational decision trees that capture the action to select in the different planning contexts. We have explained how to use the learned trees to solve classical planning problems, applying them directly as a policy or as search control for ordering the node evaluation in EHC.

This work contributes the state-of-the-art of learning based planning in three ways:

1. Representation. We propose a new representation for learning general policies that encodes in predicate logic the meta-state of the search. As oppose to previous works based on learning in predicate logic (Khardon 1999), our

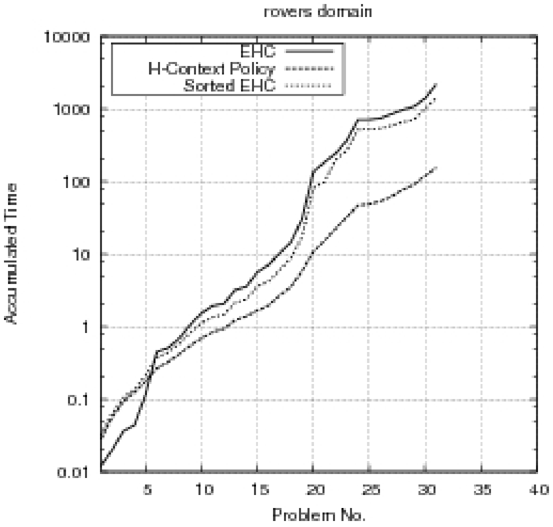

Figure 10: Accumulated time in the Rovers domain.

representation does not need extra background to learn efficient policies for domains with recursive predicates such as the blocksworld. Besides, since our policies are extracted from the relations of the domain actions, the resulting learned trees provide useful hierarchical information of the domain that could be used in hierarchical planning systems (the actions tree gives us a total order of the actions according to the different planning contexts).

2. Learning. We have defined the task of acquiring planning control knowledge as a standard classification task. Thus, we can use an off-the-shelf classifier for learning the *helpful context*-action policy. Results in the paper are obtained learning relational decision trees with the TILDE tool, but we could have used any other relational classifier. Because of this fact, advances in the field of relational learning can be directly applied to learn faster and better control knowledge for planning.

3. Planning. We introduced two methods for using policies to reduce the node evaluation: (1) the novel algorithm *Depth-First H-Context Policy* that allows a direct application of the H-Context policies and (2) the *sorted EHC*, an algorithm that allows the use of a general policy with greedy heuristic search algorithms traditionally effective in heuristic planning such as EHC.

The experimental results showed that our approach improved the scalability of the baseline heuristic planning algorithm EHC over a variety of IPC domains. In terms of quality, our approach generates worse solutions than the baseline in domains with resources management like the Zenotravel or Logistics. In these domains our definition of the *helpful context* is not enough to always capture the best action to apply because the arguments of the best action do not always correspond to the problem goals or the static predicates. We plan to refine the definition of the *helpful context* to achieve good quality plans in such domains too. We also want to test the integration of the learned trees with other search algorithms suitable for policies such as *Limited Discrepancy Search*. Furthermore, given that the learning of decision trees is robust to noise, we want to apply this same approach to probabilistic planning problems. Finally, for the time being we are providing the learner with a fixed distribution of learning examples. In the near future, we plan to explore how to generate the most convenient distribution of learning examples according to a target planning task.

Acknowledgments

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05 and regional CAM-UC3M project CCG06-UC3M/TIC-0831.

References

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

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. *JAIR* 24:581–621.

Coles, A., and Smith, A. 2007. Marvin: A heuristic search planner with online macro-action learning. *JAIR* 28:119–156.

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 CBR*, 137–148.

Dzeroski, S.; De Raedt, L.; and Blockeel, H. Relational reinforcement learning. In *International Workshop on ILP*.

Fern, A.; Yoon, S. W.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *JAIR* 25:85–118.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J. 2005. Where ”ignoring delete lists” works: Local search topology in planning benchmarks. *JAIR* 24:685–758.

Khardon, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence* 113:125–148.

Leckie, C., and Zukerman, I. 1998. Inductive learning of search control rules for planning. *Artificial Intelligence* 101(1–2):63–98.

Martin, M., and Geffner, H. 2004. Learning generalized policies from planning examples using concept languages. *Appl. Intell* 20:9–19.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *Artif. Intell.* 42(2-3):363–391.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *ICAPS*.

Quinlan, J. 1986. Induction of decision trees. *Machine Learning* 1:81–106.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *JETAI* 7(1):81–120.

Xu, Y.; Fern, A.; and Yoon, S. W. 2007. Discriminative learning of beam-search heuristics for planning. In *IJCAI 2007, Proceedings of the 20th IJCAI*, 2041–2046.

Yoon, S.; Fern, A.; and Givan, R. 2006. Learning heuristic functions from relaxed plans. In *ICAPS*.

Yoon, S.; Fern, A.; and Givan, R. 2007. Using learned policies in heuristic-search planning. In *Proceedings of the 20th IJCAI*.